Berücksichtigung herstellungsbedingter Oberflächenstrukturen auf die Schwingfestigkeit von Motorkomponenten in der virtuellen Bauteilauslegung

DIPLOMARBEIT

Markus TAUSCHER

Montanuniversität Leoben
Lehrstuhl für Allgemeinen Maschinenbau
Leiter: Univ.-Prof. Dipl.-Ing. Dr.mont. Florian Grün

Betreuer:
Assoz.-Prof. Dipl.-Ing. Dr.mont. Michael Stoschka
Dipl.-Ing. Christian Garb

Leoben, September 2017

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Leoben, am 15.09.2017 Markus Tauscher

Sperrvermerk

Diese Arbeit beinhaltet vertrauliche Informationen, die der Geheimhaltung unterliegen. Aus diesem Grund darf die Arbeit ohne vorherige schriftliche Zustimmung der BMW AG nicht vervielfältigt oder veröffentlicht werden. Dieser Zustimmungsvorbehalt endet am: 25.09.2022

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die positiv an der Entstehung dieser Diplomarbeit mitgewirkt haben und mir bei den Herausforderungen unterstützend zur Seite gestanden sind.

Seitens der Montanuniversität möchte ich als erstes

Univ.-Prof. Dipl.-Ing. Dr.mont. Florian Grün,

dem Vorstand des Instituts für Allgemeinen Maschinenbau, danken, der die Durchführung dieser Diplomarbeit am Institut ermöglicht hat.

Für die Betreuung dieser Diplomarbeit, als auch zahlreicher fachlicher Diskussionen und Optimierungsvorschlägen möchte ich mich bei

Assoz.-Prof. Dipl.-Ing. Dr.mont. Michael Stoschka

und

Dipl.-Ing. Christian Garb

bedanken.

Besonderer Dank für die zahlreichen fachlichen Diskussionen und Anregungen gilt Herrn

Dipl.-Ing. Dr.tech. Robert Ehart

der auch an der Ermöglichung dieser Arbeit entscheidend beteiligt war.

Weiters möchte ich mich bei Herrn **Dipl.-Ing. Dr.mont. Paul Kainzinger** und Frau **Dipl.-Ing. Dr.mont. Sabine Redik** bedanken, die mich in der Anfangsphase dieser Arbeit maßgebend unterstützten.

Ein besonderer Dank aus Herzen gilt meinen Eltern, die mir das Studium ermöglichten und mich in dieser Zeit immer unterstützt haben.

Kurzfassung

Im Entwicklungsprozess spielt die virtuelle Bauteilauslegung heutzutage eine große Rolle und wird in Zukunft weiter an Bedeutung gewinnen. Für eine möglichst exakte Vorhersage der Betriebsfestigkeit von auszulegenden Komponenten, ist die Verbesserung von Modellen zur Beschreibung von fertigungstechnologischen Einflüssen auf die Lebensdauer unerlässlich. Einen wesentlichen Parameter stellt der Rauhigkeitseinfluss an technischen Oberflächen dar. Ziel dieser Arbeit war es eine methodische Vorgehensweise zu entwickeln, welche ausgehend von einer durch Messung bestimmten dreidimensionalen Oberflächentopologie den Rauhigkeitseinfluss mittels eines lokalen Konzepts beschreiben kann. Als Eingangsparameter sollten dabei nur bereits vorhandene, grundlegende Werkstoffparameter zur Beschreibung des Dauerfestigkeitseinflusses Verwendung finden.

Im ersten Schritt wird ein effizientes Verfahren vorgestellt um aus dreidimensionalen Messpunkten eine triangulierte Oberfläche erzeugen zu können. Mittels Gaußfilter werden zunächst Frequenzbereiche eliminiert, welche auf die Lebensdauer keinen Einfluss mehr haben aber die weitere numerische Verarbeitung der Oberfläche stark erschweren würden. Anschließend wird noch die Formabweichung korrigiert um geometriebedingte Einflüsse ausschließen zu können. Danach werden mittels eines dreidimensionalen linear elastischen Finite Elemente Modells die auftretenden Spannungen bei Standardlastfällen (Zug 0°, Zug 90°, Schub) ermittelt. Die Netzgröße hat dabei einen hohen Einfluss auf die Ergebnisqualität und die Rechenzeit. Um diesen Zielkonflikt bestmöglich zu lösen ermöglicht ein benutzerdefiniertes Verfahren die Vorgabe der Netzfeinheit je nach Oberflächenkrümmung. Abschließend erfolgt zur Beschreibung der möglichen Vorzugsorientierung der Oberfläche und zur statistischen Bewertung der Mikrokerben eine lineare Spannungsüberlagerung der Standardlastfälle um die multiaxiale Beanspruchung der Oberfläche abzubilden.

Diese umfangreiche, FE-basierte Methodik liefert eine Aussage über den zu erwartenden lokalen Abminderungsfaktor zur Dauerfestigkeit, welcher mithilfe eines Spannungsmittelungskonzeptes für jeden Oberflächenknoten berechnet wird. Die dazu erforderlichen Spannungsverläufe werden durch das Superconvergent-Patch-Recovery Verfahren bestimmt. Die Mittelungslänge ergibt sich dabei aus dem Langrissschwellwert und der Dauerfestigkeit des defektfreien Materials.

Zur Bewertung der Kerbwirkung werden die lokale Minima der Abminderungsfaktoren über die gesamte, zu bewertende Oberfläche bestimmt. Jedes dieser lokalen Minima grenzt sich dabei durch einen definierten Schwellwert von anderen ab. Die relative Häufigkeit dieser Abminderungsfaktoren beschreibt den statistischen Lebensdauereinfluss der Oberflächenrauheit und stellt somit auch einen statistischen Größeneinfluss dar. Diese umfassende numerische Methodik ermöglicht eine neuartige Bewertung der Kerbwirkung von Oberflächentopographien, welche für eine gussraue Oberfläche experimentell validiert wurde.

Abstract

In the modern development process, the virtual component design plays a decisive role and a further increase in importance is expected. Hence, the improvement of methods describing the manufacturing process related fatigue strength is mandatory. Surface roughness plays a vital role on describing fatigue strength of various components. The aim of this work was to develop a methodology, which gives an enhanced understanding of the local surface roughness effect, based on measured three dimensional surfaces topologies. The input parameters of the developed method should necessitate only a small number of well-known material parameters to avoid costly material tests or complex additional measurements.

The investigation starts with an improved method to get a triangulated surface from a measured, spatial point cloud. Such a measurement data contains high frequency signals, which have no influence on the fatigue strength, but makes the further processing of the surface data unnecessarily difficult. Thus, a digital Gauss filter was applied on the measurement data. To suppress the influence of the measured macroscopic surface waviness, the shape deviation is additionally corrected. The resulting surface data is integrated in a three dimensional linear elastic finite element model. Three standard load cases are set-up including two tensile one shear load step. The mesh size of the finite element model has an utmost impact on either the quality of the results and on the calculation time. Areas of comparably high stress levels also possess an increased surface curvature value. Therefore, a user-defined method is developed to determine the mesh size based on the maximum local curvature. To study the preferred roughness orientation and the effect of micronotches within the surface, a linear superposition of the three standard load cases is applied. This features also a multiaxial evaluation.

This comprehensive methodology facilitates numerically based fatigue strength reduction factors for each surface node, which are based on the critical distance theory. The corresponding stress evaluation paths perpendicular to the surface are evaluated by using an advanced Superconvergent Patch Recovery approach. The critical distance length is based only on the fatigue strength and the crack propagation threshold of the defect free material. This meets the desired goal of a minimized number of material input parameters within this approach.

The roughness effect under cyclic loading is locally assessable by these novel reduction factors for each surface node. Each of these local minima regions are bounded by certain threshold limit values to their adjacent regions. The relative occurrence of these local minima describes the influence as roughness fatigue factor and additionally enables a statistical size effect of the rough surface.

The presented novel approach is numerically effective and therefore applicable for different surface conditions. A comparison of the developed roughness evaluation method against a rough surface of an aluminium casting concludes the work.

Inhaltsverzeichnis

1. Einleitung

Die Automobilindustrie wird von immer strengeren Flottenverbrauchsgrenzwerten und kürzer werdende Produktzyklen vor bedeutende Herausforderungen gestellt. Da die Fahrzeugmasse einen wesentlichen Einfluss auf den Energiebedarf zur Zurücklegung einer definierten Strecke, und daraus folgend auch dem Verbrauch, hat ist die Gewichtsoptimierung von Fahrzeugkomponenten eine der ständigen Zielsetzungen im Auslegungsprozess. Darüber hinaus erfordern auch die kürzer werdenden Produktzyklen einen verbesserten Auslegungsprozess unter Berücksichtigung des Leichtbaupotentials.

Viele Motor- und Antriebsstrangkomponenten können aufgrund ihrer sehr komplexen Geometrie in großer Mehrheit nur durch Gießverfahren wirtschaftlich hergestellt werden. Um die Masse möglichst gering zu halten, kommen dabei meist Aluminiumlegierungen zum Einsatz. Mit dem Herstellungsprozess gehen aber auch untrennbar variierende Materialeigenschaften, wie Defektverteilungen oder auch Oberflächenqualitäten einher.

Die Zuverlässigkeit der aus der virtuellen Bauteilauslegung erhaltenen Ergebnisse und damit auch die erforderliche Sicherheitsspanne bei der Dimensionierung, hängt neben der Kenntnis lokaler Materialeigenschaften stark von der korrekten Modellierung von relevanten Einflussparametern ab. Höchstbelastete Bereiche bei Motorkomponenten aus Aluminiumgusswerkstoffen wie dem Zylinderkurbelgehäuse liegen üblicherweise an der Oberfläche. Bei der Bewertung der Schwingfestigkeit der Randschicht ist dabei der Einfluss der Oberflächentopographie bzw. der Rauheit von entscheidender Bedeutung.

Bisher angewendete Modelle beschreiben den Einfluss der Rauheit auf das Schwingfestigkeitsverhalten des Bauteils im Wesentlichen durch vereinfachte Abminderungsfaktoren, die auf Rauheitsparametern wie der gemittelten Rautiefe (Rz) und der verwendeten Werkstoffgruppe basieren. Die eigentliche Oberflächenmorphologie wird in den vorhandenen, empirischen Modellen allerdings nur indirekt abgebildet. Betrachtet man die Rauheit beispielsweise als eine Aneinanderreihung von Kerben, so ist die Form dieser Kerben für die gemittelte Rautiefe nicht relevant, obwohl ein Einfluss auf die Schwingfestigkeit zu erwarten wäre. Zusätzlich basieren die benötigten Werkstoffparameter nicht auf allgemeinen technologischen Größen, sondern werden empirisch für die jeweilige Materialgruppe ermittelt. Diese vereinfachte Herangehensweise macht die Anwendung von höheren Sicherheitsfaktoren notwendig, die zu einer Überdimensionierung von Bauteilen führen.

1.1. Aufgabenstellung und Lösungsansatz

Es soll ein Modell entwickelt werden um, ausgehend von dreidimensionalen Geometriedaten einer Oberfläche, den Rauhigkeitseinfluss auf das Schwingfestigkeitsverhalten verbessert beschreiben zu können. Das Modell sollte dabei möglichst nur auf definierte Materialparameter zurückgreifen müssen, um eine universelle Einsetzbarkeit zu garantieren. Des Weiteren soll der Einsatz an Personenstunden bei der Beschreibung einer neuen Oberfläche möglichst minimiert werden. Ein hoher Automatisierungsgrad der Routinen ist deshalb unumgänglich.

Großes Augenmerk soll auch auf eine effiziente Umsetzung gelegt werden. Die Rechenzeit zur spannungsmechanischen Bewertung sollte mit derzeit zur Verfügung stehenden Ressourcen einige Stunden je Oberfläche nicht überschreiten. Der nachfolgende Auswerteprozess zur numerischen Bewertung der Schwingfestigkeit sollte ebenfalls nur eine begrenzte Rechenzeit benötigen.

Um diese Anforderungen erfüllen zu können wurde ein Konzept welches auf bruchmechanischen Werkstoffparametern basiert entwickelt. Für Defekte wie zum Beispiel Poren existieren bereits bewährte bruchmechanische Konzepte zur Bestimmung der lokalen Schwingfestigkeit bei bekannter Porosität [ETS79; Mur02]. Diese Konzepte basieren im Wesentlichen auf der Erkenntnis, dass an Defekten bei geringer Belastung zwar kurze Risse initiieren, diese jedoch aufgrund der mit zunehmendem Abstand vom Defekt abnehmenden Spannungsintensität nicht in das Langrisswachstum übergehen. Da die Oberfläche ebenfalls als Ansammlung von Mikrokerben betrachtet werden kann, erscheint es zweckmäßig ähnliche Ansätze zur Beschreibung der Oberflächenrauheit anzuwenden.

Der Lösungsansatz gliedert sich somit in folgende Themenbereiche

- Triangulation der gemessenen Punktewolke
- Erstellung eines FE- tauglichen Oberflächennetzes
- Erstellung eines modularen, dreidimensionalen Modells in die unterschiedlichen Oberflächen schnell implementiert werden können
- Berechnen von Standardlastfällen und Überlagern dieser zu Zuglastfällen unterschiedlicher Richtung
- Berechnung des Hauptnormalspannungsverlaufs senkrecht zur Referenzoberfläche ausgehend von jedem Oberflächenknoten
- Ermittlung des Mittelwertes für jeden Spannungsverlauf über eine sich aus Materialparametern ergebende Distanz und Berechnung des Abminderungsfaktors
- Finden von lokalen Minima der betreffenden Abminderungsfaktoren auf der Oberfläche, wobei diese durch einen vorgegebenen Schwellwert abgegrenzt sind
  Ausgabe des kritischen Abminderungsfaktors für jedes erkannte lokale Minimum der zugrundeliegenden statischen Abminderungsfaktorenverteilung der Oberflächen

2. Stand der Technik

2.1. Verfahren zur Lebensdauerberechnung

Der Betriebsfestigkeitsnachweis von Bauteilen kann sowohl auf experimentelle als auch auf rechnerische Art durchgeführt werden, wobei bei letzterem Ansatz die Ermüdungsfestigkeit, neben den Beanspruchungsarten und dem zugrundegelegtem Schädigungsmodell auf experimentell bestimmten Werkstoffparametern sowie interagierenden Einflussfaktoren wie Größeneffekt, höchstbeanspruchtem Volumen, Verteilung von Gefügemerkmalen, Oberflächeneigenschaften sowie Umgebungseinflüssen basiert.

Sofern die auftretenden Belastungen am Prüfstand korrekt darstellbar sind, bietet der rein experimentelle Ansatz vor allem bei sich komplex überlagernden Lastsituationen und vielen zu berücksichtigenden Einflussfaktoren den Vorteil einer hohen Zuverlässigkeit. Er wird daher häufig am Ende eines Entwicklungsprozesses eingesetzt um den Betriebsfestigkeitsnachweis zu erbringen.

Im Laufe des Entwicklungsprozesses werden rein experimentelle Verfahren aus folgenden Gründen jedoch möglichst vermieden.

- Hohe Kosten durch komplexen Prüfaufbau als auch durch Prototypenfertigung
- Hoher Zeitaufwand
- Dadurch nur eine sehr begrenzte Anzahl von konstruktiven Ansätzen prüfbar
- Es kann stets nur die kritischste Stelle erkannt und verbessert werden

Unter anderem aufgrund dieser Nachteile ist ein rechnerischer Betriebsfestigkeitsnachweis in der Entwicklung nach heutigem Standard unverzichtbar, da nur mit einem tieferen Verständnis des schädigungsrelevanten Werkstoffverhaltens unter vorgegebenen Belastungen ein technisch als auch wirtschaftlich zufriedenstellendes Bauteildesign erzielt werden kann.

Bemessungsprinzipien

SAFE LIFE – Bemessung:
Bei der Safe Life Auslegung ist sicherzustellen, dass das Bauteil während der gesamten geplanten Einsatzdauer, bzw. aus Sicherheitsaspekten ein Vielfaches davon, nicht versagt. Das technische Anrissstadium als auch das Makrorisswachstum sind dabei in diesem Konzept inkludiert.

LEAK BEFORE BREAK – Bemessung:

Dieses Bemessungsprinzip wird vor allem bei unter Druck stehenden Bauteilen angewandt. Durch entsprechende Auslegung muss sichergestellt werden, dass sich ein bevorstehendes Versagen stets durch Leckagen bemerkbar macht, oder durch Druckminderung sogar verhindern lässt. Ein Gewaltbruch darf dabei erst nach fortgeschrittenem Makrorisswachstum auftreten.

CRACK FREE – Bemessung:

Das technische Anrissstadium darf während der gesamten vorgesehenen Lebensdauer, aus Sicherheitsgründen üblicherweise ein Vielfaches davon, nicht überschritten werden. Dieses Auslegungskriterium hat in maschinenbaulichen Anwendungsfällen die größte Verbreitung.

DAMAGE TOLERANCE – Bemessung:

Die gesamte vorgesehene Lebensdauer muss im technischen Makrorisswachstum erreichbar sein. Dieses Konzept findet vor allem bei Strukturkomponenten im Flugzeugbau Anwendung. Bei technischen Inspektionen wird dabei die Risslänge dokumentiert und bewertet. Falls diese bis zur nächsten Inspektion einen kritischen Wert erreichen kann, wird das Bauteil ersetzt.

Im Folgenden wird der rechnerische Lebensdauernachweis an von Proben und Bauteilen zusammengefasst, wobei die unterschiedlichen Konzepte nach den verwendeten Eingangsparametern klassifiziert werden.

2.1.1. Nennspannungskonzept

Dem Nennspannungskonzept liegen aus der elementaren Festigkeitslehre berechnete Spannungen zu Grunde. Die tatsächliche Spannungsverteilung weicht je nach Gestalt des betrachteten Bauteils, vor allem an Kerben von der Nennspannung σ_N ab. Spannungskonzentrationen σ_K, wie sie beispielsweise an Ausrundungsradien von Wellenabsätzen auftreten werden mittels tabellierter Kerbformzahlen K_t ermittelt die ausschließlich von der Geometrie der Kerbe, der Belastungsart sowie bei homogenen Werkstoffverhalten von der Poisson Zahl υ abhängen.

$$K_t = \frac{\sigma_K}{\sigma_N} \qquad \text{Glg. (2.1)}$$

Die zyklisch ertragbaren Spannungsspitzen $\sigma_{aD,\,Kerbe}$ im Kerbgrund sind jedoch aufgrund der Stützwirkung n im Kerbgrund erhöht. Die Kerbwirkungszahl K_f beschreibt das Verhältnis der dauerfest ertragbaren Spannungsamplitude σ_{aD} zu der Ermüdungsfestigkeit der gekerbten Probe.

$$K_f = \frac{\sigma_{aD}}{\sigma_{aD,Kerbe}} \qquad \text{Glg. (2.2)}$$

$$n = \frac{\mathrm{K}_t}{K_f} \qquad \text{Glg. (2.3)}$$

Für die Beschreibung der Stützwirkung existieren unterschiedliche Konzepte, die sowohl auf spannungsmechanischen als auch auf bruchmechanischen Ansätzen basieren. Ingenieurmäßig anwendbare Konzepte sind z.B. in der FKM-Richtlinie [FOR12] zusammengefasst.

Die wesentliche Voraussetzung um das Nennspannungskonzept anwenden zu können besteht zum einen in der Berechenbarkeit der auftretenden Nennspannungen und zum anderen in der Verfügbarkeit tabellierter Kerbform- bzw. Kerbwirkungszahlen für auslegungsrelevante Kerben. Dies schränkt das Verfahren allerdings auf relativ einfache Bauteile, wie zum Beispiel Wellen mit geometrisch definierten Übergängen, ein.
Die größten Vorteile des Nennspannungskonzepts liegen in der einfachen Anwendbarkeit, als auch in der Verfügbarkeit von entsprechenden Datensätzen.

2.1.2. Örtliche Konzepte

Örtliche Konzepte basieren auf lokalen Spannngs- bzw. Dehungswerten, im Unterschied zum Nennspannungskonzept kann durch diese Verfahren auch bei geometrisch komplexen Bauteilen eine Schwingfestigkeitsbewertung erfolgen. Die benötigten Spannungswerte werden meist mit der Finite-Elemente Methode (FEM) ermittelt. Im Zuge einer FEM Auswertung können sowohl innere Lasten, z.B. in Form von Massenkräften, als auch äußere Belastungen berücksichtigt werden. Diese Ergebnisse, unter anderem Spannungs- und Dehnungstensorwerte, liegen sowohl an der Oberfläche als auch im Volumen an Integrationspunkten vor da diese die höchste Konvergenzordnung aufweisen [ZZ92]. Die Lebensdauerbewertung erfolgt jedoch häufig anhand von extrapolierten Knotenergebnissen, da sich Elementknoten im Unterschied zu Integrationspunkten auch an der Oberfläche des betrachteten Bauteils befinden, wo üblicherweise die höchsten Beanspruchungen auftreten. Auf die in der vorliegenden Arbeit angewendete Extrapolationsmethode, der Superconvergent Patch Recovery wird im Kapitel 5.1.2 detaillierter eingegangen.
In der FEM- Analyse von komplexen Geometrien wie Motorkomponenten kommen hauptsächlich dreidimensionale Solid Elemente zum Einsatz. Neben der korrekten Abbildung der Randbedingungen haben dabei auch der Diskretisierungsgrad bzw. die sich daraus einstellende Netzgröße, die Elementqualität und der Elementtyp entscheidenden Einfluss auf die Ergebnisqualität der numerischen Berechnung. Des Weiteren kann an der betrachteten Stelle nicht direkt vom Tensor auf den jeweiligen lokalen Anteil an Zug oder Biegespannungen, welche zum Beispiel beim Nennspannungskonzept zur Beschreibung der Stützwirkung verwendet werden, geschlossen werden. Auch kann beim lokalen Konzept im Gegensatz zum Nennspannungskonzept an für die Auslegung relevanten Kerben keine Bestimmung von Spannungskonzentrationsfaktoren und daraus abgeleiteten Kerbwirkungszahlen vorgenommen werden, da weder die zugrundliegenden Nennspannungen noch die genaue Belastungsart bekannt sind. Auch bei gleicher Belastung kann sich das Schwingfestigkeitsverhalten je nach Position durch Fertigungseinflüsse beträchtlich unterscheiden. Bei Aluminiumgussbauteilen haben beispielsweise je nach Position stark variierende Werkstoffeigenschaften wie der sekundäre Dendritenarmabstand und die Porosität entscheidenden Einfluss auf die lokale Schwingfestigkeit. Um die Vergleichbarkeit mit Versuchsdaten herzustellen werden daher für jeden Knoten lokale Wöhlerlinien generiert. Wichtige Einflussparameter dabei sind unter anderem:

- Mittelspannung
- Stützwirkung
- Randschicht (Oberflächentopographie, Eigenspannungen, Gefüge)
- Defekte (Poren, Einschlüsse)
- Lastfolge
- Temperatur
- korrosive Medien
- mehrachsige Belastungen

Bestehende Konzepte zur Bewertung von Gussoberflächen zielen derzeit nur darauf ab den Einfluss der Rauhigkeit unter einachsiger Zug- Druck Belastung zu erfassen. Dies kann jedoch auch auf Biegelastfälle, also Lastfälle mit einem globalen und nicht durch die Oberflächenkerben verursachten Gradienten, erweitert werden. Auf die Implementierung weiterer Einflussfaktoren wie der Lastfolge, der direkte Interaktion oberflächennaher Poren mit der Gussoberfläche sowie Eigenspannungen wurde in dieser Arbeit bewusst verzichtet um eine schrittweise, systematische Bewertung des Oberflächeneinflusses zu ermöglichen.

Stützwirkungskonzepte

Gemäß dem skizzierten Lösungsansatz wird die erfasste Oberfläche zuerst spannungsmechanisch bewertet. Daraus ergeben sich für jede Einzelkerbe unterschiedliche Spannungsüberhöhungen bzw. normiert auf die aufgebrachte Nominalspannung auswertbare Kerbformzahlen. Bei stark lokalisierten Spannungsüberhöhungen kommt es jedoch je nach Werkstoff zu Stützwirkungseffekten, welche den Einfluss einer Kerbe auf die lokale Schwingfestigkeit abmildern. Die Ursache der in Versuchen beobachteten Stützwirkung wird dabei, je nach Bewertungsmethode, auf unterschiedliche Weise begründet. Bei auf spannungsmechanischen Ansätzen aufbauenden Konzepten nach Eichelseder [Eic89], Neuber [Neu68] oder Peterson [Pet53] wird die Stützwirkung mit dem Überschreiten der werkstofflichen Fließgrenze in einem kleinen Materialbereich begründet, wobei die darunter liegende Schicht den Werkstoff stützt. Bei auf bruchmechanischen Überlegungen basierenden Verfahren werden häufig Mikrorisse angenommen, die erst ab einer gewissen Spannungsintensität wachsen, bzw. in das Langrisswachstum übergehen können. [WAD71] [KL92] [TAY 99].

Um die Stützwirkung zu beschreiben sind die folgenden Verfahren weit verbreitet.

- Spannungsgradientenansätze
- Spannungsabstandskonzepte
- Spannungsmittelungskonzepte
- Werkstoffvolumenansätze

Nachfolgend werden kurz ausgewählte Stützwirkungskonzepte vorgestellt und deren Vor- bzw. Nachteile im Hinblick auf die Oberflächenbewertung diskutiert.

Spannungsgradientenansätze

Diese Verfahren basieren im Wesentlichen auf der lokalen Oberflächenspannung und dem zugehörigen Spannungsgradienten. Beispielhaft sei an dieser Stelle das Gradientenkonzept nach Eichelseder erwähnt [Eic89]. Im Gegensatz zu anderen Gradientenmodellen ist bei diesem Konzept die Berechnung von Kerbformzahlen aufgrund von Nennspannung und Kerbform nicht notwendig. Es wird neben den Materialparametern nur die lokale Spannung mit dem zugehörigen spezifischen Spannungsgradienten χ benötigt.

$$\chi = \frac{1}{\sigma_k} \cdot \left(\frac{d\sigma}{dx}\right) \qquad \text{Glg. (2.4)}$$

Das Modell basiert auf dem Verhältnis der Wechselfestigkeiten unter Biegebeanspruchung σ_{bW} sowie unter Zug/Druck σ_{zdW} Beanspruchung. K_D stellt einen werkstoffabhängigen Materialparameter dar. Der Parameter b beschreibt die Probendicke. Die in diesem Konzept angeführten Zug-Druck und Biegung-Wechselfestigkeitskennwerte werden oftmals an Proben mit einem charakteristischen Durchmesser $b = 7{,}5\ mm$ ermittelt. Die am Bauteil auftretende Dicke Stützwirkung ergibt sich aus dem Verhältnis von relativen Spannungsgradienten zu dem für Biegung erzielten Gradienten der Probenversuche von zwei durch Probenbreite.

$$n = 1 + \left(\frac{\sigma_{bW}}{\sigma_{zdW}}\right) \cdot \left(\frac{\chi}{2/b}\right)^{K_D} \qquad \text{Glg. (2.5)}$$

Das Modell kommt unter anderem in der kommerziellen Post Processing Software FEMFAT zur Lebensdauerbewertung zum Einsatz.
Gradientenkonzepte, wie auch das dargestellt nach Eichelseder, stoßen bei sehr scharfen Kerben bzw. Rissen an ihre Grenzen, da in diesem Fall der Gradient gegen unendlich tendiert und keine sinnvollen Aussagen mehr möglich sind. Der Einsatz für die Oberflächenbewertung erscheint daher nur begrenzt möglich.

Spannungsmittelungsansätze

Neuber entwickelte bereits ab 1936 ein Spannungsmittelungskonzept, um den Lebensdauereinfluss von Kerben beschreiben zu können [Neu68]. Die Motivation bestand darin ein Modell zu entwickeln um die Spannung an scharfen Kerben korrekt zu berücksichtigen. In der linear elastischen Spannungsmechanik treten bei geringer werdenden Kerbradien immer höhere Spannungen auf, was jedoch nicht der Realität entsprechen kann. Er erkannte, dass sich ein Material nicht beliebig zerteilen lässt, sondern sich aus Bereichen definierter Größe zusammensetzt, über die eine aus einfacher linear elastischer Rechnung ermittelte Spannungswerte mitteln lassen.

Da zu dieser Zeit Computer zu strukturmechanischen Spannungsberechnung noch nicht zur Verfügung standen entwickelte er eine empirische Gleichung Glg. (2.6) um vom Kerbfaktor K_t ausgehend die Kerbwirkungszahl K_f berechnen zu können.

$$K_f = 1 + \frac{K_t - 1}{1 + \sqrt{\frac{\rho'}{\rho}}} \qquad \text{Glg. (2.6)}$$

Als Eingangsparameter werden die empirische Ersatzstrukturlänge ρ'und der Kerbradius ρ benötigt. Während der prinzipielle Lösungsansatz für die Bewertung der Oberflächenkerben vielversprechend erscheint, stellt die Notwendigkeit eines Kerbradius ein Hindernis dar. Die lokale Krümmung quer zur Beanspruchungsrichtung kann zwar bestimmt werden, ist allerdings nicht zwingend repräsentativ für eine unregelmäßig geformte Oberflächenkerbe.

In weiterführenden Arbeiten stellte Taylor [TAY99] einen Formalismus zwischen Spannungsabstands und Spannungsmittelungskonzepte her und definiert einen Zusammenhang zwischen grundlegenden bruchmechanischen Materialparametern mit der Ersatzstrukturgröße.

2.2. Bruchmechanische Bemessung

Bei Gussbauteilen haben neben der Gefügestruktur und der gewählten Legierung sowie deren Wärmebehandlung vor allem mikrostrukturelle Einflüsse eine entscheidende Bedeutung auf die zu erwartende Lebensdauer. Derartige Imperfektionen führen dazu das im Langzeitfestigkeitsbereich keine ausgeprägte Dauerfestigkeit auftritt. Daher sind lokale Auslegungsmethoden zu verwenden welche eine Bewertung der Restlebensdauer nach Auftreten eines initialen Anrisses, bzw. einer herstellungstechnisch basierten Porositätskenngröße als Defektmerkmal, ermöglichen. Diese bruchmechanisch basierte Auslegung von defektbehafteten Werkstoffen wird als Kitagawa-Diagramm beschrieben. Für die Erstellung eines gesamtheitlichen Konzeptes zur Lebensdauerbewertung ist es deshalb naheliegend einen derartigen Ansatz mit bruchmechanischem Hintergrund zu wählen.

