Diploma Thesis

Implementation of a rectangular flat shell element in slangTNG for FEM-based pushover analysis of selected building models

submitted in satisfaction of the requirements for the degree of
Diplom-Ingenieur
of the TU Wien, Faculty of Civil Engineering

Diplomarbeit

Implementierung eines rechteckigen, ebenen Schalenelements in slangTNG zur FEM-basierten Pushover-Analyse ausgewählter Gebäudemodelle

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom-Ingenieurs
eingereicht an der Technischen Universität Wien, Fakultät für Bauingenieurwesen

von

Matthias Grobauer, BSc

Matr.Nr.: 01125784

unter der Anleitung von

Univ.Prof. Dipl.-Ing. Dr.techn. **Christian Bucher**

E208 Institut für Hochbau, Baudynamik und Gebäudetechnik
Forschungsbereich für Strukturdynamik und Risikobewertung von Tragwerken
Technische Universität Wien
Karlsplatz 13/208-01 , 1040 Wien, Österreich

Wien, im August 2020 _____

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Ich habe zur Kenntnis genommen, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

D I P L O M A R B E I T

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Ich erkläre weiters an Eides statt, dass ich meine Diplomarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur genannt habe.

11.08.2020
Datum

Unterschrift

Kurzfassung

In dieser Arbeit werden ausgewählte Gebäudemodelle mit Hilfe der Pushover-Analyse untersucht, um anschließend Aussagen bezüglich ihres Verhaltens unter einer Erdbebeneinwirkung treffen zu können.

Anfangs wird ein kurzer Überblick über die Berechnungsverfahren nach der europäischen Erdbebennorm Eurocode 8[14] und das statische Tragsystem von Wiener Gründerzeithäusern, welche im speziellen untersucht werden sollen, gegeben. Dazu sollen ein regelmäßiges und ein unregelmäßiges L-förmiges Gebäudemodell untersucht werden, um insbesondere den Einfluss von Rotationseigenformen auf die Tragsicherheit von Bauwerken zu ermitteln, da diese oftmals im Rahmen von Pushover-Analysen vernachlässigt werden.

Um die Berechnung zu bewerkstelligen, wird auf die an der TU Wien entwickelte Skriptsprache slangTNG[17] zurückgegriffen. Diese soll um das rechteckige Finite Element **RQuad** erweitert werden, um die Wandscheiben der Gebäudemodelle zu modellieren. Bei diesem Element handelt es sich um ein rechteckiges, ebenes Schalenelement, dessen Knoten jeweils die vollen sechs Freiheitsgrade zur Verfügung stehen.

Nach der Implementierung des Elements soll das neue Materialmodell **ElasticFailure** definiert werden, welches sprödes Materialversagen abzubilden vermag. Dadurch wird sichergestellt, dass die im Zuge der Pushover-Analyse monoton gesteigerten Horizontallasten ab einem gewissen Punkt der Berechnung zur Verminderung der Steifigkeit und folglich zum Ausfall einzelner Elemente führen, wodurch es zu einem Abflachen der Kurve der Kraft-Verformunsbeziehungen kommt. Erst dadurch wird eine Pushover-Analyse möglich.

Anschließend werden theoretische Grundlagen erläutert, die zum Verständis der abschließenden Anwendungsbeispiele erforderlich sind und die Vorgehensweise bei der Berechnung dieser Beispiele beschrieben. Schließlich können die zuvor neu in slangTNG[17] implementierten Funktionalitäten eingesetzt werden, um die Untersuchungen der beiden Gebäudemodelle vorzunehmen.

Zusammenfassend kann festgestellt werden, dass für ein unregelmäßiges Gebäudemodell auch eine Rotationseigenform maßgebend für die Bemessung unter Erdbebeneinwirkung werden kann. Auf Grund dieser Tatsache wird empfohlen, bei Vernachlässigung höherer Eigenformen bei der Pushover-Analyse ausreichende Tragreserven vorzusehen. Empfehlenswert ist jedoch die Berücksichtigung aller maßgebenden Eigenformen, um Abschätzungen eventuell erforderlicher Tragreserven zu vermeiden.

abstract

In this thesis, selected building models are examined with the help of pushover analysis in order to be able to make statements about their behaviour under the influence of an earthquake.

In the beginning a short overview of the calculation methods according to the European earthquake standard Eurocode 8[14] and the static load bearing system of Viennese masonry buildings, which are to be investigated in particular, is given. For this purpose, a regular and an irregular L-shaped building model will be investigated, in particular to determine the influence of rotational eigenmodes on the structural safety of buildings, as these are often neglected in pushover analyses.

To perform the calculation, the script language slangTNG[17] developed at the Vienna University of Technology is used. The rectangular finite element **RQuad** will be added to this language to model the walls of the building models. This element is a rectangular, flat shell element whose nodes each have the full six degrees of freedom available.

After the implementation of the element, the new material model **ElasticFailure** is to be defined, which is able to model brittle material failure. This ensures that the monotonically increased horizontal loads in the course of the pushover analysis lead to a reduction in stiffness from a certain point of the calculation on and consequently to the failure of individual elements, which in turn leads to a flattening of the force-deformation curve.

Subsequently, theoretical principles are explained which are necessary to understand the final application examples and the procedure for the calculation of these examples is described. Finally, the functionalities newly implemented in slangTNG[17] can be used to perform the investigations of the two building models.

In summary, it can be stated that for an irregular building model, a rotational eigenmode can also