Beanspruchungsarten

In der Bruchmechanik wird zwischen drei grundlegenden Beanspruchungsarten unterschieden.

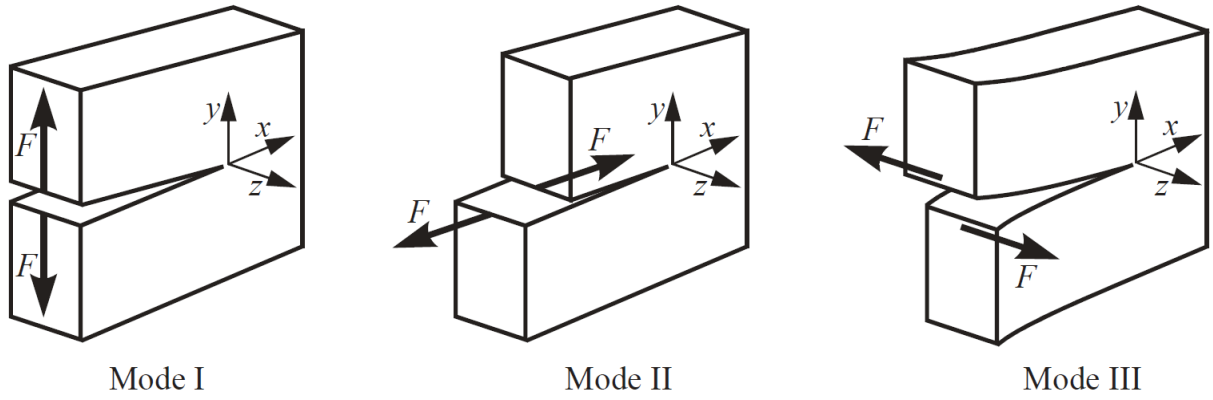


Abb. 2.1: Rissmoden in der Bruchmechanik

Es treten drei Rissöffnungsmoden je nach Belastungsrichtung zur Rissebene auf:

- Mode I: Beanspruchungen normal zur Rissebene
- Mode II: Schubbeanspruchung, die zum entgegengesetzten Gleiten der Rissebenen in der Ausbreitungsrichtung des Risses führt
- Mode III: Schubbeanspruchung, die zum Gleiten der Rissoberfläche quer zur Ausbreitungsrichtung führt

Die reale, an einem Bauteil wirkende Beanspruchung kann durch Superposition der drei dargestellten Beanspruchungsarten abgebildet werden (Mixed-Mode-Beanspruchung). Reine Mode I Belastung liegt vor, wenn die aufgebrachten Lasten in der Ebene normal zur Rissausbreitungsrichtung wirken. Die Rissausbreitung hängt dabei nur von der jeweils herrschenden Normalspannungsbelastung vor. Bei Mode II Belastung liegt ein ebener Schubspannungszustand vor, während bei Mode III ein nicht ebener Schubspannungszustand vorherrscht. Ein Beispiel für eine Mode III Belastung stellt ein normal zur Wellenachse wachsender Riss unter reiner Torsionsbeanspruchung dar. In den meisten Fällen reicht es aus nur den Mode I Belastungsfall zur betrachten, da sich Risse meist normal zur makroskopischen Hauptbeanspruchungsrichtung ausbreiten.
Bei rein elastischem Materialverhalten tritt an der Rissspitze eine Spannungssingularität auf. Ausgehend von der Rissspitze nimmt die mechanische Spannung mit der Quadratwurzel des Abstandes ab.

Zur Beschreibung des Spannungsfeldes an der Rissspitze kommt der Spannungsintensitätsfaktor K_I zur Anwendung.

$$K_I = \sigma \cdot Y \cdot \sqrt{\pi \cdot a} \qquad \text{Glg. (2.7)}$$

Dieser hängt von der anliegenden mechanischen Fernfeld-Spannung σ, der Risslänge a sowie dem Geometrieeinflussfakor Yab. Der Spannungsintensitätsfaktor ist nur in einer unendlich großen Platte unabhängig von den Probenabmessungen. Der Geometrieeinflussfaktor stellt dabei die Unabhängigkeit vom lokalen Spannungszustand und der Probenabmessungen her. Für dynamische Beanspruchung erhält man, unter der Voraussetzung dass die gesamte Spannungsschwingbreite effektiv an der Rissspitze wirkt, den zyklischen Spannungsintensitätsfaktor ΔK zu Glg. (2.7). Die zyklische Schwingbreite $\Delta\sigma$ beschreibt somit die doppelte Amplitude der anliegenden Spannung.

$$\Delta K = \Delta\sigma \cdot Y \cdot \sqrt{\pi \cdot a} \qquad \text{Glg. (2.8)}$$

Ermüdungsrisswachstum

Prinzipiell muss beim Ermüdungsrisswachstum zwischen kurzen und langen Rissen unterschieden werden. Kurze Risse wachsen bereits bei deutlich geringeren Spannungsintensitätsfaktoren als lange Risse. Die Unterscheidung zwischen langen und kurzen Rissen findet üblicherweise Aufgrund von Rissstadien statt. Im ersten Rissstadium hängt die Rissausbreitung stark von mikrostrukturellen Parametern wie der Korngröße oder Gleitbändern ab. Im Langrissstadium wächst der Riss nahezu unabhängig von mikrostrukturellen Gegebenheiten normal zur Hauptnormalspannungsrichtung.
Dauerfestigkeit liegt dann vor, wenn ein Riss nicht in das stabile Langrisswachstum übergehen kann, während Kurzrisswachstum toleriert werden kann. Der Spannungsintensitätsfaktor welcher den Übergang zwischen Kurz- und Langrisswachstum definiert, wird als Langrisschwellwert K_{th} bezeichnet und kann für ein Material mit definierten Gefügeeigenschaften und Wärmebehandlung mit überschaubarem Versuchsaufwand ermittelt werden.

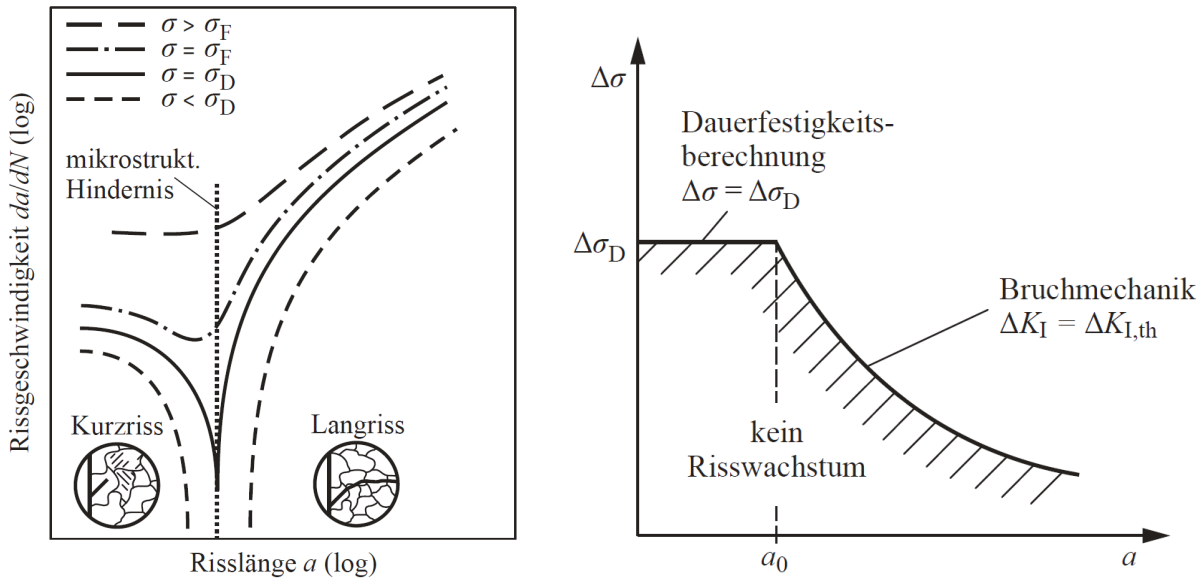


Abb. 2.2: Übergang von Kurz zu Langrisswachstum nach [KE92]

Durch Einsetzen der Spanne des Langrissschwellwertes ΔK_{th} und der Dauerfestigkeitsschwingbreite $\Delta\sigma_D$ erhält man durch umformen von Glg. (2.8) die charakteristische Mikrostrukturlänge a_0.

$$a_0 = \frac{1}{\pi} \cdot \left(\frac{\Delta K_{th}}{Y \cdot \Delta\sigma_D} \right)^2 \qquad \text{Glg. (2.9)}$$

Zur Schwellwertbeschreibung von Rissen wird an dieser Stelle auf zwei gängige Konzepte eingegangen, welche grafisch in Abb. 2.3 dargestellt werden. Die im Kitagawa-Takahashi-Diagramm eingezeichneten Grenzkurven trennen jeweils den dauerfesten Bereich, wo grundsätzlich kein Langrisswachstum zu erwarten ist, vom Bereich mit Risswachstum ab.

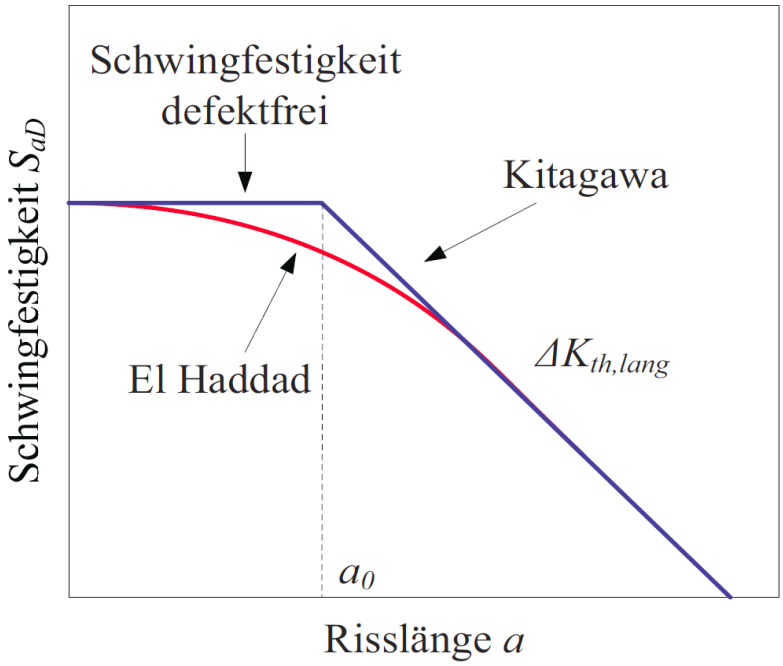


Abb. 2.3: Kitagawa-Takahashi-Diagramm nach [KIT76]

Kitagawa-Takahashi [KIT 76] beschreiben das Schwellwertverhalten von Rissen durch zwei Funktionen. Risse oder als Risse zu betrachtende Defekte welche kleiner als die Mikrostrukturgröße a0 sind haben keinen schwingfestigkeitsmindernden Einfluss. Für Risse oder Defekte die größer als die charakteristische Mikrostrukturlänge sind, wird die Schwellwertamplitude durch Glg. (2.10) beschrieben. Diese lässt sich aus Glg. (2.8) ableiten.

$$\Delta\sigma_D = \frac{\Delta K_{th,lang}}{Y \cdot \sqrt{\pi \cdot a}} \qquad \text{Glg. (2.10)}$$

El Haddad et al [ETS79] führt zur Beschreibung des Schwellwertverhaltens eine fiktive Rissverlängerung ein, welche der bereits erwähnten charakteristischen Mikrostrukturlänge a0 entspricht ein. Dies ermöglicht die Beschreibung der Grenzkurve des Kitagawa-Takahashi-Diagramms zu:

$$\Delta\sigma_D = \frac{\Delta K_{th,lang}}{Y \cdot \sqrt{\pi \cdot (a + a_0)}} \qquad \text{Glg. (2.11)}$$

Dies führt zu einem kontinuierlicheren Übergang vom Kurzrissverhalten zur bruchmechanischen Langrisslösung. In Bezug auf die vorliegende Arbeit muss an dieser Stelle die große Bedeutung für die Beschreibung des Schwingfestigkeitseinflusses von

Defekten in Gusswerkstoffen hervorgehoben werden. Eine wesentliche Motivation der vorliegenden Arbeit bestand darin, die Modellierung der Oberflächenrauheit möglichst konsistent mit der Beschreibung von Defekten darstellen zu können.

2.2.1. Konzept des kritischen Abstandes

In den bruchmechanischen Ansätzen sind Defekte und Risse hinsichtlich Schädigungswirkung als äquivalent anzusehen. Demzufolge lässt sich das zyklische Verhalten mit der Einführung von fiktiven Rissen anstelle von realen Defekten beschreiben, an denen ermittelt wird, ob die Schwingbeanspruchung den Spannungsintensitäts-Schwellwert bzw. die Dauerfestigkeit des Grundmaterials überschreiten und es somit zu einem zyklischen Risswachstum kommt. Wachstum ist dann möglich, wenn der Spannungsintensitätsfaktor an der Rissspitze höher als der Langrissschwellwert ist. Da auch Defekte im eigentlichen Sinn im Material eingebettete Kerben darstellen, ist es naheliegend das für scharfe Defekte mit Erfolg angewandte Konzept auf Kerben im Allgemeinen zu generalisieren. Taylor [TAY 99] vergleicht dazu das Spannungsfeld vor einer Rissspitze mit dem Spannungsfeld unter einer Kerbe. Die ermüdungswirksame Beurteilung kann zum Beispiel über ein Spannungsmittelungskonzept erfolgen. Dieser Ansatz ist somit als geeignet für die schwingfeste, ermüdungswirksame Bewertung von Oberflächen zu sehen.

Verallgemeinert lässt sich das Spannungsfeld vor einer Rissspitze mit Zug-Druckbelastung normal zur Rissspitze wie folgt darstellen. Die Risslänge a bezieht sich auf die Risstiefe eines Oberflächenanrisses oder die halbe Länge eines Risses im Inneren. Das elastische Spannungsfeld $\Delta\sigma(r)$ Spannung im Abstand r von der Rissspitze verändert sich nichtlinear.

$$\Delta\sigma(r) = \frac{\Delta\sigma}{\sqrt{1-\left(\frac{a}{a+r}\right)^2}} \qquad \text{Glg. (2.12)}$$

Für lange Risse ($r<<a$) kann Glg. (2.12) vereinfacht werden.

$$\Delta\sigma(r) = \Delta\sigma\sqrt{\frac{a}{2r}} \qquad \text{Glg. (2.13)}$$

Wenn die zyklische Spannungsintensität eines Risses gleich dem Schwellwert ist, d.h. $\Delta K = \Delta K_{th}$, dann ist gemäß der angegebenen Näherungsformel für lange Risse die Spannung im Abstand $a/2$ gleich der dauerfest ertragbaren Spannungsschwingbreite $\Delta\sigma_0$. Dieser Zusammenhang wird als Punktmethode nach Taylor bezeichnet.

Durch Einsetzen von Glg. (2.9) in Glg. (2.13) und der Mittelwertbildung durch Integration über die doppelte charakteristische mikrostrukturelle Größe lässt sich zeigen, dass im Grenzfall ab dem Langrisswachstum auftritt, die gemittelte Spannung gleich der Dauerfestigkeit des Grundmaterials ist.

$$\Delta\sigma_{av|(r=0-2a_0)} = \frac{1}{2a_0}\int_0^{2a_0} \Delta\sigma\sqrt{\frac{a}{2r}}\,dr = \Delta\sigma_0 \qquad \text{Glg. (2.14)}$$

Die dargestellte Methode wird in der Literatur üblicherweise als Linienmethode bezeichnet. Neben der Bewertung des Spannungsverlaufs entlang einer Linie in Rissausbreitungsrichtung, lässt sich auch ein flächenbasiertes und ein volumenbasiertes Konzept ableiten.

Anwendung auf kurze Risse

Im nächsten Schritt werden neben langen auch kurze Risse berücksichtigt. Dazu muss anstatt des vereinfachten Spannungsfeldes aus Glg. (2.13) die allgemeinere Darstellung aus Glg. (2.12) verwendet werden.

$$\Delta\sigma_{av|(r=0-2a_0)} = \frac{1}{2a_0}\int_0^{2a_0} \Delta\sigma\frac{1}{\sqrt{1-\left(\frac{a}{a+r}\right)^2}}\,dr \qquad \text{Glg. (2.15)}$$

$$= \Delta\sigma\sqrt{\frac{a+a_0}{a_0}} = \Delta\sigma_0$$

Es lässt sich also nach Taylor zeigen, dass die Anwendung des Linienkonzepts aquivalent zum empirisch ermittelten Konzept nach El Haddad ist. Der Langrissschwellwert und die Schwingfestigkeit des defektfreien Materials hängen stark von den Gefügeeigenschaften wie der Korngröße, der Wärmebehandlung und nicht zuletzt von der Legierungswahl ab. Sind diese beiden Materialparameter bekannt, dominieren bei der weiteren Betrachtung Defekte und die Oberflächenrauheit. Daher ist es von großer Bedeutung diese beiden Effekte in einem gesamtheitlichen Konzept zu vereinen.

2.3. Grundlagen zur Rauheitsmessung

Prinzipiell werden von einem Messgerät alle Gestaltabweichungen von der Idealform erfasst. Gemäß DIN 4760 werden diese Gestaltabweichungen in sechs Ordnungsklassen eingeteilt. Nur die Ordnungen drei bis fünf tragen zur Oberflächenrauheit bei, die niedrigeren Ordnungen tragen zu den Formabweichungen der Oberfläche bei.

Gestaltabweichung (als Pofilschnitt überhöht dargestellt)		Beispiele für die Art der Abweichung	Beispiele für die Entsehungsursache
1. Ordnung: Formabweichungen 		Unebenheit Unrundheit	Fehler in den Führungen der Werkzeugmaschine, Durchbiegung der Maschine oder des Werkstückes, falsche Einspannung des Werkstückes, Härteverzug, Verschleiß
2. Ordnung: Welligkeit 		Wellen	Außermittige Einspannung oder Formfehler eines Fräsers, Schwingungen der Werkzeugmaschine oder des Werkzeuges
3. Ordnung: 	Rauheit	Rillen	Form der Werkzeugschneide, Vorschub oder Zustellung des Werkzeuges
4. Ordnung: 		Riefen Schuppen Kuppen	Vorgang der Spanbildung (Reißspan, Scherspan, Aufbauschneide), Werkstoffverformung beim Sandstrahlen, Knospenbildung bei galvanischer Behandlung
5. Ordnung: nicht mehr in einfacher Weise bildlich darstellbar		Gefügestruktur	Kristallisationsvorgänge, Veränderung der Oberfläche durch chemische Einwirkung (z.B. Beizen), Korrosionsvorgänge
6. Ordnung: nicht mehr in einfacher Weise bildlich darstellbar		Gitteraufbau des Werkstoffes	Physikalische und chemische Vorgänge im Aufbau der Materie, Spannungen und Gleichungen im Kristallgitter
 Überlagerung der Gestaltabweichungen 1. bis 4. Ordnung			

Abb. 2.4: Ordnungssystem für Gestaltabweichungen gemäß DIN 4760

Um ein korrektes dreidimensionales Primärprofil aus abgetasteten Messdaten zu erhalten sind im ersten Schritt Artefakte zu erkennen und aus den Rohdaten zu entfernen. Anschließend werden die fehlenden Datenpunkte interpoliert. Dabei kommt meist ein bikubisches oder billineares Interpolationsverfahren zum Einsatz.

Das derart ermittelte Primärprofil setzt sich aus einem Formanteil, der Welligkeit und dem Rauheitsanteil zusammen. Diese Anteile werden durch Profilfilter getrennt. Dafür ist laut DIN EN ISO 11562 ein phasenkorrekter Gauss-Filter zu verwenden. Je nach vorhandener Rauhigkeit sind in der Norm für zweidimensionale Oberflächenkennwerte unterschiedliche Grenzwellenlängen zu verwenden, um die Rauhigkeit von der Welligkeit einerseits und Signalanteilen die höher als die Rauhigkeit sind abzugrenzen.

Daran anschließend können unterschiedliche Rauheitsparameter wie Mittenrauheit, gemittelte Rauhtiefe oder maximale Rautiefe ermittelt werden, welche gemäß DIN 4768 definiert sind.

Klassische Betrachtung der Oberflächenrauheit

Klassische Konzepte um den Einfluss der Rauheit auf die Schwingfestigkeit zu beschreiben basieren zumeist auf der gemittelten Rautiefen R_z, der maximal vorhandenen Rautiefe R_t oder einer fertigungsbezogenen Beschreibung des Oberflächenzustandes. Eine weiterführende Beschreibung der Oberflächentopologie findet dabei in der Regel nicht statt.

Das Materialverhalten wird nach FKM [FOR12] durch drei Parameter bestimmt. Neben der Zugfestigkeit R_m des verwendeten Werkstoffes und der minimalen Zugfestigkeit Zugfestigkeit der Werkstoffgruppe $R_{m,N,min}$ wird das Ermüdungsfestigkeitsverhalten auch durch einen rein empirischen Parameter $a_{R,\sigma}$ welcher durch die Materialgruppe definiert ist beschrieben.

$$K_{R,\sigma} = 1 - a_{R,\sigma} \cdot lg\left(\frac{Rz}{\mu m}\right) \cdot lg\left(\frac{2 \cdot R_m}{R_{m,N,min}}\right) \qquad \text{Glg. (2.16)}$$

Abb. 2.5: Abminderungsfaktor der Dauerfestigkeit in Abhängigkeit der gemittelten Rauheit Rz

3. Experimentelle Untersuchungen

In diesem Kapitel werden die experimentellen Tätigkeiten zur Ermittlung der Schwingfestigkeit gussrauer Oberflächen aus Zylinderkurbelgehäusen dargestellt. Ziel der Wöhlerversuche ist es, die Notwendigkeit einer numerischen Bewertung des Oberflächeneinflusses auf die Ermüdungsfestigkeit bei Gussoberflächen darzustellen bzw. eine experimentelle Datenbasis für eine Evaluierung zu schaffen.

3.1. Entnahmestellen

Die Proben wurden aus einem im Kokillengussverfahren gefertigten Zylinderkurbelgehäuse aus dem Werkstoff AlSi7MgCu0,5 gefertigt. Um die Gussoberfläche möglichst ohne weitere Einflüsse wie zum Beispiel einem variierenden Werkstoffgradienten in Dickenrichtung durch unterschiedliche Abkühlraten bzw. damit verbunden variierende Dendritenarmabstände bzw. Porengehalte zu minimieren, wurden Entnahmestellen mit konstanter Wanddicke gewählt. Anzumerken ist jedoch, dass in den Bereichen konstanter Wanddicke aufgrund der Volumenschwindung beim Erstarren der Aluminiumgusslegierung stets auch Schwindungsporen im Inneren bzw. im Randschichtbereich auftreten können. In den betrachteten Zylinderkurbelgehäusen kommen sowohl Bereiche mit einer Sandgussoberfläche am Kern und einer Gussoberfläche an der Kokille vor.

Die aus dem Kokillenbereich gefertigten Proben wurden aus dem Kettenkastenbereich entnommen Abb. 3.1, Proben mit Sandgussoberfläche Abb. 3.2 wurden aus dem Wassermantelbereich entnommen.

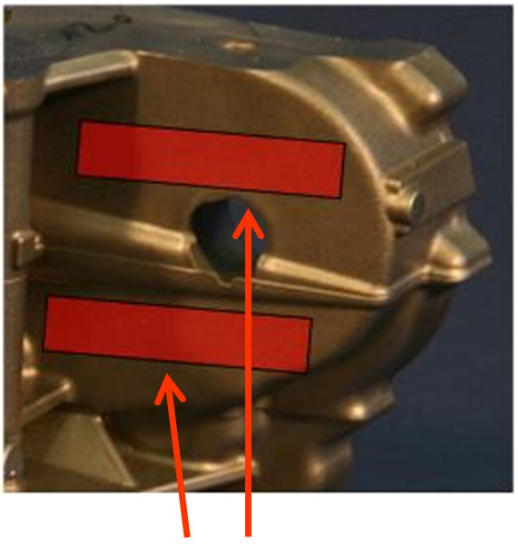


Abb. 3.1: Entnahmestelle aus dem Kokillenbereich

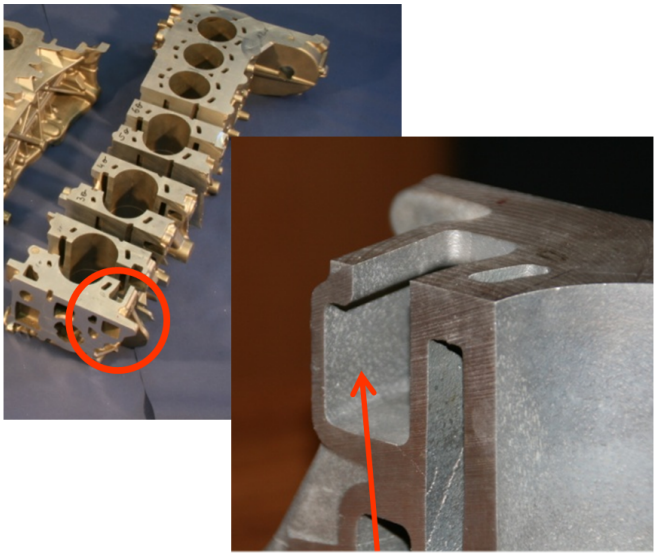


Abb. 3.2: Entnahmestelle aus dem Sandkernbereich

3.2. Probengeometrie und Fertigung

Die Zielsetzung liegt darin, die Oberfläche unabhängig vom Einspanneinfluss hinsichtlich Schwingfestigkeit prüfen zu können. Dabei muss das gemäß den Entnahmestellen verfügbare Probenvolumen auch ausreichend Platz für eine sichere Probenspannung bieten. Angesichts der geometrisch bedingten Einschränkungen des Entnahmebereichs wurde die in Abb. 3.3 dargestellte Geometrie gewählt.

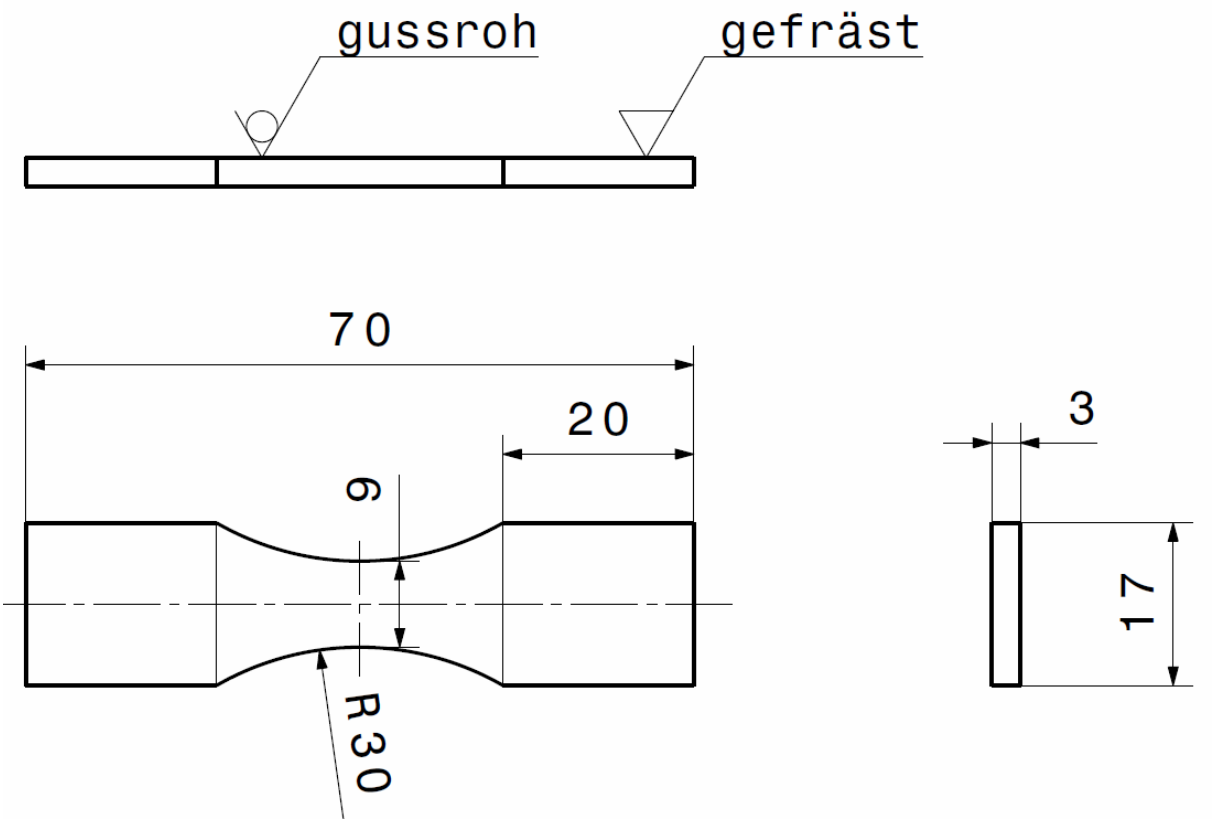


Abb. 3.3: Verwendete Probengeometrie

Die Probenoberfläche ist auf der Oberseite gussroh und an den Einspannstellen sowie auf den Seitenflächen maschinell bearbeitet. Um Anrisse an den Seiten beziehungsweise an den Kanten der Probe zu vermeiden wurden sämtliche Proben zusätzlich per Hand gründlich entgratet und die Seitenränder mit Schleifpapier poliert.
Anzumerken ist, dass diese vergleichsweise scharfe Kerbe in der Prüfzone die Biegebeanspruchung in der Randschicht im Eckenbereich um sieben Prozent, im Vergleich zum ebenen Dehnungszustand in der Probenmitte, erhöht.

3.3. Versuchsdurchführung

Die aus dem Zylinderkurbelgehäuse gefertigten Flachproben wurden auf einem Hydropulsprüfstand mit einer Maximalkraft von 100kN der Firma Winter Hydraulik GmbH geprüft. Die auftretenden Kräfte wurden mit einer 20kN Kraftmessdose erfasst. Die Prüffrequenz betrug bei beiden Versuchsserien 15 Hz.

Um die technische Rissinitierung an die unbearbeitete Oberfläche zu verlegen wurde auf schwellende Biegung (R=0) geprüft. Die verwendete Vierpunkt-Biege-Prüfvorrichtung ist in Abb. 3.4 dargestellt.

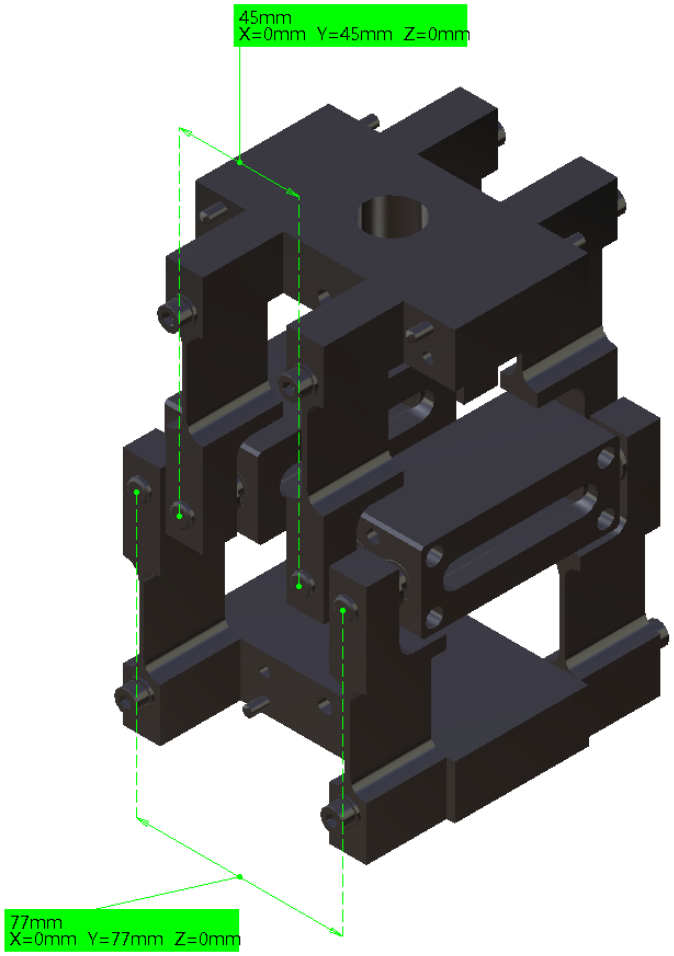


Abb. 3.4: Darstellung der Prüfvorrichtung

Nach dem Vierpunktbiegeprinzip tritt zwischen den inneren Auflagern ein konstantes Biegemoment auf. Werden die dargestellten Abstände der Vorrichtung mit a und b bezeichnet, so erhält man für die Maschinenkonstante als Verhältnis von Biegemoment zu Prüfkraft am Hydropulszylinder einen Wert von $k_m = 8Nmm/N$. Die verwendete Probenbreite weist ein axiales Widerstandsmoment von $13{,}5mm^3$ auf.

Demzufolge würde eine Prüfkraft am Servozylinder von $100N$ zu einer Biegenennspannung von $59{,}3N/mm^2$ führen. Die dauerfest ertragbaren Spannungsamplituden dieser Legierung

liegen somit in einem vergleichsweise geringen Prüfbereich des hydraulischen Aktuators bzw. der verwendeten Kraftmeßdose.

$$k_m = \frac{M_b(F)}{F} = \frac{a-b}{4} \qquad \text{Glg. (3.1)}$$

Abb. 3.5: Maschinenkonstante der Probe auf Biegung

3.4. Ergebnisse

Die mittels Abtastung ermittelte Oberflächenqualität war sowohl bei der Kokillenoberfläche als an der Sandkernoberfläche vergleichbar hoch. Die am Sandkern entnommenen Proben weisen jedoch eine um etwa fündundzwanzig Prozent höhere Schwingfestigkeit im Vergleich zu Proben aus dem Kokillenbereich auf.

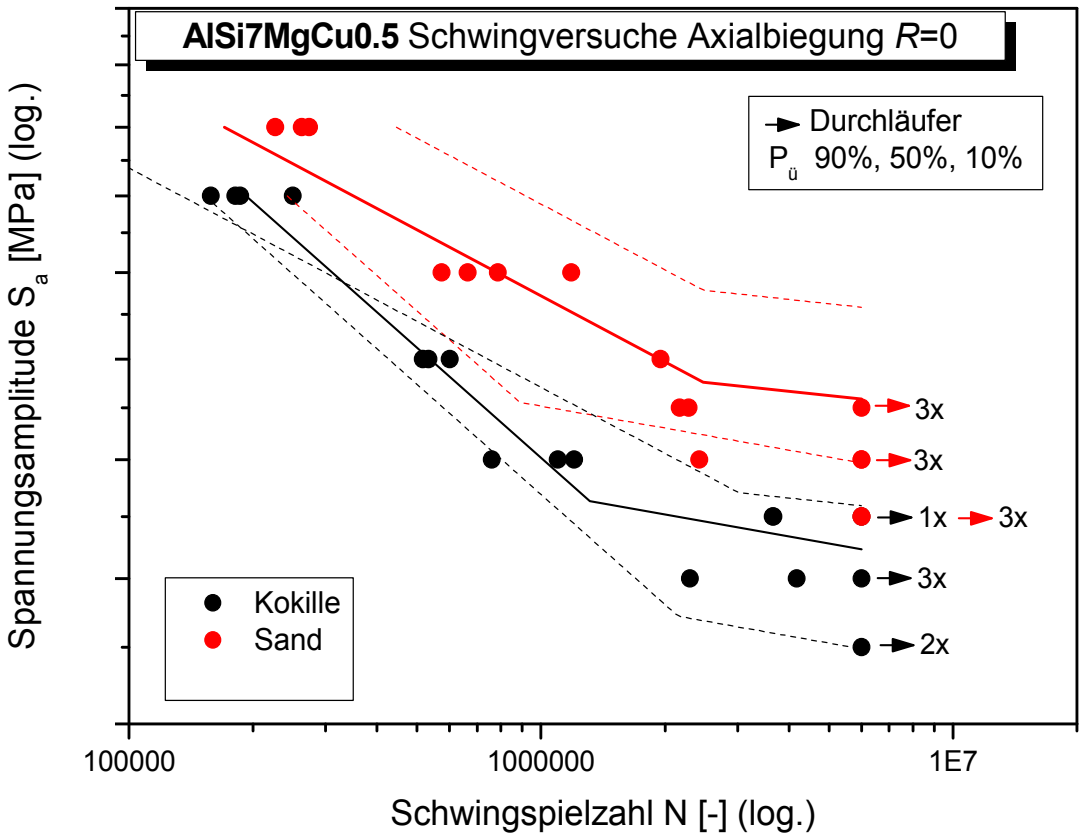


Abb. 3.6: Wöhlerversuchsergebnisse an Sand und Kokillenoberfläche

3.4.1. Untersuchung der Bruchflächen

Um den Anrissort und den Versagensmechanismus feststellen zu können wurden alle Bruchflächen am Rasterelektronenmikroskop untersucht. Bei den aus dem Kokillenbreich entnommenen Proben zeigte sich nur bei einer Probe ein klarer Anriss von der Oberfläche. Alle anderen Proben versagten aufgrund von oberflächennaher Porosität.

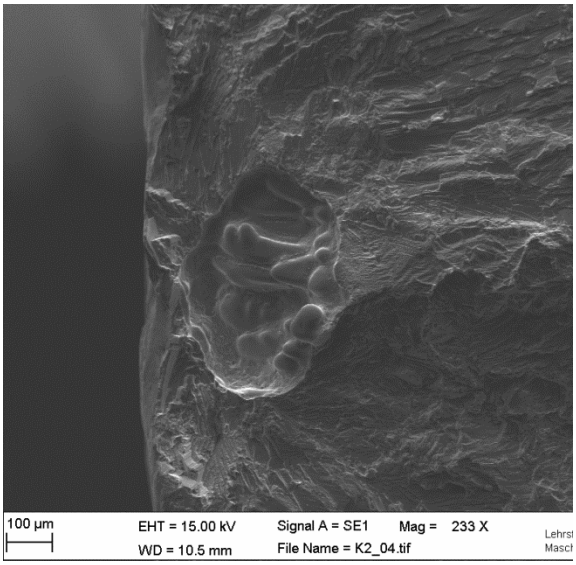


Abb. 3.7: Oberflächennahe Pore

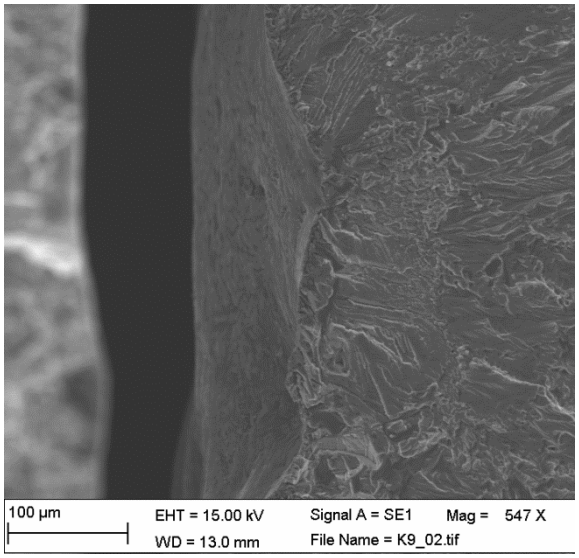


Abb. 3.8: Rissausgang an der Oberfläche

Bei den Proben aus dem Sandkernbereich waren mehr Mischbrüche, die von der Oberfläche und Defekten ausgingen, erkennbar. Insgesamt muss aber trotzdem davon ausgegangen werden, dass auch hier die Mikroporosität die überwiegende Schadensursache darstellt. Daher konnten die beiden Versuchsreihen nicht zur Validierung der Oberflächenkennwerte eingesetzt werden. Jedoch kann die Oberflächentopographie weiterhin zur numerischen Bewertung verwendet werden um auf Unterschiede zwischen den beiden Verfahren zu schließen.

Aus der Schadensanalyse ist zu schließen, dass in der Randschicht nicht nur die Oberflächenqualität bzw. Kenngrößen wie die maximale Rauheit entscheidend sind, sondern vielmehr das Zusammenwirken von Mikroporosität und Oberflächentopographie beurteilt werden muss.

4. Numerische Modellierung

Das Ziel dieses Kapitels stellt die Entwicklung von umfassenden Simulationswerkzeugen zur spannungsbasierten Bemessung technischer Oberflächen dar. Besonderes Augenmerk wird auf die numerisch effiziente Umsetzung gelegt, um sowohl zeit- als auch kosteneffizient unterschiedliche Oberflächentopologien zu charakterisieren. Ein weiterer Schwerpunkt liegt in der Minimierung der personenbezogenen Arbeitszeit die aufzuwenden ist um eine Oberfläche zu charakterisieren.

4.1. Vermessen der Oberfläche

Um das Simulationsmodell aufzubauen wurde zunächst die Oberfläche einer Sandgussprobe zerstörungsfrei mittels Laser-Konfokalmikroskopie eingescannt. Dazu wurde ein Laser-Konfokalmikroskop vom Type „mysurf custom“ der Firma „nano Focus“ verwendet. Der ca. 7x7 mm große Bereich hat eine laterale Auflösung bei der Abtastung von etwa 1,6 µm.

4.2. Filterung der Gussoberfläche

Durch Messfehler und dem begrenzten Flankenwinkel kommt es in den Rohdaten zu ungültigen Messpunkten. Diese wurden zuerst entfernt und durch Interpolation Ersatzmesspunkte ermittelt. Die Interpolation erfolgte dabei durch ein bikubisches Verfahren. In einem weiteren Schritt wurde die gemessene Oberfläche um den Ursprung zentriert und parallel zur xy - Ebene ausgerichtet. Anschließend wurden Formabweichungen aus den Messdaten entfernt. Dazu wird wahlweise eine biquadratische oder bikubische Funktion an die Oberfläche angepasst und die x- als auch y- Koordinaten der Messwerte auf diese analytische Fläche projeziert. Aus der Abwicklung dieser analytischen zweidimensionalen Fläche lässt sich nun eine formkorrigierte Oberfläche mit geringem Aufwand berechnen.

Im in Abb. 4.1 dargestellten Primärprofil wird ersichtlich, dass zusätzlich zu der langwelligen makroskopischen Rauigkeit noch ein sehr kurzwelliger Anteil überlagert ist. Zum Teil entspricht dieser kurzwellige Anteil definitionsgemäß nicht mehr der Rauheit. Zusätzlich würde eine ausreichend feine Vernetzung dieser kurzwelligen Strukturen zu einer nicht mehr praktikablen Modellgröße führen und erscheint auch im Hinblick auf den zu erwartenden Einfluss auf die Festigkeit als nicht sinnvoll. Des Weiteren erschwert die kurzwellige Rauigkeit den nachfolgenden Vernetzungsvorgang. Daher wurde der kurzwellige Anteil mittels eines robusten Gauß-Filters mit einer Grenzwellenlänge von 80 µm entfernt (Abb. 4.2, Abb. 4.3). Die erfasste und gefilterte Oberfläche weist keine sichtbare Vorzugsorientierung oder Riefen im Sinne mechanischer Bearbeitung auf.

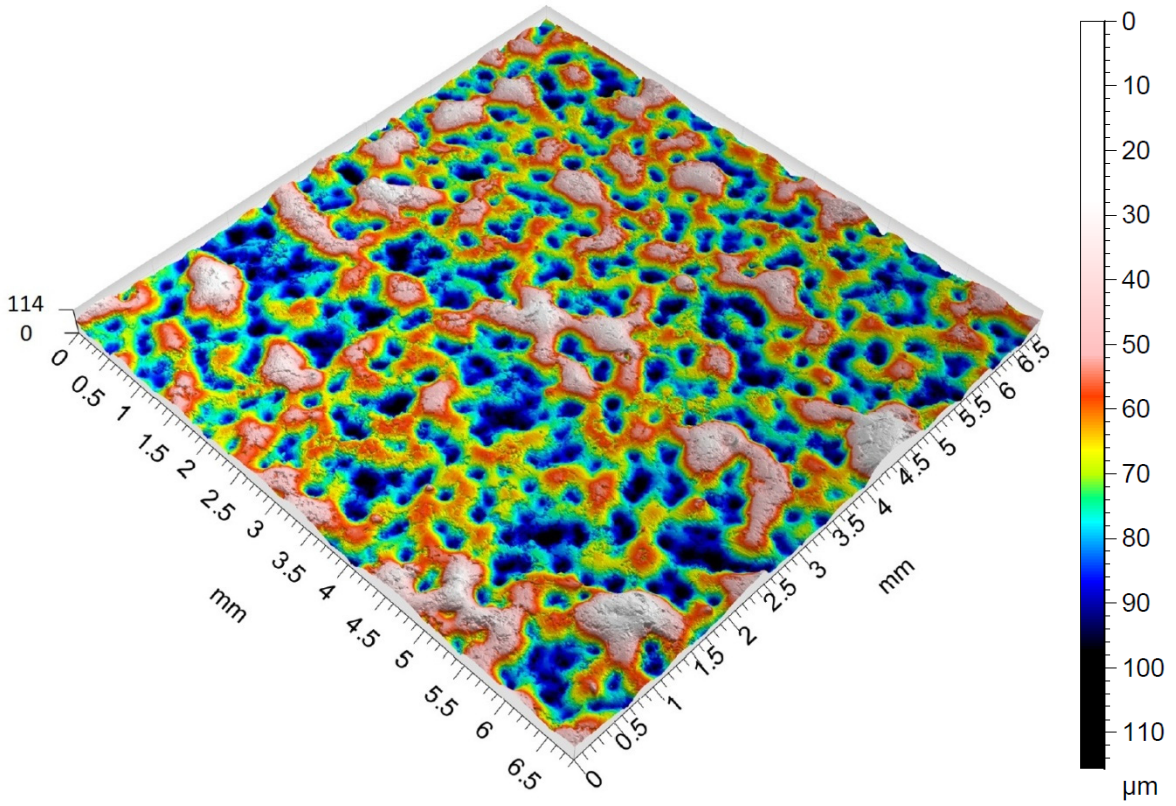


Abb. 4.1: Dreidimensionale Darstellung der Oberflächenmorphologie vor Anwendung des Filters

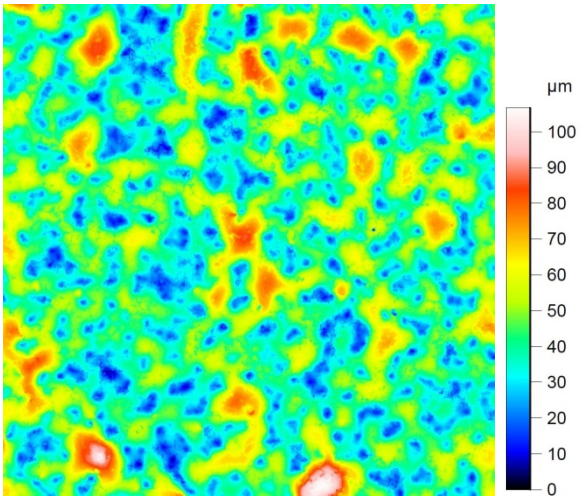


Abb. 4.2: 2D-Oberflächenmorphologie vor Anwendung des Gauß-Filters

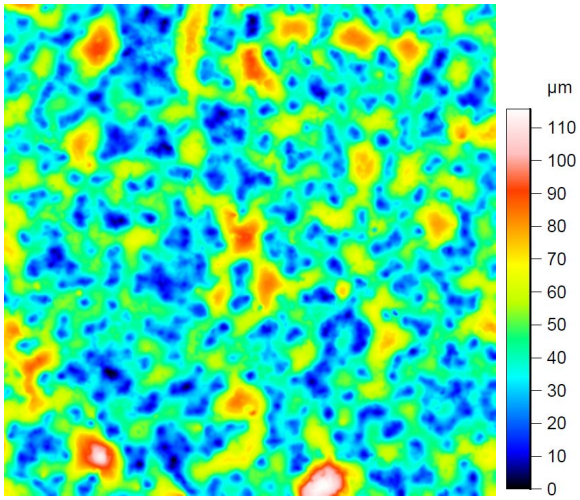


Abb. 4.3: 2D-Oberflächenmorphologie nach Anwendung des Gauß-Filters

4.3. Erstellung des Oberflächennetzes

4.3.1. Export der Oberflächenmorphologie

Nach der Filterung der Oberfläche liegt diese in Form von in regelmäßigen Abständen angeordneten Datenpunkten vor. Der erste Schritt bestand daher darin, aus den Datenpunkten eine tesselierte Oberfläche zu erzeugen, die danach in bestehende FE-Vernetzungsprogramme importiert werden kann. Dies wurde mithilfe eines im Rahmen der Arbeit erstellten benutzerdefinierte Python-Scripts realisiert, welches ein strukturiertes Netz erstellt. Als Ausgabeformate stehen verschiedene Standardformate zur Verfügung. Aufgrund der weitreichenden Unterstützung seitens der nachfolgenden Vernetzungsprogramme erfolgt die Ausgabe jedoch standardmäßig im STL Format (Surface Tesselation Language) in seiner binären Form. Die weitere Vernetzung der Oberfläche erfolgte mit dem Tool Gmsh [GR09].

4.3.2. Krümmungsabhängige Vernetzung

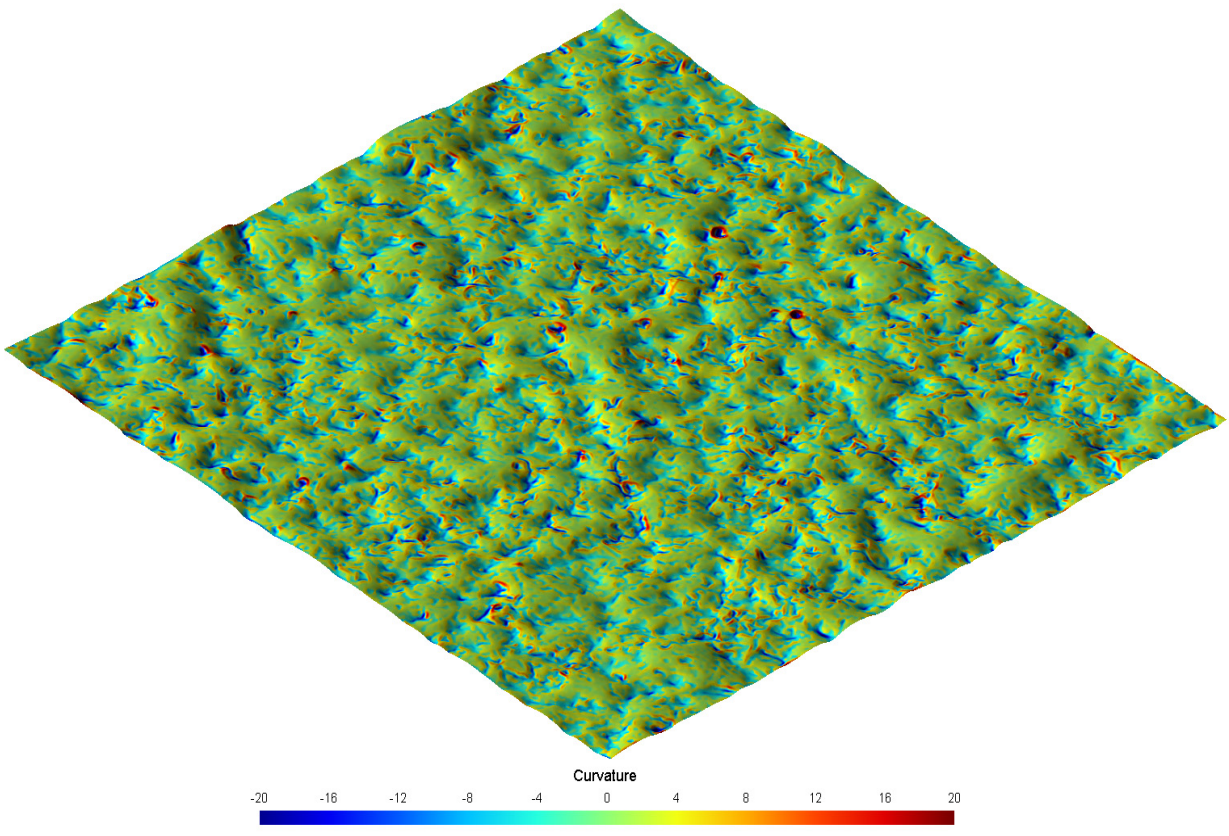


Abb. 4.4: Darstellung der maximalen Hauptnormalkrümmung auf der betrachteten Gussoberfläche

Eine hohe Netzfeinheit sowie Netzqualität ist erforderlich, um möglichst exakte Spannungswerte erhalten zu können. Eine stark erhöhte Netzfeinheit bzw. Elementanzahl steigert allerdings massiv die Rechenzeit und ist selbst mit der zum heutigen Zeitpunkt zur

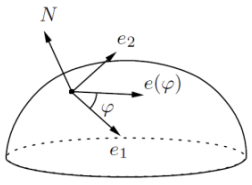
Verfügung stehenden Rechenleistung nicht mehr sinnvoll darstellbar. Um möglichst exakte Spannungsergebnisse zu erhalten, sollte das Oberflächennetz möglichst Dreieckelemente mit einem Eckenwinkel von 60 Grad aufweisen, anstatt der gleichmäßigen 45 Grad, welche sich bei der Tesselierung der Datenpunkte ergeben. Spannungsüberhöhungen, die einen Anriss an der entsprechenden Stelle begünstigen, treten wie in Abb. 4.6 und Abb. 4.7 dargestellt tendenziell an Stellen mit hoher konkaver Krümmung in Belastungsrichtung auf. Neben der Kerbschärfe, bzw. Krümmung der Oberfläche sind für die Höhe der Spannung an der betrachteten Stelle ebenfalls noch die Kerbtiefe und die Wechselwirkungen mit benachbarten Kerben relevant.

Diese Zielnetzgröße kann in Gmsh direkt vorgegeben werden. Diese lokalen Parameter zur Bestimmung der Netzgröße können entweder aus geometrischen Daten wie der Krümmung oder auch auf Basis vorheriger Spannungsergebnisse berechnet werden. Im Rahmen dieser Arbeit wurde ein Interface entwickelt, um direkt in Python ein entsprechendes Verfahren zur Zielnetzgrößenbestimmung abbilden zu können und dann in einfacher Weise im Gmsh-Vernetzungsprogramm abarbeiten zu können. Die Krümmungsberechnung wird mit einem Algorithmus nach Rusinkiewicz [SZY04] durchgeführt. Dabei werden die Hauptnormalkrümmungen mittels gewichteten Oberflächennormalen an den Netzknoten berechnet.

In Abb. 4.6 wird die Oberflächenkrümmung in eine vorgegebene Belastungsrichtung dargestellt. Dazu wird der die Belastungsrichtung beschreibende Vektor zuerst auf die jeweilige lokale Tangentialfläche projiziert. Danach wird der Winkel zwischen dem projizierten Lastvektor und der 1. Hauptkrümmungsrichtung berechnet. Abschließend wird gemäß der in Abb. 4.5 dargestellten Gleichungen die Krümmung in Belastungsrichtung für jeden Knoten ermittelt.

$$e(\varphi) = \cos\varphi\, e_1 + \sin\varphi\, e_2$$

$$\kappa(\varphi) = \lambda\cos^2\varphi + \mu\sin^2\varphi.$$



e1.... 1.Hauptkrümmungsrichtung
e2.... 2. Hauptkrümmungsrichtung
λ..... 1. Hauptkrümmung
μ..... 2.Hauptkrümmung
κ..... Krümmung in vorgegebene Richtung

Abb. 4.5: Berechnung der Krümmung in eine vorgegebene Richtung [BOB06]

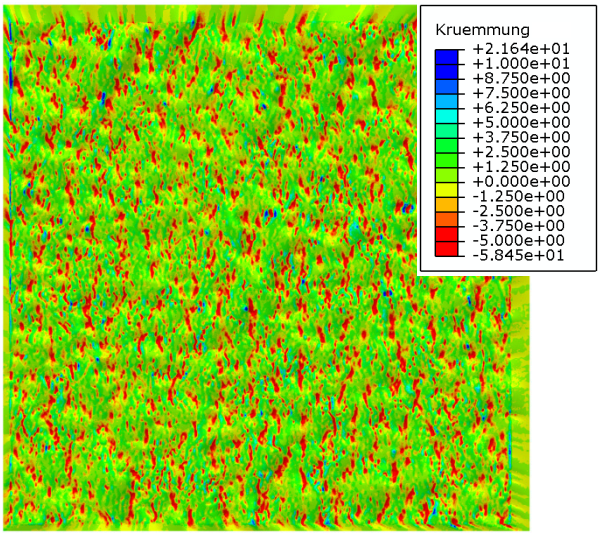


Abb. 4.6: Krümmung in Belastungsrichtung [1/mm]

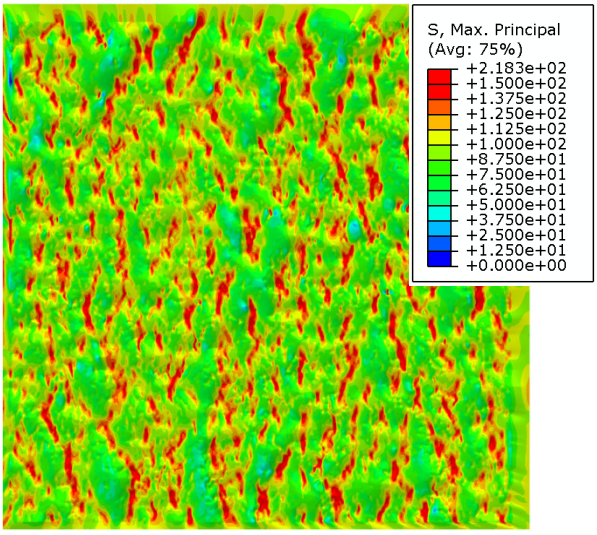


Abb. 4.7: Hauptnormalspannungen [MPa]

Wie aus Abb. 4.6 ersichtlich wird, streut die Krümmung lokal relativ stark und weist zusätzlich einen hohen Gradienten auf. Würde man direkt aus der Krümmung die lokale Netzgröße vorgeben, so würde die lokale Netzgröße ebenfalls stark variieren. Für eine gute Ergebnisqualität ist es allerdings wünschenswert die Änderung der Netzfeinheit in Abhängigkeit des Abstandes zum jeweiligen Bezugspunkt zu begrenzen.

Prinzipiell kann die Berechnung der Zielnetzgröße in folgende Schritte unterteilt werden:

- Ermittlung des Eingabeparameters (im gezeigten Fall der maximalen Hauptnormalkrümmung der Oberfläche)
- Berechnen der Zielnetzgröße aus dem Eingabeparameter
- Glätten der Zielnetzgrößen
- Grenzwerte für die Netzgröße festlegen (maximale und minimale Netzgröße)
- Festlegen der Wachstumsrate der Netzgröße
- Übergabe der Daten an das Tool Gmsh und vernetzen der Oberfläche

Berechnen der Zielnetzgröße

Hierzu muss zuerst zwischen negativer (konkaver) und positiver (konvexer) Krümmung unterschieden werden. Für konvexe und konkave Krümmungen können unterschiedliche Übertragungsfunktionen genutzt werden. Bei konkaver Krümmung kommt eine quadratische Übertragungsfunktion zum Einsatz.

Glätten der Zielnetzgrößen

Um kleinflächige Ausreißer zu reduzieren kommt an dieser Stelle ein Mittelungsansatz zu Einsatz. Jeder Knotenwert im Ursprungsnetz wird dazu mit allen Nachbarknotenwerten gemittelt. Bei Bedarf kann diese Mittelungsfunktion auch angepasst werden. Mögliche Anpassungen sehen vor dabei nur in einem gewissen Maße ähnliche Knotenwerte zur Mittelung heranzuziehen, oder eine abstandsabhängige Gewichtung der Nachbarknoten vorzunehmen.

Festlegen der Wachstumsrate der Netzgröße

Dazu wird im ersten Schritt der Abstand jedes Knotens zu all seinen direkten Nachbarknoten ermittelt. Die maximale Wachstumsrate wird durch eine linear vom Abstand abhängige Funktion beschrieben. Die erforderliche Konstante kann vorgegeben werden und entspricht der in vielen Vernetzungsprogrammen üblichen Growth-Rate Vorgabe.

Im zweiten Schritt wird die zugewiesene Netzgröße jedes Knotens mit seinen Nachbarknoten verglichen. Ist die Abweichung dabei größer als vorgegeben, das heißt würde die Netzgröße zu stark ansteigen, wird dem jeweiligen Knoten jener Wert zugewiesen welcher der Wachstumsbedingung gerade noch genügt. Dieser Vorgang wird so lange wiederholt bis jeder Knotenwert den Vorgaben entspricht.
Dieses Konzept kann auch auf dreidimensionale Netze erweitert werden, in dem statt mit Nachbarknoten, welche aus der Elementkonnektivität ermittelt werden, mit Knoten die sich in einer definierten räumlichen Nähe befinden verglichen wird. Das so erweiterte Konzept kann zum Beispiel bei der Vernetzung von Imperfektionen in Gussmaterialen nutzbringend angewandt werden.

Durch den vorgestellten Ablauf lässt sich eine Oberfläche nicht nur krümmungsabhängig vernetzen, sondern es kann auch eine hervorragende Elementqualität erreicht werden. Die erforderlichen Methoden wurden in Python 2.7 umgesetzt. Um den Berechnungsvorgang zu beschleunigen kommt der Just in Time Compiler Numba [LAM15] zum Einsatz, welcher Python Code in Maschinencode umsetzen kann. Um einen effizienten Maschinencode zu erhalten sind nur wenige Typdeklarationen beziehungsweise Angaben über die Arrays, die an die Funktion übergeben werden, erforderlich. Bei Ausführung des derart compilierten Codes kann auch der Global Interpreter Lock von Python umgangen werden und der Berechnungsvorgang auf alle zur Verfügung stehende Kerne aufgeteilt werden. Selbst für eine Million Knoten beträgt die Rechenzeit zur Ermittlung der lokalen Netzgrößen unter einer Minute.

Für eine hinreichend genaue Beschreibung der Oberfläche liegen die Netzgrößen üblicherweise im Bereich von $8\mu m$ bis $50\mu m$. Als Elementtyp kamen quadratische Dreieckselemente zum Einsatz. Bei einer betrachteten Fläche von $7x7mm^2$ beträgt somit allein die Oberflächenelementanzahl durchwegs mehrere Millionen Elemente.

4.4. Aufbau des 3D- Modells

4.4.1. Allgemeine Modellbeschreibung

Der Aufbau des dreidimensionalen Modells erfolgte modular, um eine effiziente Anpassbarkeit auf zukünftig zu untersuchende Oberflächen gewährleisten zu können. Das Gesamtmodell (Abb. 4.8) setzt sich daher aus drei Teilen zusammen. In der Mitte befindet sich der Versuchsträger auf dessen Oberseite die erfasste Oberflächenmorphologie eingebaut wurde. Der Kontakt zwischen Versuchsträger und Versuchsträgeraufnahme wurde als Tied - Contact ausgeführt. In den ersten Modellversionen wurde die Versuchsträgeraufnahme je nach Belastungsrichtung gedreht, während in den weiter fortgeschrittenen Versionen die Spannungsergebnisse wie nachfolgend in Kapitel 4.4.2 beschrieben aus Standardlastfällen ermittelt wurden, um die Rechenzeit zu reduzieren. Dieses Verfahren setzt allerdings linear elastisches Materialverhalten voraus. Daher wurde die Drehbarkeit des Versuchsträgers beibehalten, um keine Einschränkungen für zukünftige Untersuchungen zu schaffen. Der Kontakt zwischen Versuchsträgeraufnahme und dem Einspannkörper, auf den die entsprechenden Lasten aufgebracht wurden, erfolgte wiederum mittels Tied - Contact. Das gesamte Modell wurde im Pre-Processing basierend auf den Oberflächendaten aufgebaut und umfasst ca. drei Millionen quadratische Tetraeder Elemente (C3D10), wobei der überwiegende Anteil dem Probenträger zugeordnet ist. Es wurden auch Ansätze getestet um die gesamte Netzerstellung des Versuchsträgers vollautomatisiert in Gmsh darstellen zu können. Jedoch erfordert dies vertiefte Kenntnisse der Benutzer hinsichtlich der Programmiersprache in Gmsh, weshalb kommerziell verfügbare Prä-Prozessoren bevorzugt wurden. Als FE- Solver des linear-elastischen Modells wurde Abaqus 6.13 verwendet.

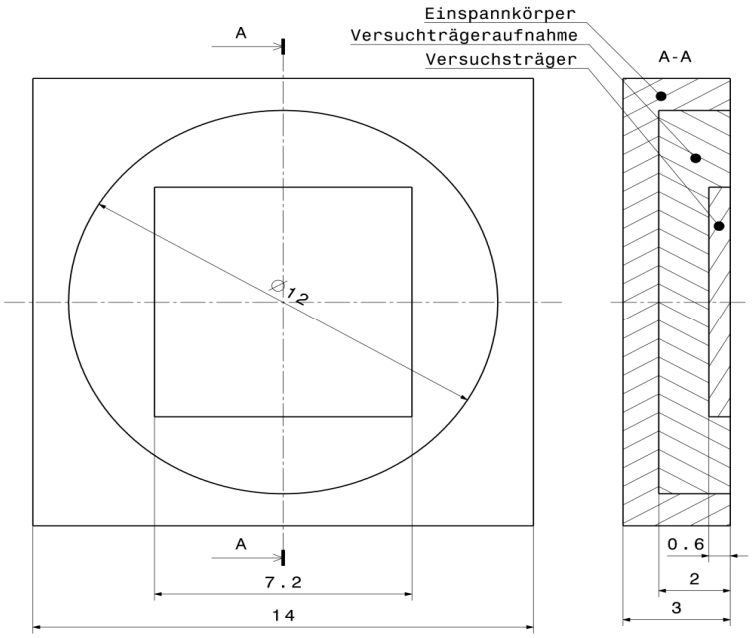


Abb. 4.8: Modellaufbau

4.4.2. Lastaufbringung

In der vorliegenden Arbeit wird der Oberflächeneinfluss ohne globalen Spannungsgradienten betrachtet. Daher wurde die Gussoberfläche ausschließlich einer Zugbelastung unterworfen. Die Höhe der zu beobachtenden Kerbwirkung hängt dabei vor allem von der Ausrichtung der Mikrokerben zur jeweiligen Belastungsrichtung, von der Kerbschärfe sowie der Kerbtiefe ab.

Da die Ausrichtung der Mikrokerben eine statistische Verteilung aufweist, müssen alle Belastungsrichtungen (0-180 Grad) berücksichtigt werden. Dabei ergibt sich einerseits ein Zielkonflikt zwischen möglichst geringer Inkrementierung der Belastungsrichtung und akzeptablen Rechenzeiten andererseits. Diesen prinzipiell konträren Anforderungen lässt sich allerdings genügen, indem jede mögliche Belastungsrichtung als Linearüberlagerung dreier in Abb. 4.9 dargestellten Standardlastfällen (Zug 0°, Zug 90°, Schub) betrachtet wird. Es werden somit alle drei Spannungstensoren in das jeweilige gedrehte Koordinatensystem transformiert und dann nach Glg. (4.1) überlagert.

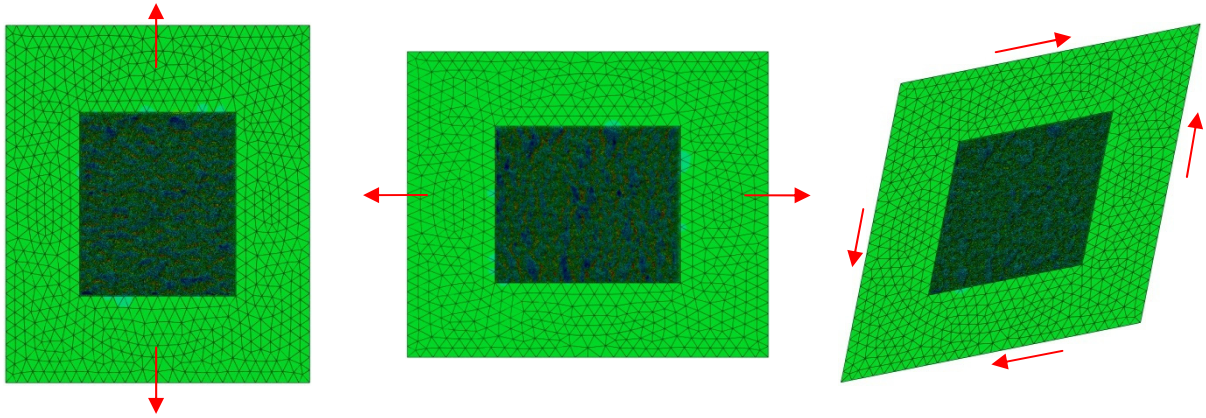


Abb. 4.9: Standardlastfälle (Zug 0°, Zug 90°, Schub)

$$\sigma(\alpha) = \sigma_{0\,Grad}^{trans} \cdot cos^2(\alpha) + \sigma_{90\,Grad}^{trans} \cdot sin^2(\alpha) + \sigma_{Schub}^{trans} \cdot \cos(\alpha) \cdot \sin(\alpha) \qquad \text{Glg. (4.1)}$$

4.4.3. Standardlastfälle

Bei den Zuglastfällen (Abb. 4.9) wurden jeweils zwei gegenüberliegende Seiten mittels Element-Based Distributed-Couplings an jene Referenzknoten gekoppelt, an denen die Einspannbedingung definiert bzw. die Last aufgebracht wurde. Diese Kopplungen sind etwas nachgiebiger als die standardmäßig verwendeten Kinematic-Couplings und führen somit zu einer Krafteinleitung ohne die Verformung der Struktur zu beeinflussen.

Beim reinen Schublastfall Abb. 4.10 wurden die vier dargestellten Referenzpunkte ebenfalls mit Distributed Coupling Elementen mit den Seitenflächen gekoppelt. Um eine Verzerrung der Seitenflächen zu verhindern, ist es hier allerdings notwendig den Rotationsfreiheitsgrad zweier gegenüberliegender Referenzknoten zueinander zu koppeln.

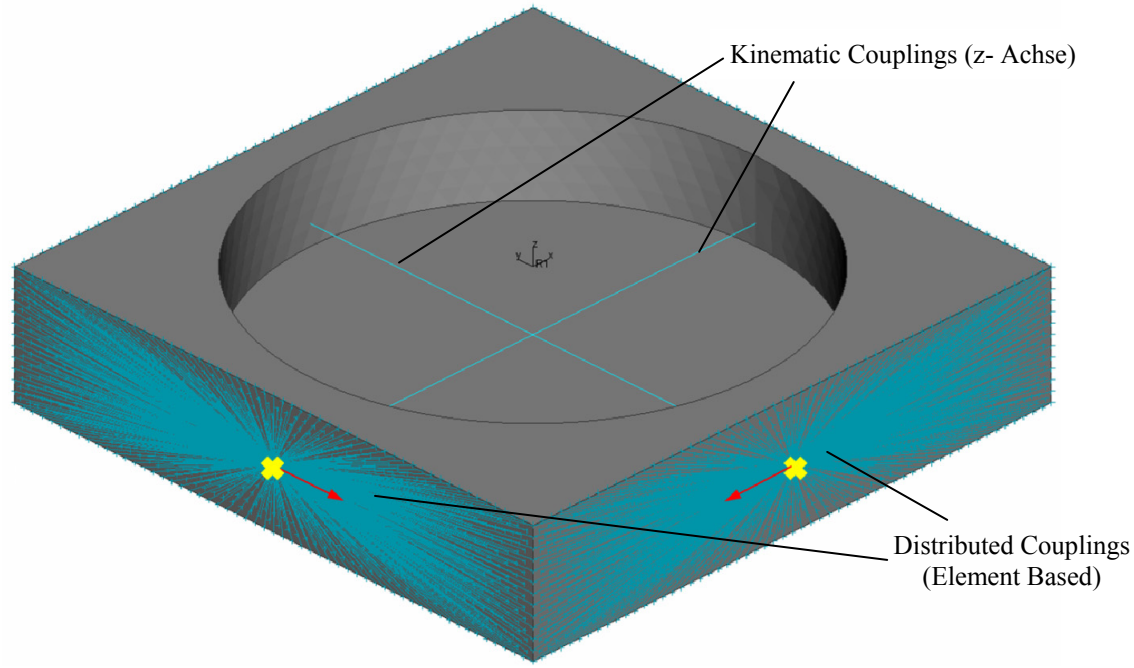


Abb. 4.10: Einspannkörper mit Randbedingungen für den Schublastfall

5. Methodik zur Bewertung des Oberflächenabminderungsfaktors

Im Allgemeinen muss bei der Ermittlung des Oberflächeneinflusses zwischen der lokalen, auf jeden Oberflächenknoten bezogenen Bewertung und der daraus folgenden gesamtheitlichen Betrachtung des Oberflächeneinflusses unterschieden werden.

Die lokale Bewertung bezieht sich dabei nur auf die gemessene Oberfläche unter einer spezifischen Belastungssituation und stellt daher nur eine statistische Verteilung der Abminderungsfaktoren auf einer am Bauteil vorhandenen, ähnlichen Oberfläche dar. Die Übertragbarkeit der Ergebnisse ist daher nur unter Kenntnis der gewünschten Ausfallswahrscheinlichkeit als auch der Größe des höchstbelasteten Bereichs am Bauteil anwendbar.

Im Folgenden wird zuerst die Ermittlung des Abminderungsfaktors für einen einzelnen Oberflächenknoten beschrieben und danach die Erweiterung zur Bewertung einer gesamten Oberfläche dargestellt. Nach jedem Abschnitt wird auf die entsprechende Sektion im Anhang verwiesen, die den Code enthält.

5.1. Ermittlung des Spannungsverlaufs normal zur Oberfläche

Um das Konzept des kritischen Abstandes anwenden zu können, ist es zunächst erforderlich, den Spannungsverlauf in die bevorzugte Rissausbreitungsrichtung zu ermitteln, welche für Mode 1 Rissfortschritt definitionsgemäß normal zur Hauptnormalspannungsrichtung liegt. Im Modell liegt diese global gesehen in der jeweiligen Belastungsrichtung. Lokal beschränkte Abweichungen, die aus den vorhandenen Mikrokerben resultieren, werden nicht berücksichtigt.

Um Auswirkungen auf die Elementqualität zu vermeiden, können die sich unter der gekrümmten Oberfläche des FE-Modells befindenden Knoten nicht in regelmäßigen Abständen normal zur Oberfläche angeordnet werden. Dadurch wird die Festlegung von virtuellen Auswertepfaden erforderlich, auf denen in weiterer Folge Interpolationspunkte definiert werden. Die zugehörige Hauptfunktion ist im Anhang in Kapitel 9.1.1 angeführt.

In Abb. 5.2 wird der gesamte Ablauf zur Berechnung der Spannungsverläufe mittels Flussdiagramm mit den entsprechenden Verweisen auf den im Anhang befindlichen Code zusammenfassend dargestellt.

5.1.1. Festlegen der Auswertepfade

Als Startpunkte der Spannungspfade dienen alle auszuwertenden Oberflächenknoten. Als nicht auswertbar galten dabei nur Knoten die einen Abstand von *0,1 mm* vom Rand des Scanbereichs aufweisen, da an den Rändern durch die Einspannung bzw. den Netzübergang Spannungsüberhöhungen auftreten können. Die Länge der Pfade wurde mit *400 µm* festgelegt, wobei jeder Pfad insgesamt durch zweihundert Punkte definiert ist.

Um den Spannungstensor für einen beliebigen Punkt im Modell interpolieren zu können, ist festzustellen in welchem Element bzw. Tetraeder sich der Punkt befindet. Vereinfachend wird dabei von Tetraedern mit ebenen Begrenzungsflächen bzw. von linearen Tetraedern ausgegangen. Bei der Überprüfung wird der Normalabstand des betreffenden Punktes zu allen vier Begrenzungsflächen ermittelt. Die Seitenflächen des Tetraeders wurden so definiert, dass alle Normalvektoren aus dem Tetraeder herauszeigen. Der Interpolationspunkt befindet sich dabei im Tetraeder, wenn alle Normalabstände zu den Begrenzungsflächen innerhalb einer festzulegenden Toleranz negativ sind. Die Anwendung eines geeigneten Toleranzwertes wird aufgrund von Rechenungenauigkeiten notwendig.

Um die erforderliche Rechenzeit zu minimieren wurde die Auswertung wie in Abb. 5.1 dargestellt in mehrere Bereiche unterteilt. Die Zuordnung der Interpolationspunkte wurde mithilfe eines Python Scripts realisiert wobei geschwindigkeitskritische Bereiche in Cython umgesetzt wurden. Dadurch konnte die erforderliche Rechenzeit zur Zuordnung der ca. 300 000 Pfade mit jeweils 200 Punkten auf wenige Minuten reduziert werden. Der zugehörige Progammcode ist im Anhang in Sektion 9.1.5 angeführt.

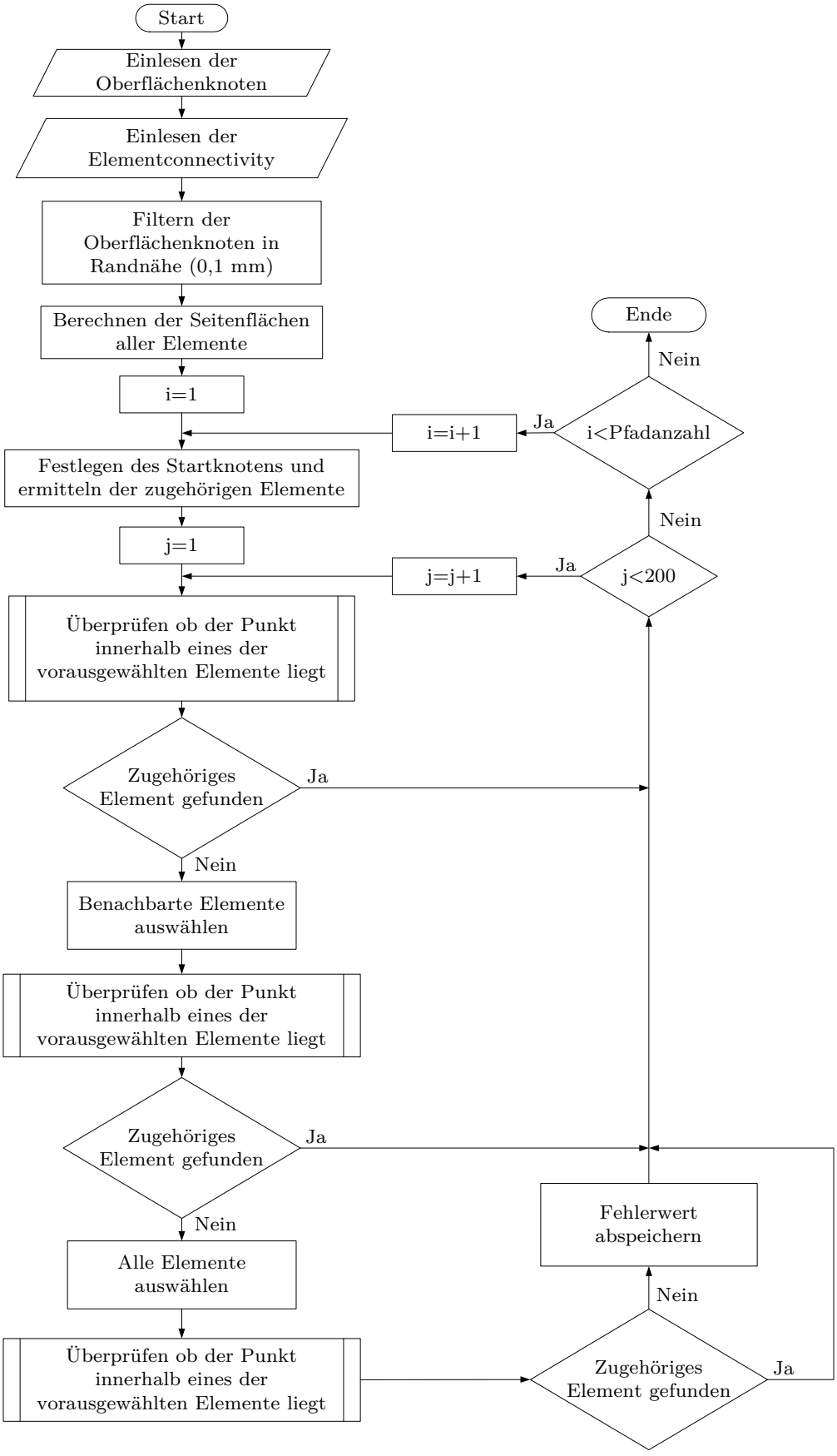


Abb. 5.1: Flussdiagramm zur Pfaderstellung

5.1.2. Superconvergent Patch Recovery

Für die nachfolgende Bewertung der Lebensdauer ist es von wesentlicher Bedeutung Spannungsergebnisse mit minimalem Fehler zu erhalten. Analog zur Berechnung des Spannungsgradienten können auch bei der Ermittlung der Spannungsverläufe in die Tiefe der Oberfläche je nach Netzgröße deutliche Unterschiede auftreten. Moderne Post-Processing Tools unterstützen verschiedene Interpolationsverfahren, beispielsweise können die Spannungswerte entlang des vorgegebenen Pfades per „Nearest-Neighbour“ Verfahren, durch lineare Interpolation oder durch Auswertung der Ansatzfunktionen im betreffenden Element ermittelt werden.

Um den Interpolationsfehler möglichst gering zu halten wurde im vorliegenden Modell ein fortgeschritteneres Verfahren, die Methode des Superconvergent Patch Recovery eingesetzt.

Allgemein variiert das Konvergenzverhalten der FE-Methode je nach gewünschtem Ergebnis mit der jeweiligen Position im Element. Für eindimensionale Elemente lässt sich zeigen, dass die Verschiebungen in den Eckpunkten einen minimalen Fehler aufweisen, während der Verschiebungsgradient bzw. die daraus ermittelten Spannungen an charakteristischen Punkten im Element den geringsten Fehler aufweisen. Diese Punkte entsprechen den Gaußpunkten, die bei der Integration der Ergebnisse zum Einsatz kommen. An diesen Punkten ist die Konvergenz um einen Grad höher als an beliebigen Punkten im Element, sie werden daher als superkonvergent bezeichnet. Im Allgemeinen tritt an den Integrationspunkten der verwendeten quadratischen Tetraeder Elemente der Abaqus-Bibliothek zwar keine Superkonvergenz auf, das Konvergenzverhalten erweist sich dennoch als sehr günstig.

Um Spannungsergebnisse hoher Genauigkeit an jeder beliebigen Stelle im Modell berechnen zu können, schlagen Zienkiewicz und Zhu [ZZ92] [KG07] vor die Elemente zu Patches zusammenzufassen. In diesen Patches wird eine Funktion Glg. (5.1) mit minimalem Fehlerquadrat an die Ergebnisse in den Integrationspunkten angepasst. Der Programmcode zur Ermittlung der Koeffizienten ist in Sektion 9.1.4 dargestellt.

$$\begin{aligned}\sigma = {} & a_0 + a_1 x + a_2 y + a_3 z + a_4 x^2 + a_5 y^2 + a_6 z^2 + a_7 xy + a_8 yz + \\ & a_9 zx + a_{10} x^3 + a_{11} y^3 + a_{12} z^3 + a_{13} x^2 y + a_{14} y^2 z + a_{15} z^2 \\ & + \quad a_{16} x^2 z + a_{17} y^2 x + a_{18} z^2 x + a_{19} xyz\end{aligned} \qquad \text{Glg. (5.1)}$$

Dieses Verfahren sorgt für einen stetigen Spannungsverlauf auch über die Elementgrenzen hinweg und weist zusätzlich ein hervorragendes Konvergenzverhalten auf.

5.1.3. Ermitteln der Spannungsverläufe

Die Patches wurden jeweils um einen Eckknoten der quadratischen Tetraeder Elemente aufgebaut, wobei alle Elemente die an den Knoten angrenzen dem jeweiligen Patch zugeordnet wurden. Als Interpolationsfunktion kam die in Glg. (5.1) dargestellte trikubische Funktion mit zwanzig Koeffizienten zum Einsatz. Um eine eindeutige Lösung des folgenden Minimierungsproblems gewährleisten zu können, muss der Patch daher mindestens fünf Elemente mit jeweils vier Integrationspunkten beinhalten. Aus Stabilitätsgründen wurde dieses untere Limit jedoch auf mindestens sieben Elemente erhöht. Patches, die diese Mindestanzahl an Elementen nicht erreichten wurden als nicht auswertbar markiert. Dieser Vorgang wurde für alle sechs Komponenten des Spannungstensors durchgeführt. Das zur Berechnung der Koeffizienten benötigte Script wurde in Cython umgesetzt und als C++-Bibliothek im Python Script angewendet.

Im Anschluss erfolgte die Berechnung der Spannungen für alle Punkte der in Kapitel 5.1.1 festgelegten Pfade. Jedem Punkt entlang des Spannungspfades wurden vier Patchknoten zugeordnet. Zuerst wurden dabei alle vier Eckknoten des Elements, in dem sich der betreffende Interpolationspunkt befindet, ausgewählt. Falls im Element nicht ausgewertete Eckknoten vorhanden waren wurden diese durch die am nächsten zum Interpolationspunkt gelegenen Patchknoten ergänzt. Zur Berechnung der Spannungswerte wurden die Relativkoordinaten der Interpolationspunkte in die zugehörigen Interpolationsfunktionen eingesetzt. Im Anschluss wurden die vier pro Eintrag im Spannungstensor erhaltenen Werte mithilfe des reziproken Abstands gewichtet gemittelt. Abschließend wurde aus den Spannungstensoren aller Interpolationspunkte die zugehörige Hauptnormalspannung berechnet. Um die erforderliche Rechenzeit zu minimieren wurde sowohl die Berechnung der Spannungswerte als auch der Hauptnormalspannungen in einem in C++ realisierten und mit Cython eingebundenen Python Modul verwirklicht.

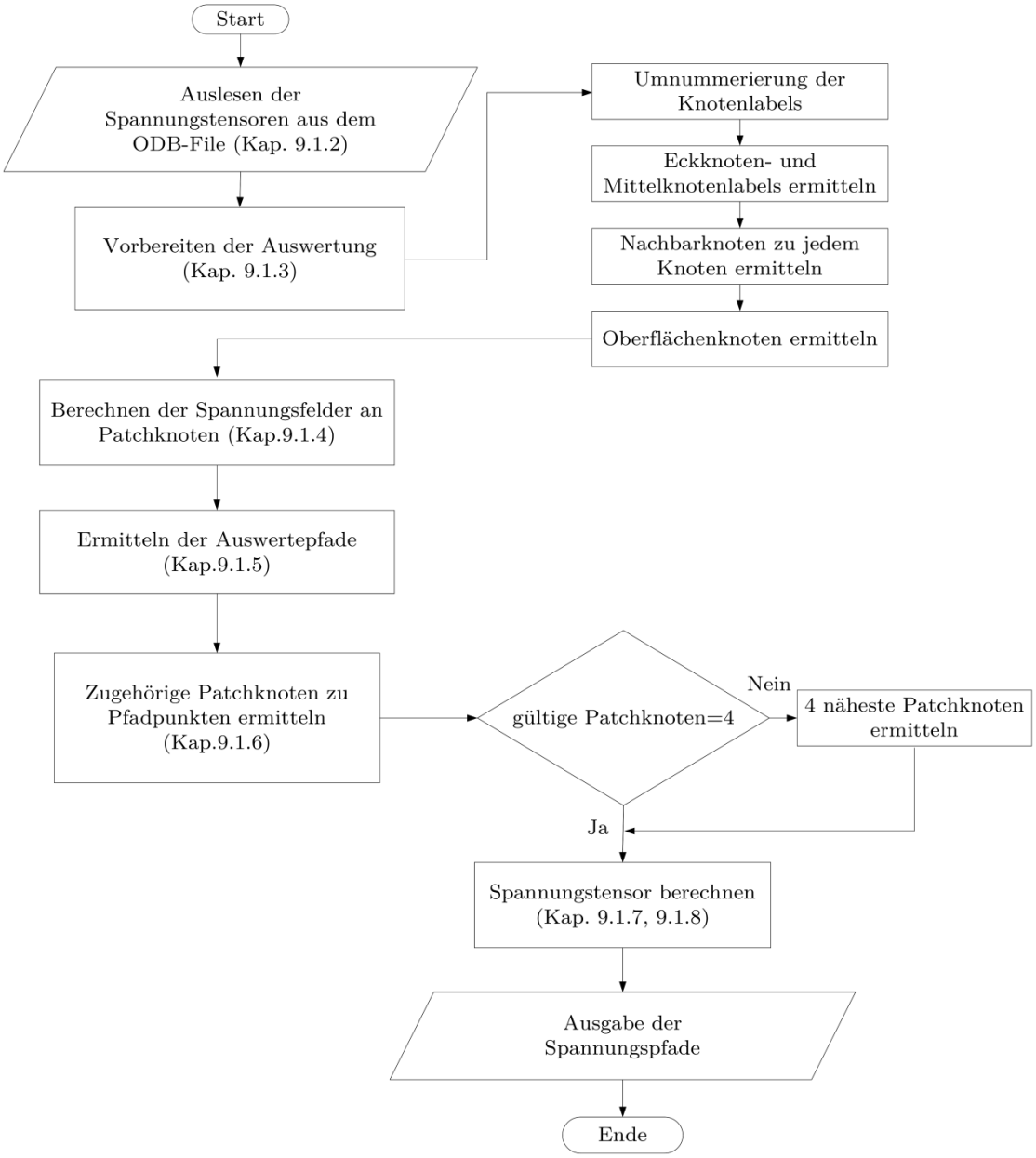


Abb. 5.2: Flussdiagramm zur Ermittlung der Spannungsverläufe normal zur Oberfläche

5.2. Ermittlung der zu erwartenden Lebensdauer

Für die Bewertung der Lebensdauer wurde die im Stand der Technik dargestellte Linienmethode verwendet, da diese mit dem bewährten Rissfortschrittskonzept nach El Haddad übereinstimmt. Ein weiterer Vorteil gegenüber der „Area Methode“ oder weiterführenden Volumenmethoden besteht zusätzlich in der einfachen Anwendbarkeit. Es wird zur Auswertung zuerst der kritische Abstand nach aus den Materialdaten eines defektfreien Werkstoffs bestimmt. Danach wird die mittlere Hauptnormalspannung im Bereich des zweifachen kritischen Abstandes durch Integration gemittelt. Die bewertete Stelle ist dabei als dauerfest zu betrachten, wenn die gemittelte Spannung unterhalb der Dauerfestigkeit liegt. Im vorliegenden Modell dient dabei die Dauerfestigkeit des defektfreien Materials unter dem betrachtetem Spannungsverhältnis Verhältnis als charakteristischer Kennwert. Aus dem Quotienten der gemittelten Spannung und der Dauerfestigkeit des defektfreien Materials ergibt sich nach Glg. (5.2) ein Abminderungsfaktor für jeden betrachteten Oberflächenknoten

$$Abm = \frac{S_{gemittelt}}{S_{ad}} \qquad \text{Glg. (5.2)}$$

An dieser Stelle ist anzumerken, dass beim implementierten Mittelungsverfahren oder Abstandsverfahren ein eigenspannungsfreier Zustand der Randschicht vorausgesetzt wird. Diese prozesstechnische Beeinflussung der Randschicht ist in weiterführenden Arbeiten zu berücksichtigen.

5.2.1. Festlegung der erforderlichen Parameter

Für einen Bereich mit ähnlichem Dendritenarmabstand ermittelte Redik [Red14] bei Versuchen mit gehippten, annähernd porenfreiem Material eine Dauerfestigkeit von *101 MPa* (Zug-Druck, wechselnd). Der Langrissschwellwert ΔK_{th} bei $R=-1$ wurde aus Literaturwerten mit $3{,}4\ MPa\sqrt{m}$ ermittelt. Da die Bauteilabmessungen groß im Vergleich zur Risslänge sind, bzw. die Risse von der Oberfläche ausgehen, wurde der Geometrieeinflussfaktor für die nachfolgende Bewertung mit $Y=1{,}12$ festgelegt.

5.2.2. Statistische Bewertung der erhaltenen lokalen Ergebnisse

Die statistische Verteilung der Abminderungsfaktoren wäre es denkbar ein einfaches Histogramm der Abminderungsfaktoren über alle Oberflächenknoten zu bilden. Dies ist jedoch aufgrund mehrerer Gründe nicht zielführend. Zum einen würde die resultierende Verteilung von der variablen Netzfeinheit abhängen, was technisch keinen Sinn ergibt und zum anderen würde ein und dieselbe Oberflächenkerbe mit mehreren unterschiedlichen Abminderungsfaktoren in die Bewertung eingehen.

Um den statistischen Oberflächeneinfluss beschreiben zu können, werden die auf Oberflächenknoten basierenden Abminderungsfaktoren zu lokalen Minima zusammengefasst. Dafür sind zwei Schwellwerte erforderlich, ein oberer Schwellwert der den maximalen Abminderungsfaktor der noch bewertet wird beschreibt und ein unterer Schwellwert der festlegt bis zu welchem Abminderungsfaktor der Bereich eines lokalen Minimums wachsen darf. Dies wird erforderlich, da es ansonsten zu einer erhebliche Übersegmentierung der Oberfläche kommen würde, da sich an den Rändern eines größeren Bereichs viele kleinere Bereiche die gerade noch den Schwellwert erfüllen anschließen würden.

Dazu wird zuerst der globale, minimale Abminderungsfaktor über alle hundertachzig Belastungsrichtungen ermittelt und daran anschließend nach Glg. (5.3) mit einem Schwellwertfaktor von *SchwF = 0,75* der verwendete Schwellwert bestimmt.

$$SchwW = (1 - Abm_{max}) \cdot SchwF \cdot Abm_{max} \qquad \text{Glg. (5.4)}$$

Folgende Parameter können von der Berechnungsroutine ausgegeben werden:

- Minimaler Abminderungsfaktor innerhalb eines lokalen Minimums und das zugehörige Knotenlabel
- Fläche des lokalen Minimums
- Knotenlabel aller im lokalen Minimum vorhandener Knoten
- Hauptnormalspannung Abminderungsfaktoren aller im lokalen Minimum befindlicher Knoten
- Ein zur Oberflächenkerbe äquivalenter Porendurchmesser

Der minimale Abminderungsfaktor in einem lokalen Minimum wird derzeit als repräsentativ für den gesamten Bereich angenommen. Da der minimale Abminderungsfaktor häufig deutlich unter dem Durchschnitt des betrachteten Bereichs liegt ist dieses Auswerteverfahren als konservativ zu bezeichnen.

Nach entsprechendem experimentellem Vergleich können jedoch durchaus auch andere Parameter zur Auswertung herangezogen werden. Zum Beispiel könnte jeweils der Durchschnitt aus den kritischsten zehn Prozent der Abminderungsfaktoren jedes lokalen Minimas als relevante Größe angesehen werden. Dies würde die Anfälligkeit auf Ausreißer verringern, wäre jedoch nicht mehr derart konservativ.

Zur einfachen Darstellung der Ergebnisse, wurden die lokalen Minima für jeden Lastfall ermittelt und die Position als auch der zugehörige Abminderungsfaktor in ein Excel-File tabellarisch ausgegeben. Hier ist zu beachten, dass in der Regel weit über 2^{16}-Zeileneinträge vorliegen können, sodass eine aktuelle Software-Version zu verwenden ist. Zusätzlich können die Ergebnisse auch in ein bestehendes Abaqus ODB-File importiert werden.

Für die statistische Auswertung können entweder die einzelnen Lastfälle, d.h. in der vorliegenden Routine hundertachtzig Lastfälle gemäß der ein Grad Winkelteilung, als statistisch unabhängig betrachtet und in einer Datenbasis zusammengefasst werden, oder es kann an jedem Knoten der kritischste Abminderungsfaktor als maßgebend betrachtet werden. Abschließend werden die lokalen Minima der Abminderungsfaktoren klassiert und in einem Histogramm dargestellt.

5.3. Anwendungsbeispiel

Die dargestellte komplexe Methodik zur numerischen Bewertung der Oberfläche wird anhand eines Scans einer Probe mit Gussoberfläche dargestellt. Eine Oberfläche kann auf zwei Arten ausgewertet werden. Beim ersten Ansatz wird jeder Lastfall einzeln betrachtet, danach erfolgt die statistische Auswertung über alle 180 Lastfälle. Beim zweiten Ansatz werden alle Lastfälle auf einen Lastfall reduziert. Dazu wird der an jedem Knoten ermittelte minimale Abminderungsfaktor über alle Lastfälle herangezogen.

Auswertung mit definierter Belastungsrichtung

In Abb. 5.3 ist die bereits im Kapitel 4 erwähnte Sandgussoberfläche unter Zugbelastung in null Grad Richtung zu sehen.

Parameter	
Nominalspannung	*100 MPa*
SaD	*101 MPa*
ΔK_{th}	*3,4 MPa √m*
Oberer Schwellwert	*0,9516*
Unterer Schwellwert	*0,858*

Tab. 5.1: Parameter der Auswertung

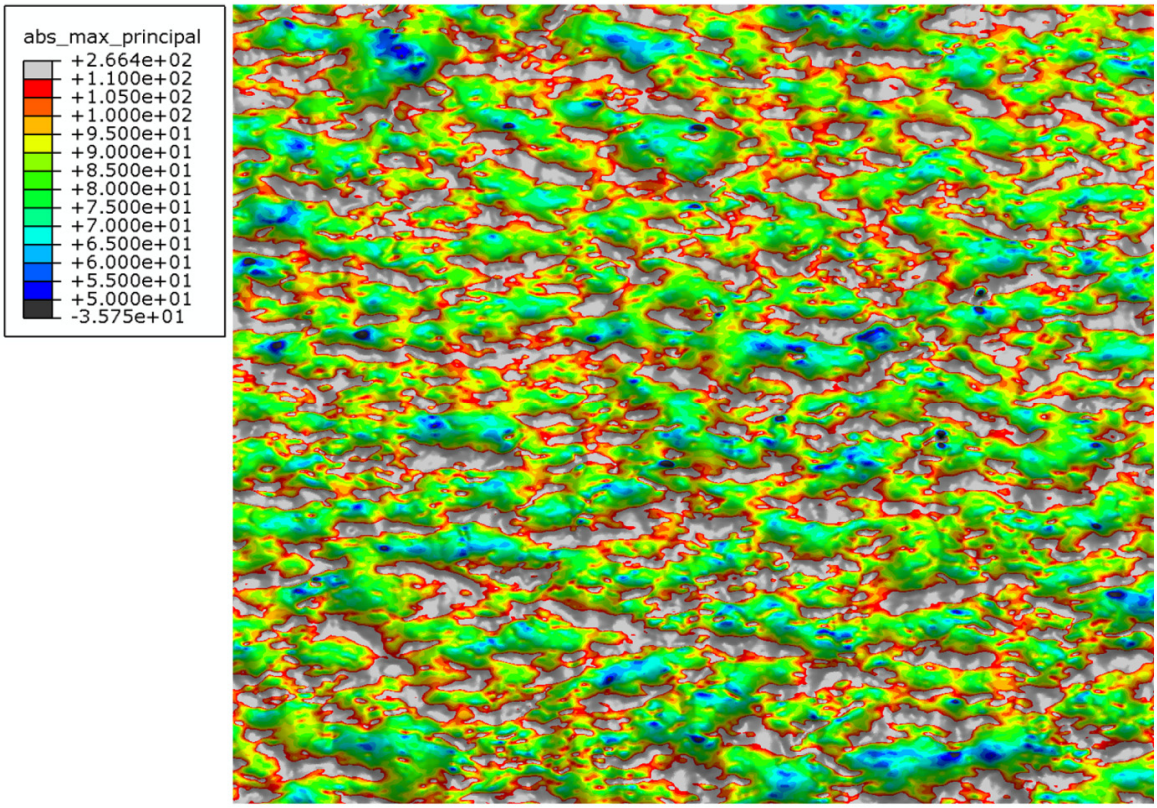


Abb. 5.3: Kerbspannungen an einer Sandgussoberfläche

Die in Abb. 5.4 dargestellten Abminderungsfaktoren weisen einen kontinuierlicheren Verlauf als die Kerbspannungsverläufe auf. Werte kleiner eins definieren eine Abminderung gegenüber der dauerfesten Auslegung. Diese Bereiche sind grau skaliert dargestellt. Zusätzlich sind die Positionen der lokalen Minima der Abminderungsfaktoren, d.h. des höchstgeschädigten Volumens, durch rote Punkte innerhalb der grauen Flächen gekennzeichnet.

Während sich die Kerbspannung an einem Oberflächenknoten zum überwiegenden Teil aus dem lokalen Kerbradius normal zur Belastungsrichtung sowie der Kerbtiefe ergibt, fließt bei der Berechnung des lokalen Abminderungsfaktors zusätzlich noch der Spannungsverlauf normal zur Oberfläche mit ein. Dies erlaubt einen Rückschluss auf das höchstbeanspruchte Volumen.

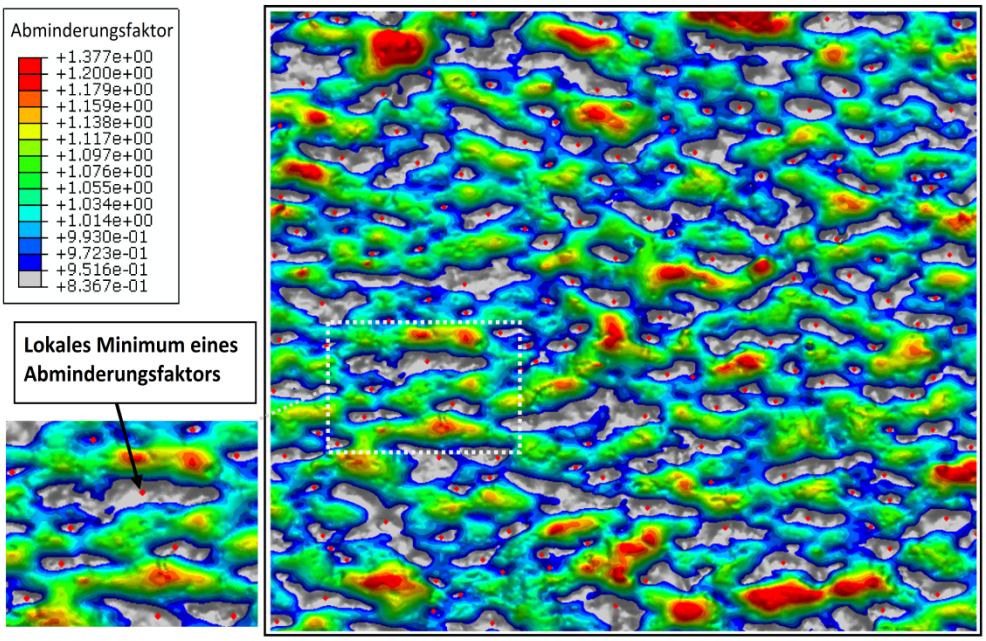


Abb. 5.4: Abminderungsfaktoren an einer Sandgussoberfläche

Durch den Mittelungsansatz haben sehr kleine, scharfe Kerben je nach Materialkennwert einen geringeren Einfluss auf die Lebensdauer, obwohl an diesen Stellen durchaus hohe Kerbfaktoren erreicht werden. Setzt man einen Abminderungsfaktor von *0,95* als Schwellwert, so beträgt der Flächenanteil dieser *28* grau markierten Bereiche etwa *10 %* für die Sandgussoberfläche.

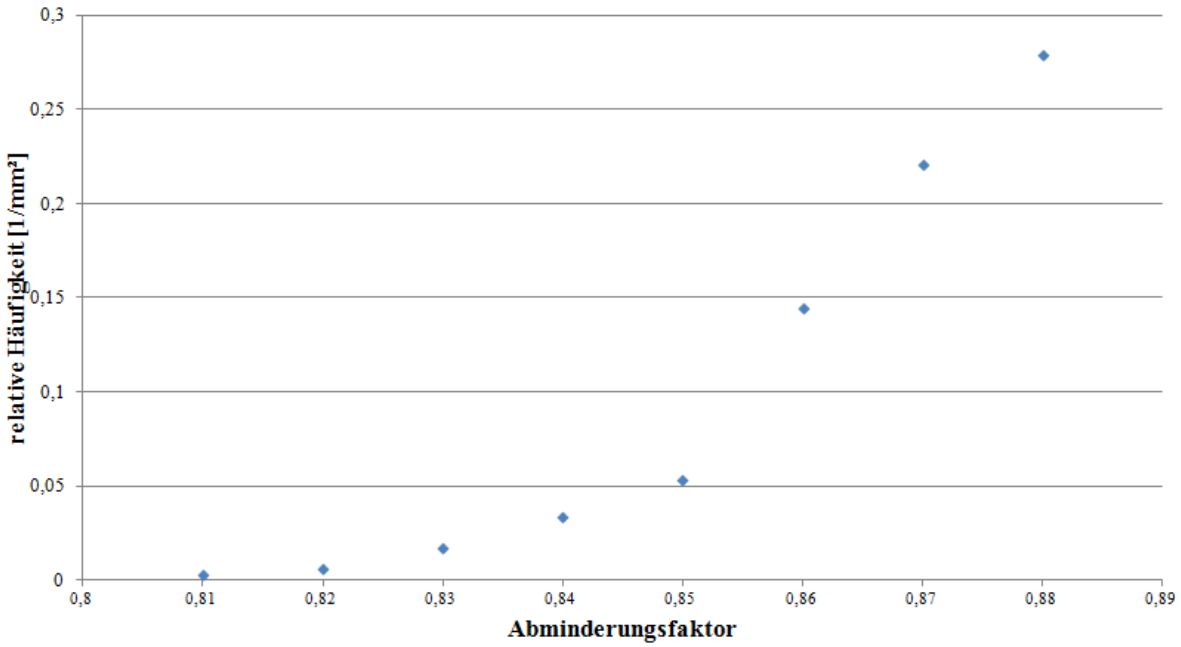


Abb. 5.5: relative Häufigkeit des Abminderungsfaktors

Auswertung mit überlagerten Abminderungsfaktoren

Die bewerteten Hauptnormalspannungen an jedem Knoten basieren wie bereits erwähnt auf dem jeweiligen Maximalwert der Hauptnormalspannungen über alle hundertachtzig Lastfälle. Der minimale Abminderungsfaktor an einem Knoten definiert sich aus dem Minimum über alle Lastfälle. Der maximale Kerbfaktor beträgt *4,06*, der minimale Abminderungsfaktor beträgt 0,528. Die zur Auswertung verwendeten Parameter sind tabellarisch zusammengefasst.

Parameter	
Nominalspannung	*100 MPa*
SaD	*440 MPa*
ΔK_{th}	*7,0 MPa √m*
Oberer Schwellwert	*0,885*
Unterer Schwellwert	*0,862*

Tab. 5.2: Parameter der Auswertung

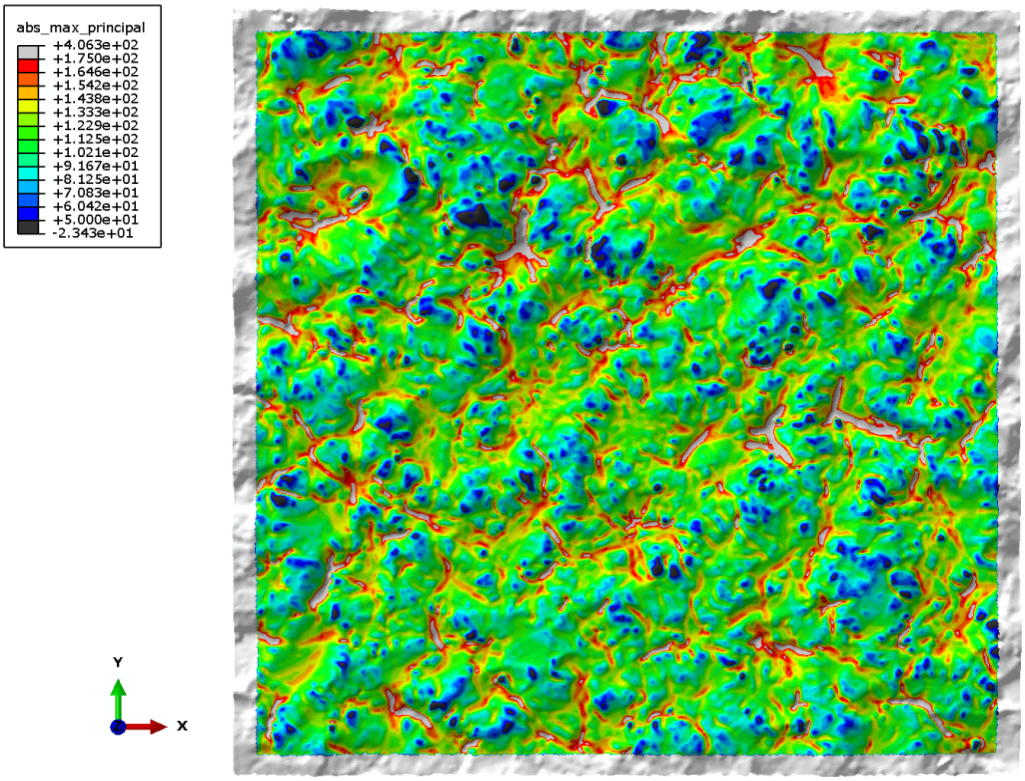


Abb. 5.6: Überlagerte Hauptnormalspannungen an der Krümmeroberfläche

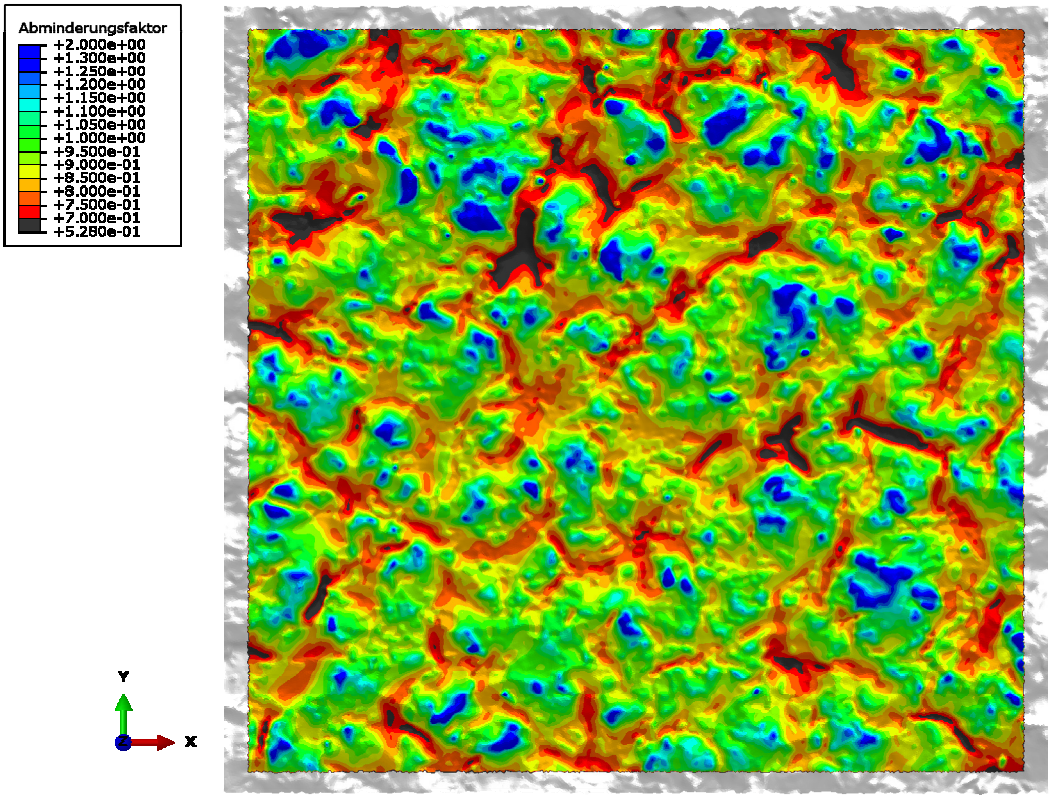


Abb. 5.7: Überlagerte Abminderungsfaktoren der Krümmeroberfläche

Je Senke der Oberfläche wird der lokale Abminderungsfaktor ausgewertet. Es zeigt sich, dass die kritischen Abminderungsfaktoren, d.h. die lokalen Minima dieser Werte, vom absoluten minimalen Abminderugsfaktor von 0,528 bis zu einem Wert von etwa 0,72 mit einer relativen Häufigkeit von etwa *0,1/mm²* auftreten. Dadurch wird auch eine Beschreibung des Größeneinflusses in dieser Auswertung ermöglicht.

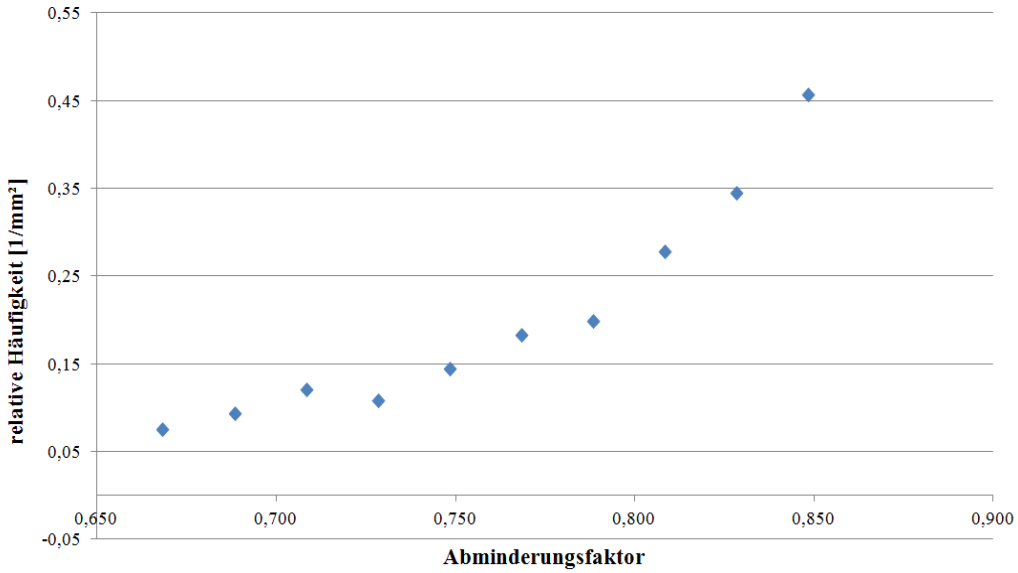


Abb. 5.8: Relative Häufigkeit der Abminderungsfaktoren

6. Zusammenfassung und Ausblick

Die vorgestellte Methodik ermöglichte eine numerische Bewertung von bearbeiteten und unbearbeiteten Oberflächen auf Basis einer linear-elastischen Berechnung von Standardlastfällen unter Berücksichtigung der realen Oberflächentopographie. Im Gegensatz zu etablierten Verfahren sind dazu keine empirischen Einflussfaktoren notwendig. Als Materialparameter werden lediglich der Langrissschwellwert und die Schwingfestigkeit des defektfreien Materials benötigt, welche aufgrund ihrer Bedeutung bei der Bruchmechanischen Bewertung in aller Regel vorliegen. Ein adaptiver Gauss-Filter ermöglicht es gezielt Welligkeiten durch Vorgabe von benutzerdefinierter Grenzwellenlängen zu unterdrücken. Eventuell vorhandene Formabweichungen können per Abwicklung der Oberfläche korrigiert werden. Dadurch wird eine ganzheitliche Bewertung der Oberflächenbeschaffenheit ermöglicht.

Durch die Anwendung des Linienkonzepts nach Taylor wird die schwingfeste Bewertung der Oberfläche auf Basis einer linear-elastischen FE-Berechnung ermöglicht. Der modulare Aufbau der programmtechnischen Umsetzung ermöglicht es zukünftig auch andere Bewertungsverfahren wie ein Spannungsabstandskonzept oder ein volumenbasiertes Konzept auf der vorliegenden Arbeit aufzubauen. Durch den Mittelungsansatz haben sehr kleine, scharfe Kerben je nach Materialkennwerten wenig Einfluss auf die Lebensdauer, obwohl an diesen Stellen durchaus hohe Kerbfaktoren erreicht werden. Aus dieser Erkenntnis kann die getroffene Annahme, dass Oberflächenstrukturen unter einer definierten Grenzwellenlänge nicht mehr berücksichtigt werden müssen, bestätigt werden. Zusammenfassend treten beispielsweise bei Materialien mit höherer Schwingfestigkeit des defektfreien Materials bzw. geringerem Risswiderstand folgende Effekte auf:

- Der maximale Abminderungsfaktor erhöht sich
- Die Bauteilfestigkeit sinkt überproportional stärker an scharfen, geometrisch klein ausgeprägten Kerben

Der Schwerpunkt dieser Arbeit lag in der Konzeption und Umsetzung der zahlreichen C++ und Python-basierten Routinen, welche eine numerische Bewertung von Oberflächen sowohl hinsichtlich der verfügbaren IT/HPC-Ressourcen als auch der notwendigen User-Zeit ermöglichte. Weiterführend werden diese erstellten Routinen zur Bewertung von diversen technischen Oberflächen am Lehrstuhl für Allgemeinen Maschinenbau eingesetzt und ermöglichen somit einen erhöhten wissenschaftlichen Output zwischen prüftechnisch basierten Schadensfällen von an Oberflächen initiierten technischen Rissen und der realen Geometrie. Anzumerken ist, dass die vorgestellte Methodik auch durch einfache Erweiterungen in der Lage ist den fertigungstechnischen Eigenspannungszustand in der Randschicht zu integrieren. Dies ist insbesondere für zukünftige Anwendungen hinsichtlich zyklische stabilisierten Mittelspannungen in der Randschicht von Interesse.

Aufgrund der direkten Komptabilität mit dem etablierten Verfahren zur Porenbewertung nach El Haddad ist es prinzipiell auch zukünftig möglich Mikrokerben an der Oberfläche mit Mikroporen unter der Oberfläche hinsichtlich ihres Einflusses auf die Schwingfestigkeit zu vergleichen. Einschränkend muss hier allerdings erwähnt werden, dass dies nur dann vollständig zutrifft, wenn die mechanischen Spannungsfelder von Defekten und Mikrokerben auf der Oberfläche nicht direkt wechselwirken. Diese Arbeit stellt allerdings eine solide Grundlage dar, um die Wechselwirkunken von Defekten mit der Oberfläche beschreiben zu können.

7. Literaturverzeichnis

BOB06: Bobenko, Aleksandr I.; *Differentialgeometrie von Kurven und Flächen,* 2006.

Eic89: Eichlseder, Wilfried; Rechnerische Lebensdaueranalyse von Nutzfahrzeugkomponenten mit der Finite Elemente Methode, Graz, 1989, Dissertation, Graz.

ETS79: El Haddad, M. H.; Topper, T. H.; Smith, K. N.; Prediction of non propagating cracks, in: *Engineering Fracture Mechanics*, 11, 1979, S. 573–584.

FOR12: Forschungskuratorium Maschinenbau; Rechnerischer Festigkeitsnachweis für Maschinenbauteile aus Stahl, Eisenguss- und Aluminiumwerkstoffen, 6. Auflage, Frankfurt am Main, 2012.

GR09: Geuzaine, Christophe; Remacle, Jean-François; Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, in: *International Journal for Numerical Methods in Engineering*, 79, 2009, S. 1309–1331.

KE92: Kujawski, D.; Ellyin, F.; A microsturally motivated model for short crack growth rate. In: Miller, K. J.; de los Rios, E. R. (eds.): *Short Fatigue Crack Growth, ESIS 13*. Mechanical Engineering Publications, London, 1992, S. 391-405

KG07: Khoei, A. R.; Gharehbaghi, S. A.; *The superconvergence patch recovery technique and data transfer operators in 3D plasticity problems*, in: Finite Elements in Analysis and Design, 43, 2007, S. 630–648.

KIT76: Kitagawa, H.; Takahashi, S.; Applicability of fracture mechanics to very small cracks or the cracks in the early stage, in: *Proceedings of the 2nd International Conference on Mechanical Behavior of Materials*, 1976.

KL92: Klesnil, M.; Lukac, P.; Fatigue of metallic materials, in: Materials science monographs, 1992.

TAY99: Taylor, D.; Geometrical effects in fatigue: A unifying theoretical model, in: *International Journal of Fatigue*, 21, 1999, S. 413–420.

Mur02: Murakami, Y.; Metal fatigue: Effects of small defects and nonmetallic inclusions, 2.e Auflage, Amsterdam u.a., Elsevier, 2002.

Neu68: Neuber, Heinz; Über die Berücksichtigung der Spannungskonzentration bei Festigkeitsberechnungen, in: *Konstruktion*, 20, 1968, S. 245–251.

Pet53: Peterson, Rudolph Earl; Stress concentration design factors, New York u.a., Wiley, 1953.

Red14: Redik, S.; The influence of microstructure on the high cycle fatigue behavior of AlSi-casting alloys, 2014, Dissertation.

SZY04: Szymon Rusinkiewicz; *Estimating Curvatures and Their Derivatives on Triangle Meshes*, in: Symposium on 3D Data Processing, Visualization, and Transmission, 2004.

WAD71: Waddoups, M. E.; Eisenmann, J. R.; Kaminski, B. E.; Macroscopic Fracture Mechanics of Advanced Composite Materials, in: *Journal of Composite Materials*, 5, 1971, S. 446–454.

ZZ92: Zienkiewicz, O. C.; Zhu, J. Z.; The superconvergent patch recovery anda posteriori error estimates. Part 1: The recovery technique, in: *International Journal for Numerical Methods in Engineering*, 33, 1992, S. 1331–1364.

8. Abbildungsverzeichnis

9. Anhang

9.1. Berechnung der Spannungsverläufe

9.1.1. Hauptfunktion

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import numpy as np
import h5py
import os,sys
pathname = os.path.dirname(sys.argv[0])
Path=os.path.abspath(pathname)
sys.path.append(Path+'\\py_Module')

from Auswertung_vorbereiten import Auswertung_vorbereiten
from getKoeff import getKoeff
from Tetraeder import get_Pfade
from get_Bezugsknoten import get_SP_Nodes_Pfad
from get_Bezugsknoten_mod import get_SP_Nodes_Pfad_ohne_hinzufuegen
from get_Spannungen_Punkt import get_Spannungen_Punkt
from get_Connectivity_Schale import*
import get_Stress
import abq_py_extension as abq

import Tkinter as tk
import tkFileDialog as tkF
import ConfigParser
import ConfigSectionMap

def main():
  # Read Config File-------------------------------------------------------------------------
  Config = ConfigParser.ConfigParser()
  file=Config.read("Spannungsauswertung.ini")

  ABQ_Syntax = ConfigSectionMap(Config,'General_Parameters')['abq_syntax']
  Zuschnitt = float(ConfigSectionMap(Config,'General_Parameters')['zuschnitt'])

  # Einlesen der Pfade-----------------------------------------------------------------------
  root = tk.Tk()
  root.withdraw()

  Abaqus_Zug_0_90 = dict(defaultextension='.odb',filetypes=[('Abaqus_Zug_0_90_ODB','*.odb')])
  odbPath_0_90=tkF.askopenfilenames(**Abaqus_Zug_0_90)
  odbPath_0_90=odbPath_0_90[0]

  Abaqus_Schub = dict(defaultextension='.odb',filetypes=[('Abaqus_Schub_ODB','*.odb')],
    initialdir=os.path.dirname(odbPath_0_90))
  odbPath_Schub=tkF.askopenfilenames(**Abaqus_Schub)
  odbPath_Schub=odbPath_Schub[0]

  Abaqus_Oberflaeche = dict(defaultextension='.inp',filetypes=[('Oberflaeche','*.inp')],
    initialdir=os.path.dirname(odbPath_0_90))
  Pfad_Oberflaeche=tkF.askopenfilenames(**Abaqus_Oberflaeche)
  Pfad_Oberflaeche=Pfad_Oberflaeche[0]

  SaveOptions = dict(initialdir=os.path.dirname(odbPath_0_90))
  Working_Path=tkF.askdirectory(**SaveOptions)

  root.deiconify()
  root.destroy()
  print('Working Path: ' + Working_Path)

  #Scratchen
  Erfolg=os.system(ABQ_Syntax +' '+Path +'\\abq_py_Module\get_Spannungen.py '+ odbPath_0_90 +
     ' ' + odbPath_Schub + ' '+Working_Path)
```

```
  #Schale einlesen
  [Connectivity_Schale_all,Knoten_Label_Schale_all,Knoten_Koord_Schale_all]=
    get_Connectivity_Schale(Pfad_Oberflaeche)

  #Einlesen der Inputvaribalen
  Inputvariablen=np.load(Working_Path+'\\Inputvaribalen.npz')
  Knotenlabels_all= Inputvariablen['KnotenLabels']
  Knotenkoordinaten_all= Inputvariablen['KnotenKoordinaten']
  Elementlabels= Inputvariablen['ElementLabels']
  Connectivity= Inputvariablen['ElementConnectivity']

  #Vorbereiten der Auswertung
  [ElementeU,Knotenkoordinaten_E,Knotenlabels_E,Knotenkoordinaten_M,Knotenlabels_M,
    IndexMatrix_E,Split_Ind_E,IndexMatrix_M,Split_Ind_M,Critical_Nodes]=
    Auswertung_vorbereiten(Knotenlabels_all,Knotenkoordinaten_all,Elementlabels,Connectivity)

  #Einlesen der Lastfaelle
  Spannungen=np.load(Working_Path+'\\Spannungen.npz')
  S_Zug_0=Spannungen['Zug_0_Grad']
  S_Zug_90=Spannungen['Zug_90_Grad']
  S_Schub=Spannungen['Schub']

  #Integrationspunktkoordinaten bestimmen
  IndexMatrix_E_Py=np.split(IndexMatrix_E,Split_Ind_E)
  IntPktKoord=get_Stress.get_Int_Pkt_Koord(ElementeU,Knotenkoordinaten_E)

  #Oberflaechenknoten sind auf false gesetzt
  Critical_Nodes_ohne_Oberflaeche=np.array(Critical_Nodes)
  Critical_Nodes[:]=1

  # Berechnung der Koeffizienten
  [Koeffizienten_Zug_0,Normierung,Critical_Nodes]=
    getKoeff(S_Zug_0,IntPktKoord,Knotenkoordinaten_E,Knotenlabels_E,
    IndexMatrix_E_Py,Critical_Nodes)
  [Koeffizienten_Zug_90,Normierung,Critical_Nodes]=
    getKoeff(S_Zug_90,IntPktKoord,Knotenkoordinaten_E,Knotenlabels_E,
    IndexMatrix_E_Py,Critical_Nodes)
  [Koeffizienten_Schub,Normierung,Critical_Nodes]=
    getKoeff(S_Schub,IntPktKoord,Knotenkoordinaten_E,Knotenlabels_E,
    IndexMatrix_E_Py,Critical_Nodes)

  # Pfade ermitteln
  Critical_Nodes[:]=1
  [Pfade,Pfad_Elements,Knoten_Label_Schale,Knoten_Label_Schale_orig]=
    get_Pfade(Knotenlabels_all,Knotenkoordinaten_all,Elementlabels,
    Connectivity,Pfad_Oberflaeche,Zuschnitt)

  # Zugehörige Patchknoten zu den Pfaden finden
  [Bezugsknoten_Pfade,m_Probleme]=get_SP_Nodes_Pfad(Critical_Nodes,Knotenkoordinaten_E,
    ElementeU,IndexMatrix_E,Split_Ind_E,Pfad_Elements,Pfade)

  # Spannungswerte ermitteln (alle 3 Standardlastfaelle)
  Spannungen_Zug_0=get_Spannungen_Punkt(Koeffizienten_Zug_0,Pfade,Bezugsknoten_Pfade,
    Knotenkoordinaten_E,Normierung)
  Spannungen_Zug_90=get_Spannungen_Punkt(Koeffizienten_Zug_90,Pfade,Bezugsknoten_Pfade,
    Knotenkoordinaten_E,Normierung)
  Spannungen_Schub=get_Spannungen_Punkt(Koeffizienten_Schub,Pfade,Bezugsknoten_Pfade,
    Knotenkoordinaten_E,Normierung)

  #Speichern der Zwischenergebnisse
  np.savez(Working_Path+'\\Zwischenergebnisse.npz',Spannungen_Zug_0=Spannungen_Zug_0,
    Spannungen_Zug_90=Spannungen_Zug_90,Spannungen_Schub=Spannungen_Schub,
    Knoten_Label_Schale=Knoten_Label_Schale,Knoten_Label_Schale_orig=Knoten_Label_Schale_orig,
    Connectivity_Schale_all=Connectivity_Schale_all,
    Knoten_Label_Schale_all=Knoten_Label_Schale_all,
    Knoten_Koord_Schale_all=Knoten_Koord_Schale_all)

if __name__ == "__main__":
  main()
```

9.1.2. Auslesen der Spannungstensoren aus dem Abaqus-ODB-File

```
from odbAccess import *
import numpy as np
import sys

def Scratchen():
  odbPath_0_90 = sys.argv[1]
  odbPath_Schub = sys.argv[2]
  Temp_Path = sys.argv[3]
  odb=openOdb(odbPath_0_90,readOnly=True)

  #######
  #Koord=odb.steps['STEP1'].frames[0].fieldOutputs['COORD']
  #IntPktKoord=Koord.bulkDataBlocks[0].data
  #######

  KnotenObj=odb.rootAssembly.instances['PART-1-1'].nodes
  KnotenLabels=np.array([i.label for i in KnotenObj],int)
  KnotenKoordinaten=np.array([i.coordinates for i in KnotenObj],float)

  ElementObj=odb.rootAssembly.instances['PART-1-1'].elements
  ElementLabels=[]
  ElementConnectivity=[]
  for i in xrange(0,len(ElementObj)):
    if (ElementObj[i].type=='C3D10'):
      ElementLabels.append(ElementObj[i].label)
      ElementConnectivity.append(ElementObj[i].connectivity)
  ElementLabels=np.array(ElementLabels,int)
  ElementConnectivity=np.array(ElementConnectivity,int)

  np.savez(Temp_Path+'\\Inputvaribalen.npz',KnotenLabels=KnotenLabels,
    KnotenKoordinaten=KnotenKoordinaten,ElementLabels=ElementLabels,
    ElementConnectivity=ElementConnectivity)

  del ElementLabels,ElementConnectivity,KnotenLabels,KnotenKoordinaten

  # Auslesen der Zuglastfaelle
  Steps=odb.steps.keys()
  Spannungstensor=odb.steps[Steps[0]].frames[1].fieldOutputs['S']
  Spannungstensor = Spannungstensor.getSubset(elementType = 'C3D10')
  Zug_0_Grad=Spannungstensor.bulkDataBlocks[0].data
  Spannungstensor=odb.steps[Steps[1]].frames[1].fieldOutputs['S']
  Spannungstensor = Spannungstensor.getSubset(elementType = 'C3D10')
  Zug_90_Grad=Spannungstensor.bulkDataBlocks[0].data
  odb.close()

  # Auslesen des Schublastfalls
  odb=openOdb(odbPath_Schub,readOnly=True)
  Steps=odb.steps.keys()
  Spannungstensor=odb.steps[Steps[0]].frames[1].fieldOutputs['S']
  Spannungstensor = Spannungstensor.getSubset(elementType = 'C3D10')
  Schub=Spannungstensor.bulkDataBlocks[0].data
  odb.close()

  # Speichern der Arrays ins Numpy Format
  np.savez(Temp_Path+'\\Spannungen.npz',Zug_0_Grad=Zug_0_Grad,Zug_90_Grad=Zug_90_Grad,
    Schub=Schub)

if __name__ == "__main__":
  Scratchen()
```

9.1.3. Vorbereiten der Auswertung

```
import numpy as np
def Auswertung_vorbereiten(Knotenlabels_all,Knotenkoordinaten_all,Elementlabels,Connectivity):
  #IntegrationsPunktKoord=Input['IntPktKoord']
  #IntegrationsPunktKoord=IntegrationsPunktKoord.reshape(Elementlabels.shape[0],12)

  # Das Gradientenfeld wird nur an den Eckknoten berechnet
  Connectivity_E=Connectivity[:,0:4]
  Connectivity_M=Connectivity[:,4:10]

  # Die Knotenlabels der Elemente entsprechen nach der Umnummerierung den Indizes
  der Spannungswerte im Array
  Knotenlabels_P, ElementeU = np.unique(Connectivity_E,return_inverse=True)
  ElementeU=ElementeU.reshape(Connectivity_E.shape)

  # Emitteln der Eckknotenlabels als auch der Eckknotenkorrdinaten
  ind=np.in1d(Knotenlabels_all,Knotenlabels_P,assume_unique=False)
  Knotenlabels_E=Knotenlabels_all[ind]
  Knotenkoordinaten_E=Knotenkoordinaten_all[ind]
  del ind

  Knotenlabels_M = np.unique(Connectivity_M,return_inverse=False)
  ind=np.in1d(Knotenlabels_all,Knotenlabels_M,assume_unique=False)
  Knotenlabels_M=Knotenlabels_all[ind]
  Knotenkoordinaten_M=Knotenkoordinaten_all[ind]
  del ind

  # Feststellen welche Elemente an einem Knoten "haengen"
  # Indexmatrix erstellen
  [IndexMatrix_E,Split_Ind_E]=get_index_Matrix(Elementlabels,Connectivity_E)
  [IndexMatrix_M,Split_Ind_M]=get_index_Matrix(Elementlabels,Connectivity_M)

  # Indizes der Oberflaechenknoten bestimmen
  O_Knoten=Oberflaechenknoten_bestimmen_ohne_MK(ElementeU)
  Critical_Nodes=np.array(np.ones(Knotenlabels_E.shape[0]),dtype=bool)
  #An den Oberflaechen werden keine Koeffizienten ermittelt
  Critical_Nodes[O_Knoten]=0

  return [ElementeU,Knotenkoordinaten_E,Knotenlabels_E,Knotenkoordinaten_M,Knotenlabels_M,
    IndexMatrix_E,Split_Ind_E,IndexMatrix_M,Split_Ind_M,Critical_Nodes]

def Oberflaechenknoten_bestimmen_ohne_MK(np_Connectivity):
  Face=np.concatenate((np_Connectivity[:,[0,1,2]],np_Connectivity[:,[0,3,1]],
    np_Connectivity[:,[1,3,2]],np_Connectivity[:,[2,3,0]]))
  ind=Oberflaeche(Face)
  return np.unique(Face[ind])

def argSortSpalten(Array):
  idx1=Array[:,2].argsort()
  idx2=Array[idx1,1].argsort(kind='mergesort')
  idx2=idx1[idx2]
  idx3=Array[idx2,0].argsort(kind='mergesort')
  return(idx2[idx3])

def Oberflaeche(Facet):
  Facet.sort()
  idx=argSortSpalten(Facet)
  Facet=Facet[idx]
  b = np.ascontiguousarray(Facet).view(np.dtype((np.void,
    Facet.dtype.itemsize * Facet.shape[1])))
  _, idx_2,counts = np.unique(b, return_index=True,return_counts=True)
  einfach=np.where(counts<2)
  idx_2=idx_2[einfach[0]]
  jj=idx[idx_2]
  return jj
```

```
def Oberflaeche_alt(Facet):
  Facet.sort()
  jj=[]
  idx=argSortSpalten(Facet)
  Facet=Facet[idx]
  print(Facet.shape)
  for i in xrange(1,Facet.shape[0]-1):
    if (Facet[i][0] != Facet[i-1][0] and Facet[i][0] != Facet[i+1][0])
      or (Facet[i][1] != Facet[i-1][1] and Facet[i][1] != Facet[i+1][1])
      or (Facet[i][2] != Facet[i-1][2] and Facet[i][2] != Facet[i+1][2]):

      jj.append(i)

  jj=idx[jj]
  return jj

def get_index_Matrix(Elementlabels,Connectivity):
  index=np.arange(Elementlabels.shape[0])
  if (Connectivity.shape[1]==4):
    # Index beschreibt das Verhaeltnis von Elementlabel zu Knotenlabel
    IndexMatrix=np.concatenate((index,index,index,index))  if (Connectivity.shape[1]==6):
    IndexMatrix=np.concatenate((index,index,index,index,index,index))
  Connectivity_flat=Connectivity.flatten(order='F') #Inkompatibel mit Abaqus Python
  ind=np.argsort(Connectivity_flat)
  Connectivity_flat=Connectivity_flat[ind]
  IndexMatrix=IndexMatrix[ind]

  # # Es wird eine Liste erzeugt die Elemente enthaelt in der ein Knoten liegt
  Split_Ind=np.nonzero(np.diff(Connectivity_flat))[0]
  Split_Ind=Split_Ind+1 #!!!!!!!!!Wichtig!!!!!!!!!#
  return IndexMatrix,Split_Ind
```

9.1.4. Berechnung der Spannungsfelder (Superconvergent-Patch-Recovery)

```
import numpy as np

def getKoeff(Spannung_Int_Pkt_Input,IntegrationsPunktKoord,Knotenkoordinaten_E,
  Knotenlabels_E,IndexMatrix_E,Critical_Nodes):

  Elementanzahl=Spannung_Int_Pkt_Input.shape[0]/4
  IntegrationsPunktKoord=IntegrationsPunktKoord.reshape(Elementanzahl,12)
  Spannung_Int_Pkt=np.zeros((6,Elementanzahl,4))
  for i in xrange(0,6):
    Spannung_Int_Pkt[i,:,:]=Spannung_Int_Pkt_Input[:,i].reshape(Elementanzahl,4)

  Koeffizienten=np.zeros((6,Knotenlabels_E.shape[0],20))
  Normierung=np.zeros((Knotenlabels_E.shape[0],3))
  for i in xrange(0,len(Knotenlabels_E)):
    #aktuelle Knotenkoordinate auslesen
    Knotenkoordinate=np.array(Knotenkoordinaten_E[i],dtype=np.float64)
    Elemente=IndexMatrix_E[i] #Einlesen der benachbarten Elemente (Index)
    # An dieser Stelle Nachbarelemente hinzufuegen
    #Auslesen der Integrationspunktkoordinaten
    Int_Koord=np.concatenate(IntegrationsPunktKoord[Elemente]
    Int_Koord=Int_Koord.reshape(Int_Koord.shape[0]/3,3)

    Sigma=np.zeros((6,len(Elemente)*4)) # 4 Integrationspunkte
    for j in xrange(0,6):
      Sigma[j,:]=np.concatenate(Spannung_Int_Pkt[j,Elemente])
    Int_Punkte=np.array(Int_Koord,dtype=np.float64)
    # Koordinaten der Integrationspunkte auf 0 normieren
    Int_Punkte[:,0]=Int_Punkte[:,0]-Knotenkoordinate[0]
    Int_Punkte[:,1]=Int_Punkte[:,1]-Knotenkoordinate[1]
    Int_Punkte[:,2]=Int_Punkte[:,2]-Knotenkoordinate[2]

    # x,y,z normieren
    Normierung[i,0]=np.max(Int_Punkte[:,0])-np.min(Int_Punkte[:,0])
    Normierung[i,1]=np.max(Int_Punkte[:,1])-np.min(Int_Punkte[:,1])
    Normierung[i,2]=np.max(Int_Punkte[:,2])-np.min(Int_Punkte[:,2])
    Int_Punkte[:,0]=Int_Punkte[:,0]/Normierung[i,0]
    Int_Punkte[:,1]=Int_Punkte[:,1]/Normierung[i,1]
    Int_Punkte[:,2]=Int_Punkte[:,2]/Normierung[i,2]

    P=KoeffMatrix_C2(Int_Punkte)
    for j in xrange(0,6):
      Koeffizienten[j,i]=np.linalg.lstsq(P,Sigma[j,:],rcond=0.005)[0]
  return Koeffizienten,Normierung,Critical_Nodes

def KoeffMatrix_C2(Koord):
  Matrix=np.zeros((Koord.shape[0],20))
  Matrix[:,0]=1.
  Matrix[:,1]=Koord[:,0]
  Matrix[:,2]=Koord[:,1]
  Matrix[:,3]=Koord[:,2]
  Matrix[:,4]=np.power(Koord[:,0],2)
  Matrix[:,5]=np.power(Koord[:,1],2)
  Matrix[:,6]=np.power(Koord[:,2],2)
  Matrix[:,7]=Koord[:,0]*Koord[:,1]
  Matrix[:,8]=Koord[:,1]*Koord[:,2]
  Matrix[:,9]=Koord[:,2]*Koord[:,0]
  Matrix[:,10]=np.power(Koord[:,0],3)
  Matrix[:,11]=np.power(Koord[:,1],3)
  Matrix[:,12]=np.power(Koord[:,2],3)
  Matrix[:,13]=np.power(Koord[:,0],2)*Koord[:,1]
  Matrix[:,14]=np.power(Koord[:,1],2)*Koord[:,2]
  Matrix[:,15]=np.power(Koord[:,2],2)*Koord[:,0]
  Matrix[:,16]=np.power(Koord[:,0],2)*Koord[:,2]
  Matrix[:,17]=np.power(Koord[:,1],2)*Koord[:,0]
  Matrix[:,18]=np.power(Koord[:,2],2)*Koord[:,1]
  Matrix[:,19]=Koord[:,0]*Koord[:,1]*Koord[:,2]
  return Matrix
```

```
def KoeffMatrixC1(Koord):
  Matrix=np.zeros((Koord.shape[0],10))
  Matrix[:,0]=1.
  Matrix[:,1]=Koord[:,0]
  Matrix[:,2]=Koord[:,1]
  Matrix[:,3]=Koord[:,2]
  Matrix[:,4]=np.power(Koord[:,0],2)
  Matrix[:,5]=np.power(Koord[:,1],2)
  Matrix[:,6]=np.power(Koord[:,2],2)
  Matrix[:,7]=Koord[:,0]*Koord[:,1]
  Matrix[:,8]=Koord[:,1]*Koord[:,2]
  Matrix[:,9]=Koord[:,2]*Koord[:,0]
  return Matrix

def get_Spannungen_Hauptknoten(Critical_Nodes,Knotenlabels_E,Knotenkoordinaten_E,
  ElementeU,IndexMatrix,Koeffizienten):

  #Ausrechnen der Knotenspannungen fuer die Eckknoten
  Knotenspannungen=np.zeros((len(Knotenlabels_E),6))
  for i in xrange(0,len(Knotenlabels_E)):
    if Critical_Nodes[i]: #Knoten die einen Patch beschreiben
      P=np.zeros((1,20),dtype=np.float64) #da auf 0 normiert
      P[0,0]=1
      for j in xrange(0,6):
        Knotenspannungen[i,j]=np.sum(P*Koeffizienten[j,i,:])
    else:
      Elemente_Patch=IndexMatrix[i]
      Knoten_Patches=np.unique(ElementeU[Elemente_Patch])

      #Knoten an denen Koeffizienten berechnet wurden
      Knoten_Patches=Knoten_Patches[Critical_Nodes[Knoten_Patches]]
      if (Knoten_Patches.shape[0]==0):
        Elemente_Patch=IndexMatrix[i]
        Knoten_Patches=np.unique(ElementeU[Elemente_Patch])
        Elemente_Patch=np.array((),int)
        for y in xrange(0,Knoten_Patches.shape[0]):
          Elemente_Patch=np.concatenate((Elemente_Patch,IndexMatrix[Knoten_Patches[y]]))
        Knoten_Patches=np.unique(ElementeU[Elemente_Patch])
        Knoten_Patches=Knoten_Patches[Critical_Nodes[Knoten_Patches]]

      Knoten_Patches_Koord=Knotenkoordinaten_E[Knoten_Patches]
      BezugsknotenKoord=Knotenkoordinaten_E[i]

      Knotenspannungen[i,:]=get_Spannungsberechnung(BezugsknotenKoord,
        Knoten_Patches,Knoten_Patches_Koord,Koeffizienten,Normierung)

  return Knotenspannungen

def get_Spannungen_Mittelknoten(Critical_Nodes,Knotenlabels_M,Knotenkoordinaten_M,
  Knotenkoordinaten_E,ElementeU,IndexMatrix_M,Split_Ind_M,IndexMatrix_E,
  Split_Ind_E,Koeffizienten,Normierung):

  #Ausrechnen der Knotenspannungen fuer die Mittelknoten
  IndexMatrix_M=np.split(IndexMatrix_M,Split_Ind_M)
  IndexMatrix_E=np.split(IndexMatrix_E,Split_Ind_E)
  l=Knotenlabels_M.shape[0]
  KnotenspannungenM=np.zeros((l,6))
  for i in xrange(0,l):
    BezugsknotenKoord=Knotenkoordinaten_M[i]
    Elemente_Patch=IndexMatrix_M[i]
    Knoten_Patches=np.unique(ElementeU[Elemente_Patch])

    #Knoten an denen Koeffizienten berechnet wurden
    Knoten_Patches=Knoten_Patches[Critical_Nodes[Knoten_Patches]]
    if (Knoten_Patches.shape[0]==0):
      Elemente_Patch=IndexMatrix_M[i]
      Knoten_Patches=np.unique(ElementeU[Elemente_Patch])
      Elemente_Patch=np.array((),int)
      for y in xrange(0,Knoten_Patches.shape[0]):
        Elemente_Patch=np.concatenate((Elemente_Patch,IndexMatrix_E[Knoten_Patches[y]]))
      Knoten_Patches=np.unique(ElementeU[Elemente_Patch])
      Knoten_Patches=Knoten_Patches[Critical_Nodes[Knoten_Patches]]

    Knoten_Patches_Koord=Knotenkoordinaten_E[Knoten_Patches]
    KnotenspannungenM[i,:]=get_Spannungsberechnung(BezugsknotenKoord,Knoten_Patches,
      Knoten_Patches_Koord,Koeffizienten,Normierung)
  return KnotenspannungenM
```

```
def get_Spannungsberechnung(BezugsknotenKoord,Knoten_Patches,Knoten_Patches_Koord,
  Koeffizienten,Normierung):

  Normierung=Normierung[Knoten_Patches,:]

  Knoten_Patches_Koord[:,0]=((Knoten_Patches_Koord[:,0]-
    BezugsknotenKoord[0])*-1)/Normierung[:,0]
  Knoten_Patches_Koord[:,1]=((Knoten_Patches_Koord[:,1]-
    BezugsknotenKoord[1])*-1)/Normierung[:,1]
  Knoten_Patches_Koord[:,2]=((Knoten_Patches_Koord[:,2]-
    BezugsknotenKoord[2])*-1)/Normierung[:,2]

  Abstand=np.sqrt(np.power(BezugsknotenKoord[0]-Knoten_Patches_Koord[:,0],2)+
  np.power(BezugsknotenKoord[1]-Knoten_Patches_Koord[:,1],2)+
  np.power(BezugsknotenKoord[2]-Knoten_Patches_Koord[:,2],2))
  Gewichtung=1./np.power(Abstand,3)

  P=KoeffMatrix_C2(Knoten_Patches_Koord)
  Knotenspannungen_Temp=np.zeros((Gewichtung.shape[0],6))
  for ii in xrange(0,Gewichtung.shape[0]):
    for j in xrange(0,6):
      Knotenspannungen_Temp[ii,j]=np.sum(P[ii,:]*
        Koeffizienten[j,Knoten_Patches[ii],:])*Gewichtung[ii]

  Knotenspannungen=np.sum(Knotenspannungen_Temp,axis=0)/np.sum(Gewichtung)
  return Knotenspannungen

def get_Spannungen_kombinieren(Knotenlabels_E,Knotenlabels_M,Knotenspannungen_E,
  Knotenspannungen_M):

  # Knotenspannungen zusammenfassen
  l1=Knotenlabels_E.shape[0]
  l2=Knotenlabels_M.shape[0]
  Knotenlabels_Ausgabe=np.zeros(l1+l2,int)
  Knotenspannungen_Ausgabe=np.zeros((l1+l2,6))

  Knotenlabels_Ausgabe[:l1]=Knotenlabels_E
  Knotenlabels_Ausgabe[l1:]=Knotenlabels_M
  Knotenspannungen_Ausgabe[:l1]=Knotenspannungen_E
  Knotenspannungen_Ausgabe[l1:]=Knotenspannungen_M

  ind=np.argsort(Knotenlabels_Ausgabe)
  Knotenlabels_Ausgabe=Knotenlabels_Ausgabe[ind]
  Knotenspannungen_Ausgabe=Knotenspannungen_Ausgabe[ind]
  return Knotenlabels_Ausgabe,Knotenspannungen_Ausgabe
```

9.1.5. Auswertefade ermitteln

```
import numpy as np
import os,sys
pathname = os.path.dirname(sys.argv[0])
Path=os.path.abspath(pathname)
sys.path.append(Path+'\\py_Module')
sys.path.append(Path+'\\c++_Module\\')
from Knotennummer import Knotennummer
from Facet_Auswertung import Facet_Auswertung

def get_Pfade(KnotenLabels_ges,KnotenKoord_ges,ElementLabels,Connectivity,
  Pfad_Oberflaeche,Zuschnitt):

  #Einlesen und sortieren der Oberflaechenknoten
  [Knoten_Label_Schale,Knoten_Koord_Schale]=Oberflaechenknoten_einlesen(Pfad_Oberflaeche)
  idx=Knoten_Label_Schale.argsort()
  Knoten_Label_Schale=Knoten_Label_Schale[idx]
  Knoten_Koord_Schale=Knoten_Koord_Schale[idx]

  #Filtern der Knotenlabels (Knoten die zu weit am Rand liegen werden nicht beruecksichtigt
  x1=np.min(Knoten_Koord_Schale[:,0])
  x2=np.max(Knoten_Koord_Schale[:,0])
  y1=np.min(Knoten_Koord_Schale[:,1])
  y2=np.max(Knoten_Koord_Schale[:,1])

  ind_x=np.logical_and(Knoten_Koord_Schale[:,0]>x1+Zuschnitt,
    Knoten_Koord_Schale[:,0]<x2-Zuschnitt)
  ind_y=np.logical_and(Knoten_Koord_Schale[:,1]>y1+Zuschnitt,
    Knoten_Koord_Schale[:,1]<y2-Zuschnitt)
  ind_ges=np.logical_and(ind_x,ind_y)

  Knoten_Label_Schale=Knoten_Label_Schale[ind_ges]
  Knoten_Koord_Schale=Knoten_Koord_Schale[ind_ges,:]

  print(Knoten_Label_Schale.shape)
  Knoten_Label_Schale_orig=np.array(Knoten_Label_Schale)

  # Eliminieren der Knoten die in der Connectivity nicht vorkommen
  [KnotenLabels,KnotenKoord]=
    KnotenLabels_u_Koord_in_Connectivity(KnotenLabels_ges,KnotenKoord_ges,Connectivity)
  del KnotenKoord_ges

  # Knotenlabels in Connectivity ersetzen
  for i in xrange(0,10):
    Connectivity[:,i]=Knotennummer(Connectivity[:,i],KnotenLabels)

  KnotenLabels_intern=np.arange(KnotenLabels.size)
  ElementLabels_intern=np.arange(ElementLabels.size)

  # Feststellen welche Elemente an einem Knoten haengen
  [Knoten_Elem_Conn,index,count]=KnotenElementConnectivity(Connectivity,ElementLabels_intern)

  # Ermittel der Indices der Oberflaechenknoten
  Knoten_Label_Schale=Knoten_Label_Schale_to_global(KnotenLabels,Knoten_Label_Schale)

  # Ermitteln der Pfade
  AnzahlDerKnoten=200
  Laenge=0.4
  Pfade=PfadeErmitteln(AnzahlDerKnoten,Laenge,Knoten_Koord_Schale)

  # Berechnen der Facets
  Facet_1=Facet(Connectivity[:,[0,1,3]],KnotenKoord)
  Facet_2=Facet(Connectivity[:,[1,2,3]],KnotenKoord)
  Facet_3=Facet(Connectivity[:,[2,0,3]],KnotenKoord)
  Facet_4=Facet(Connectivity[:,[0,2,1]],KnotenKoord)

  print(Knoten_Label_Schale.shape)
  [Pfad_Elements, Fehler]=Zugehoerige_Elemente_finden(
    Knoten_Label_Schale,Pfade,Facet_1,Facet_2,Facet_3,Facet_4,KnotenLabels_intern,
    ElementLabels_intern,Knoten_Elem_Conn,Connectivity,index,count)
  return Pfade,Pfad_Elements,Knoten_Label_Schale,Knoten_Label_Schale_orig
```

```
def Zugehoerige_Elemente_finden(Knoten_Label_Schale,Pfade,Facet_1,Facet_2,Facet_3,Facet_4,
  KnotenLabels_intern,ElementLabels_intern,Knoten_Elem_Conn,Connectivity,index,count):
  Anzahl_Knoten=Knoten_Label_Schale.shape[0]
  LaengePfad=Pfade.shape[1]

  Anzahl_Knoten_ges=KnotenLabels_intern.shape[0]
  Fehler=0
  Pfad_Elements=np.zeros([Anzahl_Knoten,LaengePfad],dtype='uint32')
  for i in xrange(0,Anzahl_Knoten):
    if i%5000==0: print(i)

    ElemLabels=Knoten_Elem_Conn[index[Knoten_Label_Schale[i]]:index[Knoten_Label_Schale[i]]+
      count[Knoten_Label_Schale[i]]]

    Facet_1_A=Facet_1[ElemLabels]
    Facet_2_A=Facet_2[ElemLabels]
    Facet_3_A=Facet_3[ElemLabels]
    Facet_4_A=Facet_4[ElemLabels]

    indicesLast=4294967294 #Zweiter Fehlerwert
    for j in xrange(1,LaengePfad): #als letzes geaendert
      P=Pfade[i,j]
      indices=Facet_Auswertung(Facet_1_A,Facet_2_A,Facet_3_A,Facet_4_A,P)

      # benachbarte Elemente hinzufuegen
      if (indices == 4294967295 and indicesLast!=4294967294):
        Nachbarknoten=Connectivity[indicesLast]
        size=count[Nachbarknoten]
        ElemLabels=np.zeros(size.sum(),dtype='int')
        s=0
        jj=0

        for jj in range(10):
          ElemLabels[s:s+size[jj]]=Knoten_Elem_Conn[index[Nachbarknoten[jj]]:
            index[Nachbarknoten[jj]]+count[Nachbarknoten[jj]]]
          s=s+size[jj]

        ElemLabels=np.unique(ElemLabels)
        Facet_1_A=Facet_1[ElemLabels]
        Facet_2_A=Facet_2[ElemLabels]
        Facet_3_A=Facet_3[ElemLabels]
        Facet_4_A=Facet_4[ElemLabels]

        indices=Facet_Auswertung(Facet_1_A,Facet_2_A,Facet_3_A,Facet_4_A,P)

      # Wenn in den Nachbarelemneten auch nichts gefunden wurde alle Elemente versuchen
      if indices == 4294967295:
        indices=Facet_Auswertung(Facet_1,Facet_2,Facet_3,Facet_4,P)
        if indices != 4294967295:
          ElemLabels=ElementLabels_intern[[indices]]
          Facet_1_A=Facet_1[[indices]]
          Facet_2_A=Facet_2[[indices]]
          Facet_3_A=Facet_3[[indices]]
          Facet_4_A=Facet_4[[indices]]
          indices=0

      # Wenn alles scheitert Fehlermeldung ausgeben und Fehlerwert speichern
      if indices == 4294967295:
        Pfad_Elements[i,j]=4294967295
        print('Nr. '+str(Fehler)+' in Pfad Nr. '+str(i)+ ' Pos Nr. '+str(j))
        Fehler=Fehler+1
      else:
        indices=ElemLabels[indices]
        indicesLast=indices
        Pfad_Elements[i,j]=indices

  #Auffuellen der ersten Pfadeintraege mit dem jeweils 2.Eintrag
  Pfad_Elements[:,0]=Pfad_Elements[:,1]

  return [Pfad_Elements, Fehler]
```

```
def PfadeErmitteln(KnotenAnzahl,Laenge,Knoten_Koord_Schale):
  Pfade=np.zeros((Knoten_Koord_Schale.shape[0],KnotenAnzahl,3),dtype='float')

  for i in range(Knoten_Koord_Schale.shape[0]):
    Pfade[i,:,0]=Knoten_Koord_Schale[i,0]
    Pfade[i,:,1]=Knoten_Koord_Schale[i,1]
    Pfade[i,:,2]=np.linspace(Knoten_Koord_Schale[i,2], Knoten_Koord_Schale[i,2]-
      Laenge, num=KnotenAnzahl, endpoint=True, retstep=False, dtype='float')

  return Pfade


def KnotenLabels_u_Koord_in_Connectivity(KnotenLabels_ges,KnotenKoord_ges,Connectivity):
  # Sortieren der Knotenlabels
  # Es koennen Knoten vorhanden sein die in der Elementconnectivity nicht vorkommen!!
  idx_Knotenlabels=KnotenLabels_ges.argsort()
  KnotenLabels_ges=KnotenLabels_ges[idx_Knotenlabels]
  KnotenKoord=KnotenKoord_ges[idx_Knotenlabels]

  # Eliminieren der Knoten die in der Connectivity nicht vorkommen
  KnotenLabels=np.unique(Connectivity)
  bool=np.in1d(KnotenLabels_ges,KnotenLabels,assume_unique=True, invert=False)
  KnotenKoord=KnotenKoord_ges[bool]
  return KnotenLabels,KnotenKoord


def Knoten_Label_Schale_to_global(KnotenLabels,Knoten_Label_Schale):
  bool=np.in1d(KnotenLabels,Knoten_Label_Schale, assume_unique=True, invert=False)
  ind=np.where(bool==True)
  return ind[0]


def KnotenElementConnectivity(Connectivity,ElementLabels_intern):
  #Die Elementconnectivity muss vorher umnummeriert werden
  #Bestimmen der Knoten/Element Connectivity
  KnotenElementConn=np.zeros((Connectivity.size,2),dtype='uint32')
  l=Connectivity.shape[0]
  b=Connectivity.shape[1]
  for i in range(b):
    KnotenElementConn[i*l:(i+1)*l,0]=Connectivity[:,i]
    KnotenElementConn[i*l:(i+1)*l,1]=ElementLabels_intern

  KnotenElementConn=KnotenElementConn[KnotenElementConn[:,0].argsort()]
  [unique,index,count]=np.unique(KnotenElementConn[:,0], return_index=True,
    return_inverse=False, return_counts=True)
  return KnotenElementConn[:,1],index,count


def Oberflaechenknoten_einlesen(File_Schale):
  # Einlesen der Schalengeometrie
  fobj = open(File_Schale, "r")
  File=fobj.read()
  fobj.close()
  FileINP = File.split("\n")

  # Einlesen der Knotenlabels und Knotenkoordinaten
  Knoten = np.array(LineNumbers(FileINP,"*NODE"))
  Knoten_Koord=np.array(Knoten[:,1:4],float)
  Knoten_Label=np.array(Knoten[:,0],int)

  # Nach Knotenlabels sortieren
  idx=Knoten_Label.argsort()
  Knoten_Label=Knoten_Label[idx]
  Knoten_Koord=Knoten_Koord[idx]
  del Knoten, idx

  return Knoten_Label,Knoten_Koord
```

```
def LineNumbers(File,String):
  i=0
  Start=0
  Bereich=[]
  for line in File:
    if Start != 0:
      Bereich.append(line.split(","))
      if "*" in line:
        break
    if String in line:
      Start = i+1
    i=i+1
  del Bereich[len(Bereich)-1]
  return Bereich


def Facet(EckknotenNummern,KnotenKoord):
  P0=KnotenKoord[EckknotenNummern[:,0]]
  P1=KnotenKoord[EckknotenNummern[:,1]]
  P2=KnotenKoord[EckknotenNummern[:,2]]

  P01=P1-P0
  P02=P2-P0

  Facets=np.zeros([EckknotenNummern.shape[0],4],float)
  Facets[:,0:3]=np.cross(P01,P02)
  Facets[:,3]=np.sum(Facets[:,0:3]*P0,axis=1)

  return Facets


import numpy as np
cimport numpy as np
cimport cython

@cython.boundscheck(False) # turn of bounds-checking for entire function
def Facet_Auswertung(np.ndarray[np.float64_t, ndim=2] Facet_1,np.ndarray[np.float64_t, ndim=2]
Facet_2,np.ndarray[np.float64_t, ndim=2] Facet_3,np.ndarray[np.float64_t, ndim=2]
Facet_4,np.ndarray[np.float64_t, ndim=1] P):
  cdef int i
  cdef unsigned int index
  cdef int FacetSize = Facet_1.shape[0]
  cdef double Toleranz=0
  index = 4294967295
  for i in range(FacetSize):
    if ((Facet_1[i,0]*P[0]+Facet_1[i,1]*P[1]+Facet_1[i,2]*P[2]-Facet_1[i,3] <Toleranz) &
      (Facet_2[i,0]*P[0]+Facet_2[i,1]*P[1]+Facet_2[i,2]*P[2]-Facet_2[i,3] <Toleranz)&
      (Facet_3[i,0]*P[0]+Facet_3[i,1]*P[1]+Facet_3[i,2]*P[2]-Facet_3[i,3] <Toleranz)&
      (Facet_4[i,0]*P[0]+Facet_4[i,1]*P[1]+Facet_4[i,2]*P[2]-Facet_4[i,3] <Toleranz)):
      index=<unsigned int> i
  return index
```

9.1.6. Zugehörige Patchknoten zu den Pfaden finden

```
import numpy as np

def get_SP_Nodes_Pfad(Critical_Nodes,Knotenkoordinaten_E,ElementeU,IndexMatrix_E,Split_Ind_E,
  Pfad_Elements,Pfade):

  # Berechnen der Knotenspannungen fuer die einzellnen Pfade
  IndexMatrix_E=np.split(IndexMatrix_E,Split_Ind_E)
  l=Pfad_Elements.shape[0]
  h=Pfad_Elements.shape[1]
  Bezugsknoten_Pfade=np.zeros((l,h,4),dtype=np.uint32)
  Bezugsknoten_Pfade[:,:,:]=4294967295
  Bezugsknoten_Pfade_Koord_rel=np.zeros((l,h,4,3),dtype=np.float32)
  Bezugsknoten_Pfade_Koord=np.zeros((l,h,4,3),dtype=np.float32)
  bool_Erweiterung=0

  # Erfassen moeglicher Problemstellen
  m_Probleme=np.zeros((l,1),dtype=np.bool)

  for i in xrange(0,l):
    if i%5000==0: print(i)
    for j in xrange(0,h):
      BezugsknotenKoord=Pfade[i,j,:]
      Elemente_Patch=Pfad_Elements[i,j]
      # Wenn kein Element definiert ist wird das aktuelle Element durch
      # tiefer liegende definiert
      if Elemente_Patch==4294967295:
        for ii in xrange(0,h):
          Elemente_Patch=Pfad_Elements[i,ii]
          if (Elemente_Patch!=4294967295):
            break

      Knoten_Patches=np.unique(ElementeU[Elemente_Patch])

      #Knoten an denen Koeffizienten berechnet wurden
      Knoten_Patches=Knoten_Patches[Critical_Nodes[Knoten_Patches]]
      #Wenn Knoten Fehlen werden die 4 am naehesten gelegenen ausgwaehlt
      if (Knoten_Patches.shape[0]<4): # Aenderung!!!
        bool_Erweiterung=1
        Knoten_Patches_Temp=np.array(Knoten_Patches) #Urspruengliche Patchknoten
        if (j!=199):
          #print('Problem in Pfad Nr. '+str(i)+ ' Pos Nr. '+str(j))
          m_Probleme[i]=1

        #alle Patchknoten des betreffenden Elements auswaehlen
        Knoten_Patches=np.unique(ElementeU[Elemente_Patch])
        Elemente_Patch=np.array((),int)
        for y in xrange(0,Knoten_Patches.shape[0]):
          #Nachbarelemente hinzufuegen
          Elemente_Patch=np.concatenate((Elemente_Patch,IndexMatrix_E[Knoten_Patches[y]]))
        Knoten_Patches=np.unique(ElementeU[Elemente_Patch]) #Patchknoten der Nachbarelemente
        Knoten_Patches=Knoten_Patches[Critical_Nodes[Knoten_Patches]]

      if (Knoten_Patches.shape[0]<1):
        print('Fehler in Pfad Nr. '+str(i)+ ' Pos Nr. '+str(j))
        return -1

      Knoten_Patches_Koord=Knotenkoordinaten_E[Knoten_Patches]
      # Wenn mehr als 4 Bezugsknoten gefunden werden, werden nur die
      # naehesten 4 beruecksichtigt

      if (Knoten_Patches.shape[0]>4):
        Abstand=np.sqrt(np.power(BezugsknotenKoord[0]-Knoten_Patches_Koord[:,0],2)+
          np.power(BezugsknotenKoord[1]-Knoten_Patches_Koord[:,1],2)+
          np.power(BezugsknotenKoord[2]-Knoten_Patches_Koord[:,2],2))
        ind=np.argsort(Abstand)
        Knoten_Patches=Knoten_Patches[ind[0:4]]
```

```
      if (bool_Erweiterung>0):
        size=Knoten_Patches_Temp.shape[0]
        Temp=np.zeros(4,int)
        Temp[:]=4294967295
        Temp[0:Knoten_Patches.shape[0]]=Knoten_Patches[:]
        Knoten_Patches=Temp
        jjj=0
        for jj in xrange(4-size,4):
          Knoten_Patches[jj]=Knoten_Patches_Temp[jjj]
          jjj=jjj+1
        bool_Erweiterung=0

      Bezugsknoten_Pfade[i,j,:]=Knoten_Patches[:]

  return [Bezugsknoten_Pfade,m_Probleme]
```

9.1.7. Spannungswerte ermitteln (Cython Wrapper)

```
import numpy as np
cimport numpy as np
import cython
from libc.stdint cimport uint32_t

DTYPE_Int = np.int
ctypedef np.int_t DTYPE_Int_t
DTYPE = np.float64
ctypedef np.float64_t DTYPE_t

#cdef extern from "stdint.h":
#ctypedef unsigned long long int

cdef extern from "get_Spannungen_Punkt_C.hpp":
  void get_Spannungen_Punkt_C(int *Koeff_size,double *Koeffizienten_in,
    int PfadAnzahl,int Knotenanzahl,double* Pfade,uint32_t *Bezugsknoten_Pfade_in,
    double *Knotenkoordinaten_E,double *Normierung,float *Knotenspannungen_Pfade_in)

@cython.boundscheck(False)
@cython.wraparound(False)
def get_Spannungen_Punkt(Koeffizienten_in,Pfade_in,Bezugsknoten_Pfade_in,
  Knotenkoordinaten_E_in,Normierung_in):

  # Check input arrays
  cdef np.ndarray[np.float64_t, ndim=3, mode="c"] Koeffizienten
  Koeffizienten = np.ascontiguousarray(Koeffizienten_in, np.float64)

  cdef np.ndarray[np.uint32_t, ndim=3, mode="c"] Bezugsknoten_Pfade
  Bezugsknoten_Pfade = np.ascontiguousarray(Bezugsknoten_Pfade_in, dtype=np.uint32)

  cdef np.ndarray[np.float64_t, ndim=2, mode="c"] Knotenkoordinaten_E
  Knotenkoordinaten_E = np.ascontiguousarray(Knotenkoordinaten_E_in, dtype=np.float64)

  cdef np.ndarray[np.float64_t, ndim=2, mode="c"] Normierung
  Normierung = np.ascontiguousarray(Normierung_in, dtype=np.float64)

  cdef np.ndarray[np.float64_t, ndim=3, mode="c"] Pfade
  Pfade = np.ascontiguousarray(Pfade_in, dtype=np.float64)

  cdef:
    int PfadAnzahl=Bezugsknoten_Pfade.shape[0]
    int Knotenanzahl=Bezugsknoten_Pfade.shape[1]
    int Koeff_size[3]

  Koeff_size[0]=Koeffizienten.shape[0]
  Koeff_size[1]=Koeffizienten.shape[1]
  Koeff_size[2]=Koeffizienten.shape[2]

  # Output Array erstellen
  cdef np.ndarray[np.float32_t, ndim=3] Knotenspannungen_Pfade
  Knotenspannungen_Pfade=np.ascontiguousarray(
    np.zeros((PfadAnzahl,Knotenanzahl,6), dtype=np.float32))

  get_Spannungen_Punkt_C(Koeff_size,<double*> Koeffizienten.data,PfadAnzahl,Knotenanzahl,
    <double*> Pfade.data,<uint32_t*> Bezugsknoten_Pfade.data, <double*>
Knotenkoordinaten_E.data,
    <double*> Normierung.data,<float*> Knotenspannungen_Pfade.data)

  return Knotenspannungen_Pfade
```

9.1.8. Spannungswerte ermitteln (C - Code)

```
#include "get_Spannungen_Punkt_C.hpp"
#include "stdint.h"
#include "math.h"
#include <iostream>
#include <omp.h>

inline double pow3(double i){
  return i*i*i;
  }

inline double pow2(double i){
  return i*i;
  }

inline void KoeffMatrix(double *Koord,int size,double *Matrix){
  #define Matrix(r, c) (Matrix[(r)*20 + (c)])
  #define Koord(r, c) (Koord[(r)*3 + (c)])
  for (int i=0;i<size;i++){
    Matrix(i,0)=1.;
    Matrix(i,1)=Koord(i,0);
    Matrix(i,2)=Koord(i,1);
    Matrix(i,3)=Koord(i,2);
    Matrix(i,4)=pow2(Koord(i,0));
    Matrix(i,5)=pow2(Koord(i,1));
    Matrix(i,6)=pow2(Koord(i,2));
    Matrix(i,7)=Koord(i,0)*Koord(i,1);
    Matrix(i,8)=Koord(i,1)*Koord(i,2);
    Matrix(i,9)=Koord(i,2)*Koord(i,0);
    Matrix(i,10)=pow3(Koord(i,0));
    Matrix(i,11)=pow3(Koord(i,1));
    Matrix(i,12)=pow3(Koord(i,2));
    Matrix(i,13)=pow2(Koord(i,0))*Koord(i,1);
    Matrix(i,14)=pow2(Koord(i,1))*Koord(i,2);
    Matrix(i,15)=pow2(Koord(i,2))*Koord(i,0);
    Matrix(i,16)=pow2(Koord(i,0))*Koord(i,2);
    Matrix(i,17)=pow2(Koord(i,1))*Koord(i,0);
    Matrix(i,18)=pow2(Koord(i,2))*Koord(i,1);
    Matrix(i,19)=Koord(i,0)*Koord(i,1)*Koord(i,2);
    }
  #undef Matrix
  #undef Koord
}


inline void getAbstand(double *Koord,int size,double *Abstand){
  for (int i=0;i<size;i++){
    Abstand[i]=sqrt(pow2(Koord[i*3+0])+pow2(Koord[i*3+1])+pow2(Koord[i*3+2]));
  }
}

void get_rel_norm_Koord(double *Koord,double *BezugsknotenKoord,double
*Knoten_Patches_Koord,double *Normierung_T,int bool_Normieren)
{
  if (bool_Normieren>0)
  {
    for (int i=0;i<4;i++)
    {
      Koord[i*3+0]=(BezugsknotenKoord[0]-Knoten_Patches_Koord[i*3+0])/Normierung_T[i*3+0];
      Koord[i*3+1]=(BezugsknotenKoord[1]-Knoten_Patches_Koord[i*3+1])/Normierung_T[i*3+1];
      Koord[i*3+2]=(BezugsknotenKoord[2]-Knoten_Patches_Koord[i*3+2])/Normierung_T[i*3+2];
    }
  }
  else{
    for (int i=0;i<4;i++)
    {
      Koord[i*3+0]=(BezugsknotenKoord[0]-Knoten_Patches_Koord[i*3+0]);
      Koord[i*3+1]=(BezugsknotenKoord[1]-Knoten_Patches_Koord[i*3+1]);
      Koord[i*3+2]=(BezugsknotenKoord[2]-Knoten_Patches_Koord[i*3+2]);
    }

  }
}
```

```
void get_Spannungen_Punkt_C(int *Koeff_size,double *Koeffizienten_in,
  int PfadAnzahl,int Knotenanzahl,double *Pfade,uint32_t *Bezugsknoten_Pfade_in,
  double *Knotenkoordinaten_E,double *Normierung,float *Knotenspannungen_Pfade_in){

  // Variablendeklarationen
  double Koeff_sum,Abstand[4],Gewichtung[4],P_in[4*20],Koord[4*3],Koord_rel[4*3]
  double Knotenspannungen_Temp[4*6],BezugsknotenKoord[3],Knoten_Patches_Koord[4*3]
  double Normierung_T[4*3];

  int Bezugsknoten_Anzahl=4;
  uint32_t Bezugsknoten[4];

  // Macrodeklarationen
  #define Koeffizienten(i,j,k) (Koeffizienten_in[(i)*Koeff_size[1]*
    Koeff_size[2] + (j)*Koeff_size[2] + (k)])
  #define Bezugsknoten_Pfade(i,j,k) (Bezugsknoten_Pfade_in[(i)*4*Knotenanzahl + (j)*4 + (k)])
  #define Knotenspannungen Pfade(i,j,k) (Knotenspannungen_Pfade_in[(i)*6*
    Knotenanzahl + (j)*6 + (k)])

  #define P(i,j) (P_in[(i)*20 + (j)])

  #pragma omp parallel for shared(Koeff_size,Koeffizienten_in,PfadAnzahl,Knotenanzahl,
    Bezugsknoten Pfade in,Knotenspannungen Pfade in,Pfade,Normierung,Knotenkoordinaten_E)
    private(Koeff sum,Abstand,Gewichtung,P in,Koord,Koord rel,
    Knotenspannungen Temp,Bezugsknoten_Anzahl,Bezugsknoten,BezugsknotenKoord,Normierung_T,
    Knoten_Patches_Koord)

  for (int i=0;i<PfadAnzahl;i++){
    for (int j=0;j<Knotenanzahl;j++){
      BezugsknotenKoord[0]=Pfade[i*Knotenanzahl*3+j*3+0];
      BezugsknotenKoord[1]=Pfade[i*Knotenanzahl*3+j*3+1];
      BezugsknotenKoord[2]=Pfade[i*Knotenanzahl*3+j*3+2];

      for (int ii=0;ii<4;ii++){
        Bezugsknoten[ii]=Bezugsknoten_Pfade(i,j,ii);

        Bezugsknoten_Anzahl=ii;
        if (Bezugsknoten[ii]==4294967295){
          break;
        }

        Knoten_Patches_Koord[ii*3+0]=Knotenkoordinaten_E[Bezugsknoten[ii]*3+0];
        Knoten_Patches_Koord[ii*3+1]=Knotenkoordinaten_E[Bezugsknoten[ii]*3+1];
        Knoten_Patches_Koord[ii*3+2]=Knotenkoordinaten_E[Bezugsknoten[ii]*3+2];

        Normierung_T[ii*3+0]=Normierung[Bezugsknoten[ii]*3+0];
        Normierung_T[ii*3+1]=Normierung[Bezugsknoten[ii]*3+1];
        Normierung_T[ii*3+2]=Normierung[Bezugsknoten[ii]*3+2];
      }

      get_rel_norm_Koord(Koord,BezugsknotenKoord,Knoten_Patches_Koord,Normierung_T,0);
      get_rel_norm_Koord(Koord_rel,BezugsknotenKoord,Knoten_Patches_Koord,Normierung_T,1);


      getAbstand(Koord,Bezugsknoten_Anzahl,Abstand);
      Abstand[0]=Abstand[0]+0.00001; //Division durch 0 verhindern
      Abstand[1]=Abstand[1]+0.00001;
      Abstand[2]=Abstand[2]+0.00001;
      Abstand[3]=Abstand[3]+0.00001;
      double sum_Gewichtung=0;
      for(int jj=0;jj<Bezugsknoten_Anzahl;jj++){
        Gewichtung[jj]=1./Abstand[jj];
        sum_Gewichtung=sum_Gewichtung+Gewichtung[jj];
      }

      KoeffMatrix(Koord_rel,Bezugsknoten_Anzahl, P_in);
      for (int ii=0;ii<Bezugsknoten_Anzahl;ii++){
        for (int jj=0;jj<6;jj++){
          Koeff_sum=0;
          for (int ki=0;ki<20;ki++){
            Koeff_sum=Koeff_sum+P(ii,ki)*Koeffizienten(jj,Bezugsknoten[ii],ki)*Gewichtung[ii];
          }
          Knotenspannungen_Temp[ii*6+jj]=Koeff_sum;
        }
      }
```

```
      double sum_Temp=0;
      for (int jj=0;jj<6;jj++){
        sum_Temp=0;
        for (int ii=0;ii<Bezugsknoten_Anzahl;ii++){
          sum_Temp=sum_Temp+Knotenspannungen_Temp[ii*6+jj];
        }
        Knotenspannungen_Pfade(i,j,jj)=(float) (sum_Temp/sum_Gewichtung);
      }

    }
  }
}
```

9.2. Berechnung des Schwingfestigkeitseinflusses

9.2.1. Hauptfunktion

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import os,sys
pathname = os.path.dirname(sys.argv[0])
Path=os.path.abspath(pathname)
sys.path.append(Path+'\\py_Module')
from Lifetime import Lifetime
import Tkinter as tk
import tkFileDialog as tkF
import ConfigParser
import ConfigSectionMap

def main():
  # Read Config File-------------------------------------------------------------------------
  Config = ConfigParser.ConfigParser()
  file=Config.read("Lebensdauerbewertung.ini")

  Nominalspannung =
    float(ConfigSectionMap(Config,'Stress_Simulation_Parameters')['nominalspannung'])
  deltaKth = float(ConfigSectionMap(Config,'Material_Parameters')['deltakth'])
  SaD = float(ConfigSectionMap(Config,'Material_Parameters')['sad'])
  Y = float(ConfigSectionMap(Config,'General_Parameters')['geometrieeinflussfaktor y'])
  Y_Rundprobe = float(ConfigSectionMap(Config,'General_Parameters')['y_rundprobe'])

  # Einlesen der Pfade
  root = tk.Tk()
  root.withdraw()

  Pfad_Eingangsdaten_dict =
    dict(defaultextension='.npz',filetypes=[('Zwischenergebnisse','*.npz')])
  Pfad_Eingangsdaten=tkF.askopenfilenames(**Pfad_Eingangsdaten_dict)
  Pfad_Eingangsdaten=Pfad_Eingangsdaten[0]

  SaveOptions = dict(initialdir=os.path.dirname(Pfad_Eingangsdaten))
  Pfad_Ausgabe=tkF.askdirectory(**SaveOptions)+'\\'

  root.deiconify()
  root.destroy()

  # Schwingfestigkeitseinfluss berechnen
  Lifetime(Pfad_Eingangsdaten,Pfad_Ausgabe,deltaKth,SaD,Y,Y_Rundprobe,Nominalspannung)

if __name__ == "__main__":
  main()
```

9.2.2. Schwingfestigkeitseinfluss für alle Pfade berechnen

```
import numpy as np
from scipy import interpolate
from scipy import integrate
import h5py
import os,sys
pathname = os.path.dirname(sys.argv[0])
Path=os.path.abspath(pathname)
sys.path.append(Path+'\\py_Module')
sys.path.append(Path+'\\c++_Module')
import get_Stress
import abq_py_extension as abq

def Lifetime(Pfad_Eingangsdaten,Pfad_Ausgabe,deltaKth,SaD,Y,Y_Probe,Nominalspannung):

  # Berechnung von ath [mm]
  ath=np.power((deltaKth/(2*Y*SaD)),2)*1/np.pi*1000
  print(ath)
  ath_Pore=np.power((deltaKth/(2*Y_Probe*SaD)),2)*1/np.pi*1000


  # Ausgabedatei erstellen
  f = h5py.File(Pfad_Ausgabe+'Ergebnisse_Kth_'+str(deltaKth)+
    '_SaD_'+str(SaD)+'_Abminderung.h5','w')


  # Spannungen einlesen
  Spannungen=np.load(Pfad_Eingangsdaten)
  Spannungen_Zug_0=Spannungen['Spannungen_Zug_0']
  Spannungen_Zug_90=Spannungen['Spannungen_Zug_90']
  Spannungen_Schub=Spannungen['Spannungen_Schub']
  shape=Spannungen_Zug_0.shape

  Spannungen_Zug_0=Spannungen_Zug_0.reshape(shape[0]*shape[1],shape[2])
  Spannungen_Zug_90=Spannungen_Zug_90.reshape(shape[0]*shape[1],shape[2])
  Spannungen_Schub=Spannungen_Schub.reshape(shape[0]*shape[1],shape[2])


  for i in xrange(0,180):
    print(str(i)+'_Grad')

    # Gruppe erstellen
    grp = f.create_group(str(i)+'_Grad')

    # Berechnen der winkelabhaengigen Spannungen
    Zug_0_Grad_T=get_Stress.get_rot_Ten_z_Deg(Spannungen_Zug_0,i)
    Zug_90_Grad_T=get_Stress.get_rot_Ten_z_Deg(Spannungen_Zug_90,i)
    Schub_T=get_Stress.get_rot_Ten_z_Deg(Spannungen_Schub,i)
    Ueberlagerter_Lastfall=
      abq.get_Spannungsueberlagerung(Zug_0_Grad_T,Zug_90_Grad_T,Schub_T,i)
    Spannungsverlaeufe=abq.get_abs_Max_Prin(Ueberlagerter_Lastfall)
    Spannungsverlaeufe=Spannungsverlaeufe.reshape(shape[0],shape[1])
    Spannungsverlaeufe=Spannungsverlaeufe[:,0:199]
    abs_max_principal=np.array(Spannungsverlaeufe[:,0],dtype=float)

    # Die Laenge des Spannungspfades ist mit 0.398mm vorgegeben
    Weg=np.arange(0.0,0.398,0.002)

    # Feststellen an welcher Stelle des Arrays das untere Ende erreicht ist
    ind = np.max(np.where(Weg < 2*ath))+1
    Spannungsverlaeufe_Auswertung=Spannungsverlaeufe[:,0:ind+1]
      Spannungsverlaeufe_Auswertung[:,ind]=(2*ath-Weg[ind-1])*
      ((Spannungsverlaeufe_Auswertung[:,ind]-Spannungsverlaeufe_Auswertung[:,ind-1])/
      (Weg[ind]-Weg[ind-1]))+Spannungsverlaeufe_Auswertung[:,ind-1]
    Weg=Weg[0:ind+1]

    # Aufintegrieren der Spannungsverlaeufe
    Sigma_M=(1/(2*ath))*np.trapz(Spannungsverlaeufe_Auswertung,Weg,axis=1)

    Grenzwert=Nominalspannung*0.5
    Sigma_M_2=np.copy(Sigma_M)
    Sigma_M_2[Sigma_M_2<Grenzwert]=Grenzwert
```

```
    SaD_K_Temp=(SaD/(Sigma_M_2/Nominalspannung))
    SaD_K=(Nominalspannung/Sigma_M_2)

    # Faktor 2, da nur halber Porenduchmesser verwendet wird
    d_aq=2*((np.power((deltaKth/(SaD_K_Temp*2*Y_Probe)),2)/np.pi)*1000000-ath_Pore*1000)

    # Ausgeben der Ergebnisse
    dset = grp.create_dataset('abs_max_principal', data=abs_max_principal)
    dset = grp.create_dataset('Sigma_M', data=Sigma_M)
    dset = grp.create_dataset('SaD_K', data=SaD_K)
    dset = grp.create_dataset('d_aq', data=d_aq)

f.close()
```

9.3. Berechnung der lokalen Minima des Abminderungsfaktors

9.3.1. Hauptfunktion

```
import numpy as np
import h5py
import os,sys
pathname = os.path.dirname(sys.argv[0])
Path=os.path.abspath(pathname)
sys.path.append(Path+'\\py_Module')
from get_Connectivity_Schale import*

import Tkinter as tk
import tkFileDialog as tkF
import ConfigParser
import ConfigSectionMap

def LokaleMinimaBestimmen():
  # Read Config File--------------------------------------------------------------------------
-----
  Config = ConfigParser.ConfigParser()
  file=Config.read("Statistik_Ausgabe.ini")

  Oberer_Schwellwertfaktor =
    float(ConfigSectionMap(Config,'Auswerteparameter')['oberer_schwellwertfaktor'])
  Unterer Schwellwertfaktor =
    float(ConfigSectionMap(Config,'Auswerteparameter')['unterer_schwellwertfaktor'])
  Oberflaeche_einlesen =
    bool(float(ConfigSectionMap(Config,'Allgemeine Parameter')['oberflaeche_einlesen']))
  Ueberlagerung_vorhanden =
    bool(float(ConfigSectionMap(Config,'Allgemeine Parameter')['ueberlagerung_vorhanden']))

  # Einlesen der Pfade------------------------------------------------------------------------
-----
  root = tk.Tk()
  root.withdraw()

  Ergebnisse dict = dict(defaultextension='.h5',filetypes=[('Ergebnisse','*.h5')])
  Pfad_Ergebnisse=tkF.askopenfilenames(**Ergebnisse_dict)
  Pfad_Ergebnisse=Pfad_Ergebnisse[0]

  Zwischenergebnisse_dict =
dict(defaultextension='.npz',filetypes=[('Zwischenergebnisse','*.npz')],
    initialdir=os.path.dirname(Pfad_Ergebnisse))
  Zwischenergebnisse_Pfad=tkF.askopenfilenames(**Zwischenergebnisse_dict)
  Zwischenergebnisse_Pfad=Zwischenergebnisse_Pfad[0]

  if Oberflaeche_einlesen:
    File Schale dict = dict(defaultextension='.inp',filetypes=[('Oberflaeche','*.inp')],
      initialdir=os.path.dirname(Pfad_Ergebnisse))
    File_Schale_Pfad=tkF.askopenfilenames(**File_Schale_dict)
    File_Schale_Pfad=File_Schale_Pfad[0]
    [Connectivity,Knoten_Label,Knoten_Koord]=get_Connectivity_Schale(File_Schale_Pfad)

  SaveOptions = dict(initialdir=os.path.dirname(Pfad_Ergebnisse),defaultextension='.h5',
    filetypes=[('Auswertung','*.h5')])
  Pfad_Auswertung=tkF.asksaveasfilename(**SaveOptions)

  root.deiconify()
  root.destroy()

  # Einlesen der Daten
  Inputvariablen=np.load(Zwischenergebnisse_Pfad)
  Knotenlabels_Ergebnis=Inputvariablen['Knoten_Label_Schale_orig']

  if not Oberflaeche_einlesen:
    Connectivity=Inputvariablen['Connectivity_Schale_all']
    Knoten_Label=Inputvariablen['Knoten_Label_Schale_all']
    Knoten_Koord=Inputvariablen['Knoten_Koord_Schale_all']
```

```
#-------------------------------------------------------------------------------
# Die Knotenlabels der Elemente entsprechen nach der Umnummerierung den Indizes der
# Spannungswerte im Array
# Es wird davon ausgegangen, dass die gleichen Knoten in den Knoteneintraegen und
# in den Elementdefinitionen vorhanden sind!
Knotenlabels_P, ElementeU = np.unique(Connectivity,return_inverse=True)
ElementeU=ElementeU.reshape(Connectivity.shape)

bool_Labels=np.in1d(Knotenlabels_P, Knotenlabels_Ergebnis)
Labels_mod=np.arange(0,Knotenlabels_P.shape[0],1)
Labels_mod=Labels_mod[bool_Labels]

# Nachbarknoten bestimmen (Die Nachbarschaftsdefinitionen enthalten die Indizes der
  Knotenspannungen im Array)
Nachbarn=np.concatenate((
ElementeU[:,[0,3]],ElementeU[:,[0,5]],
ElementeU[:,[1,3]],ElementeU[:,[1,4]],
ElementeU[:,[2,4]],ElementeU[:,[2,5]],
ElementeU[:,[3,0]],ElementeU[:,[3,1]],
ElementeU[:,[4,1]],ElementeU[:,[4,2]],
ElementeU[:,[5,0]],ElementeU[:,[5,1]],))

# Flaecheninhalt der auf die x,y- Ebene projezierten Dreiecke berechnen
A=Knoten_Koord[ElementeU[:,1],:]-Knoten_Koord[ElementeU[:,0],:]
B=Knoten_Koord[ElementeU[:,2],:]-Knoten_Koord[ElementeU[:,0],:]
A[:,2]=0
B[:,2]=0
El_Flaeche=np.cross(A,B)
El_Flaeche=np.linalg.norm(El_Flaeche,axis=1)/2
Gesamtflaeche=np.sum(El_Flaeche)

idx=np.argsort(Nachbarn[:,0])
Nachbarn=Nachbarn[idx]

# Jedem Knoten seine Nachbarknoten zuweisen
Nachbarn_final=[]
Nachbarn_final.append([Nachbarn[0][0]])
ii=0
for i in xrange(1,len(Nachbarn)):
  if Nachbarn[i][0]>Nachbarn[i-1][0]:
    Nachbarn_final.append([Nachbarn[i][0],Nachbarn[i][1]])
    ii=ii+1
  else:
    Nachbarn_final[ii].append(Nachbarn[i][1])

# doppelte Eintaege entfernen
for i in xrange(0,len(Nachbarn_final)):
  Nachbarn_final[i]=np.unique(Nachbarn_final[i])

# Vorbereiten der Auswertung
f = h5py.File(Pfad_Ergebnisse,'r')

glob_minimum=10e30
for Winkel in xrange(0,180):
  SaD_K= np.array(f[str(Winkel)+'_Grad/SaD_K'],dtype=float)
  minimum=np.min(SaD_K)
  minimum_Label=np.argmin(SaD_K)
  if (minimum<glob_minimum):
    glob_minimum=minimum
    glob_minimum_Winkel=Winkel
    glob_minimum_Label=minimum_Label

# Berechnen des Schwellwertes
Schwellwert_o=(1-glob_minimum)*Oberer_Schwellwertfaktor+glob_minimum
Schwellwert_u=(1-glob_minimum)*Unterer_Schwellwertfaktor+glob_minimum
print('Oberer_Schwellwert '+str(Schwellwert_o))
print('Unterer_Schwellwert '+str(Schwellwert_u))
```

```
  # Ausgabedatei erstellen
  f_Ausgabe = h5py.File(Pfad_Auswertung,'w')
  print(Pfad_Auswertung)
  grp = f_Ausgabe.create_group('Allgemeines')
  dset = grp.create_dataset('Dateiname_Input', data=os.path.basename(Pfad_Ergebnisse))
  dset = grp.create_dataset('Globales_Minimum', data=glob_minimum)
  dset = grp.create_dataset('Winkel_des_globalen_Minimums', data=glob_minimum_Winkel)
  dset = grp.create_dataset('Label_des_globalen_Minimums',
data=Knoten_Label[glob_minimum_Label])
  dset = grp.create_dataset('Koordinaten_des_globalen_Minimums',
    data=Knoten_Koord[glob_minimum_Label])
  dset = grp.create_dataset('Oberer_Schwellwertfaktor', data=Oberer_Schwellwertfaktor)
  dset = grp.create_dataset('Unterer_Schwellwertfaktor', data=Unterer_Schwellwertfaktor)
  dset = grp.create_dataset('Oberer_Schwellwert', data=Schwellwert_o)
  dset = grp.create_dataset('Unterer_Schwellwert', data=Schwellwert_u)
  dset = grp.create_dataset('Gesamtflaeche', data=Gesamtflaeche)

  grp = f_Ausgabe.create_group('Ergebnisse')
  for Winkel in xrange(0,180):
    print('Bearbeite Winkel '+str(Winkel) + ' Grad')
    grp_Winkel = grp.create_group(str(Winkel)+'_Grad')
    grp_Detailergebnisse = grp_Winkel.create_group('Detailergebnisse')
    grp_Detail_Knotenlabels = grp_Detailergebnisse.create_group('Knotenlabels')
    grp_Detail_Knotenkoordinaten = grp_Detailergebnisse.create_group('Knotenkoordinaten')

    SaD_K= np.array(f[str(Winkel)+'_Grad/SaD_K'],dtype=float)
    Spannung=np.zeros(Knotenlabels_P.shape[0],float)
    Spannung[:]=10e30
    Spannung[Labels_mod]=SaD_K
    Max_Prin=np.array(Spannung)

    # Suchen der lokalen Minima
    # globales Minima
    ii=0
    Vergleichsliste=[]
    Erweiterungsliste=[]
    Knotensets=[]
    Maxima=[]
    lokale_Maxima_Gruppen=[]

    while Spannung.min() < Schwellwert_u:
      Index_Maximum=Spannung.argmin()
      # Die Indizes ab wann ein neues lokales Minimum auftritt wird ausgegeben
      lokale_Maxima_Gruppen.append(len(Maxima))
      Maxima.append(Index_Maximum)
      Spannung[Index_Maximum]=10e+30
      Vergleichsliste.extend(Nachbarn_final[Index_Maximum])
      Knotensets.append([])
      Knotensets[ii].extend([Index_Maximum])
      jj=0 #Anzahl der Knoten abzaehlen
      while len(Vergleichsliste)!=0:
        for i in xrange(0,len(Vergleichsliste)):
          if Spannung[Vergleichsliste[i]]<Schwellwert_o:
            jj=jj+1
            Knotensets[ii].extend([Vergleichsliste[i]])
            Spannung[Vergleichsliste[i]]=10e+30
            Erweiterungsliste.extend(Nachbarn_final[Vergleichsliste[i]])
        Vergleichsliste=Erweiterungsliste
        Erweiterungsliste=[]
      ii=ii+1
      Vergleichsliste=[]

    dset = grp_Winkel.create_dataset('Knoten_Labels_Extremwert', data=Knoten_Label[Maxima])
    dset = grp_Winkel.create_dataset('Knoten_Koord_Extremwert', data=Knoten_Koord[Maxima])
    dset = grp_Winkel.create_dataset('Abminderung_Extremwert', data=Max_Prin[Maxima])

    Extremwerte_Flaeche=np.zeros((len(Maxima),1),dtype=float)
```

```
    # Ausgabe von Knotenlabels und Knotenkoordinaten der lokalen Minima
    for i in xrange(len(Knotensets)):
      grp_Detail_Knotenlabels.create_dataset(str(i), data=Knoten_Label[Knotensets[i]])
      grp_Detail_Knotenkoordinaten.create_dataset(str(i), data=Knoten_Koord[Knotensets[i]])
      Extremwerte_Flaeche[i]=get_Flaeche(ElementeU,El_Flaeche,Knotensets[i])

    dset = grp_Winkel.create_dataset('Flaeche_der_lokalen_Extremwerte',
data=Extremwerte_Flaeche)

  # Bearbeiten der Ueberlagerten Ergebnisse
  if Ueberlagerung_vorhanden:
    grp_Winkel = f_Ausgabe.create_group('Ueberlagerte_Ergebnisse')
    grp_Detailergebnisse = grp_Winkel.create_group('Detailergebnisse')
    grp_Detail_Knotenlabels = grp_Detailergebnisse.create_group('Knotenlabels')
    grp_Detail_Knotenkoordinaten = grp_Detailergebnisse.create_group('Knotenkoordinaten')

    SaD_K= np.array(f['Globale_Maxima/SaD_K'],dtype=float)
    Spannung=np.zeros(Knotenlabels_P.shape[0],float)
    Spannung[:]=10e30
    Spannung[Labels_mod]=SaD_K
    Max_Prin=np.array(Spannung)
    # Suchen der lokalen Minima
    # globales Minima
    ii=0
    Vergleichsliste=[]
    Erweiterungsliste=[]
    Knotensets=[]
    Maxima=[]
    lokale_Maxima_Gruppen=[]

    while Spannung.min() < Schwellwert_u:
      Index_Maximum=Spannung.argmin()
      # Die Indizes ab wann ein neues lokales Minimum auftritt wird ausgegeben
      lokale_Maxima_Gruppen.append(len(Maxima))
      Maxima.append(Index_Maximum)
      Spannung[Index_Maximum]=10e+30
      Vergleichsliste.extend(Nachbarn_final[Index_Maximum])
      Knotensets.append([])
      Knotensets[ii].extend([Index_Maximum])
      jj=0 #Anzahl der Knoten abzaehlen
      while len(Vergleichsliste)!=0:
        for i in xrange(0,len(Vergleichsliste)):
          if Spannung[Vergleichsliste[i]]<Schwellwert_o:
            jj=jj+1
            Knotensets[ii].extend([Vergleichsliste[i]])
            Spannung[Vergleichsliste[i]]=10e+30
            Erweiterungsliste.extend(Nachbarn_final[Vergleichsliste[i]])
        Vergleichsliste=Erweiterungsliste
        Erweiterungsliste=[]
      ii=ii+1
      Vergleichsliste=[]

    dset = grp_Winkel.create_dataset('Knoten_Labels_Extremwert', data=Knoten_Label[Maxima])
    dset = grp_Winkel.create_dataset('Knoten_Koord_Extremwert', data=Knoten_Koord[Maxima])
    dset = grp_Winkel.create_dataset('Abminderung_Extremwert', data=Max_Prin[Maxima])

    Extremwerte_Flaeche=np.zeros((len(Maxima),1),dtype=float)

    # Ausgabe von Knotenlabels und Knotenkoordinaten der lokalen Minima
    for i in xrange(len(Knotensets)):
      grp_Detail_Knotenlabels.create_dataset(str(i), data=Knoten_Label[Knotensets[i]])
      grp_Detail_Knotenkoordinaten.create_dataset(str(i), data=Knoten_Koord[Knotensets[i]])
      Extremwerte_Flaeche[i]=get_Flaeche(ElementeU,El_Flaeche,Knotensets[i])

    dset = grp_Winkel.create_dataset('Flaeche_der_lokalen_Extremwerte',
data=Extremwerte_Flaeche)

  f.close()
  f_Ausgabe.close()
```

```
def get_Flaeche(ElementeU,Element_Flaechen,Labels_intern):
  shape=ElementeU.shape
  ElementeU=ElementeU.reshape(shape[0]*shape[1])
  in_Array=np.in1d(ElementeU,Labels_intern)
  ElementeU=ElementeU.reshape((shape[0],shape[1]))

  in_Array=in_Array.reshape((shape[0],shape[1]))
  in_Array=np.sum(in_Array,axis=1,dtype=np.int)

  # Alle Knoten im Element vorhanden
  F_bool_alle_Knoten=np.where(in_Array>shape[1]-1)[0]
  F_bool_mindestens_1_Knoten=np.where(in_Array>0.5)[0]

  Flaeche_min=np.sum(Element_Flaechen[F_bool_alle_Knoten])
  Flaeche_max=np.sum(Element_Flaechen[F_bool_mindestens_1_Knoten])

  return (Flaeche_min+Flaeche_max)/2


def LineNumbers(File,String):
  i=0
  Start=0
  Bereich=[]
  for line in File:
    if Start != 0:
      Bereich.append(line.split(","))
      if "*" in line:
        break
    if String in line:
      Start = i+1
    i=i+1
  del Bereich[len(Bereich)-1]
  return Bereich
################################################################################

if __name__ == '__main__':
  LokaleMinimaBestimmen()
```

9.4. Ausgabe der Ergebnisse

9.4.1. Ergebnisse in Excel darstellen

```
import xlsxwriter
import numpy as np
import h5py
import Tkinter as tk
import tkFileDialog as tkF
import os,sys

def main():
  # Definieren der Pfade
  root = tk.Tk()
  root.withdraw()

  Auswertung_dict = dict(defaultextension='.h5',filetypes=[('Auswertung','*.h5')])
  Pfad_Auswertung=tkF.askopenfilenames(**Auswertung_dict)
  Pfad_Auswertung=Pfad_Auswertung[0]

  SaveOptions = dict(initialdir=os.path.dirname(Pfad_Auswertung),
  defaultextension='.xlsx',filetypes=[('Excel-Auswertung','*.xlsx')])
  Pfad_Auswertung_Excel=tkF.asksaveasfilename(**SaveOptions)

  root.deiconify()
  root.destroy()

  # HDF5-Auswertefile Oeffnen
  f = h5py.File(Pfad_Auswertung,'r')
  grp_Ergebnisse=f['Ergebnisse']

  # Workebook erstellen
  workbook = xlsxwriter.Workbook(Pfad_Auswertung_Excel, {'strings_to_numbers': True})
  bold = workbook.add_format({'bold': True,'align': 'center'})
  Nachkomma_3 = workbook.add_format({'num_format': '#,###0.000','align': 'center'})
  italic = workbook.add_format({'italic': True})
  center = workbook.add_format({'align': 'center'})

  for Winkel in xrange(0,180):
    # Daten einlesen
    grp_Winkel_Ergebnisse=grp_Ergebnisse[str(Winkel)+'_Grad']

    KnotenLabels=np.array(grp_Winkel_Ergebnisse['Knoten_Labels_Extremwert'])
    Knotenkoodinaten=np.array(grp_Winkel_Ergebnisse['Knoten_Koord_Extremwert'])
    Abminderungsfaktor=np.array(grp_Winkel_Ergebnisse['Abminderung_Extremwert'])
    Flaeche=np.array(grp_Winkel_Ergebnisse['Flaeche_der_lokalen_Extremwerte'])

    # Arrays fur das Zusammenfassen erstellen
    if (Winkel <1):
      KnotenLabels_all=KnotenLabels
      Knotenkoodinaten_all=Knotenkoodinaten
      Abminderungsfaktor_all=Abminderungsfaktor
      Flaeche_all=Flaeche
    else:
      KnotenLabels all=np.hstack((KnotenLabels all,KnotenLabels))
      Knotenkoodinaten_all=np.vstack((Knotenkoodinaten_all,Knotenkoodinaten))
      Abminderungsfaktor_all=np.hstack((Abminderungsfaktor_all,Abminderungsfaktor))
      Flaeche_all=np.vstack((Flaeche_all,Flaeche))

    # Worksheet schreiben
    Write_Worksheet(workbook,str(Winkel)+'_Grad',KnotenLabels,
      Knotenkoodinaten,Flaeche,Abminderungsfaktor)

  # Gesamtdaten schreiben

Write_Worksheet(workbook,'Alle_Winkel',KnotenLabels_all,Knotenkoodinaten_all,Flaeche_all,Abmin
derungsfaktor_all)
```

```
  # Einlesen und schreiben der Gesamtueberlagerung
  grp_Winkel=f['Ueberlagerte_Ergebnisse']
  KnotenLabels=np.array(grp_Winkel['Knoten_Labels_Extremwert'])
  Knotenkoodinaten=np.array(grp_Winkel['Knoten_Koord_Extremwert'])
  Abminderungsfaktor=np.array(grp_Winkel['Abminderung_Extremwert'])
  Flaeche=np.array(grp_Winkel['Flaeche_der_lokalen_Extremwerte'])
  Write_Worksheet(workbook,'Globale_Ueberlagerung',KnotenLabels,Knotenkoodinaten,
    Flaeche,Abminderungsfaktor)

  # Zusammenfassung schreiben
  grp = f['Allgemeines']
  Dateiname_Input = grp['Dateiname_Input']
  Globales_Minimum = grp['Globales_Minimum']
  Winkel_des_globalen_Minimums = grp['Winkel_des_globalen_Minimums']
  Label_des_globalen_Minimums = grp['Label_des_globalen_Minimums']
  Koordinaten_des_globalen_Minimums = grp['Koordinaten_des_globalen_Minimums']
  Oberer_Schwellwertfaktor = grp['Oberer_Schwellwertfaktor']
  Unterer_Schwellwertfaktor = grp['Unterer_Schwellwertfaktor']
  Oberer_Schwellwert = grp['Oberer_Schwellwert']
  Unterer_Schwellwert = grp['Unterer_Schwellwert']
  Gesamtflaeche = grp['Gesamtflaeche']

  worksheet = workbook.add_worksheet('Zusammenfassung')
  worksheet.set_column(0, 0, 30)
  worksheet.set_column(1, 1, 45,center)
  worksheet.write(0, 0, 'Zusammenfassung',bold)
  worksheet.write(1, 0, '')
  worksheet.write(2, 0, 'Dateiname Input',italic)
  worksheet.write(3, 0, 'Oberer Schwellwertfaktor',italic)
  worksheet.write(4, 0, 'Unterer Schwellwertfaktor',italic)
  worksheet.write(5, 0, 'Oberer Schwellwert',italic)
  worksheet.write(6, 0, 'Unterer Schwellwert',italic)
  worksheet.write(7, 0, '')
  worksheet.write(8, 0, 'Minimaler Abminderungsfaktor',italic)
  worksheet.write(9, 0, 'bei Winkel',italic)
  worksheet.write(10, 0, 'Knotenlabel',italic)
  worksheet.write(11, 0, 'Koordinate X',italic)
  worksheet.write(12, 0, 'Koordinate Y',italic)
  worksheet.write(13, 0, 'Koordinate Z',italic)

  worksheet.write(2, 1, str(Dateiname_Input.value),Nachkomma_3)
  worksheet.write(3, 1, Oberer_Schwellwertfaktor.value,Nachkomma_3)
  worksheet.write(4, 1, Unterer_Schwellwertfaktor.value,Nachkomma_3)
  worksheet.write(5, 1, Oberer_Schwellwert.value,Nachkomma_3)
  worksheet.write(6, 1, Unterer_Schwellwert.value,Nachkomma_3)
  worksheet.write(8, 1, Globales_Minimum.value,Nachkomma_3)
  worksheet.write(9, 1, Winkel_des_globalen_Minimums.value,center)
  worksheet.write(10, 1, Label_des_globalen_Minimums.value,center)
  worksheet.write(11, 1, Koordinaten_des_globalen_Minimums[0],Nachkomma_3)
  worksheet.write(12, 1, Koordinaten_des_globalen_Minimums[1],Nachkomma_3)
  worksheet.write(13, 1, Koordinaten_des_globalen_Minimums[2],Nachkomma_3)
  workbook.close()

def Write_Worksheet(workbook,Name,KnotenLabels,Knotenkoodinaten,Flaeche,Abminderungsfaktor):
  # Worksheet hinzufuegen und Formatieren
  bold = workbook.add_format({'bold': True,'align': 'center'})
  Nachkomma_3 = workbook.add_format({'num_format': '#,###0.000','align': 'center'})
  center = workbook.add_format({'align': 'center'})

  worksheet = workbook.add_worksheet(Name)
  worksheet.set_column(0, 0, 12,center)
  worksheet.set_column(1, 1, 24.5,center)
  worksheet.set_column(2, 2, 24.5,center)
  worksheet.set_column(3, 3, 24.5,center)
  worksheet.set_column(4, 4, 20.0,center)
  worksheet.set_column(5, 5, 30,center)

  worksheet.write(0, 0, 'Knotenlabel',bold)
  worksheet.write(0, 1, 'Knotenkoordinate X [mm]',bold)
  worksheet.write(0, 2, 'Knotenkoordinate Y [mm]',bold)
  worksheet.write(0, 3, 'Knotenkoordinate Z [mm]',bold)
  worksheet.write(0, 4, 'Flaeche [mm^2]',bold)
  worksheet.write(0, 5, 'minimaler Abminderungsfaktor',bold)
```

```
  # Werte schreiben
  for row in xrange(0,KnotenLabels.shape[0]):
    worksheet.write(row+1, 0, KnotenLabels[row],center)
    worksheet.write(row+1, 1, Knotenkoodinaten[row,0],Nachkomma_3)
    worksheet.write(row+1, 2, Knotenkoodinaten[row,1],Nachkomma_3)
    worksheet.write(row+1, 3, Knotenkoodinaten[row,2],Nachkomma_3)
    worksheet.write(row+1, 4, Flaeche[row],center)
    worksheet.write(row+1, 5, Abminderungsfaktor[row],Nachkomma_3)

if __name__ == "__main__":
    main()
```

9.4.2. Ausgewählte Ergebnisse in Abaqus-ODB schreiben

```
from odbAccess import *
import numpy as np
import h5py
import os,sys
pathname = os.path.dirname(sys.argv[0])
Path=os.path.abspath(pathname)
sys.path.append(Path+'\\py_Module')
sys.path.append(Path+'\\abq_py_Module')
from Ergebnisse_ODB_schreiben import Ergebnisse_ODB_schreiben
import Tkinter as tk
import tkFileDialog as tkF

def main():
  # Ausgabe der Ergebnisse
  # Einlesen der Pfade-------------------------------------------------------------------------
  root = tk.Tk()
  root.withdraw()

  Abaqus_ODB_dict = dict(defaultextension='.odb',filetypes=[('Oberflaeche','*.odb')])
  Abaqus_ODB=str(tkF.askopenfilenames(**Abaqus_ODB_dict))

  Zwischenergebnisse_dict =
dict(defaultextension='.npz',filetypes=[('Zwischenergebnisse','*.npz')])
  Zwischenergebnisse=str(tkF.askopenfilenames(**Zwischenergebnisse_dict))

  Ergebnisse_h5_Pfad_dict = dict(defaultextension='.h5',filetypes=[('Ergebnisse','*.h5')])
  Ergebnisse_h5_Pfad=str(tkF.askopenfilenames(**Ergebnisse_h5_Pfad_dict))

  StepName='Kth_2,4_SaD_101'
  Input=np.load(Zwischenergebnisse)
  Knotenlabels_S=np.array(Input['Knoten_Label_Schale_orig'],int)

  Ergebnisse_ODB_schreiben(Abaqus_ODB,Ergebnisse_h5_Pfad,StepName,Knotenlabels_S)

if __name__ == "__main__":
    main()
```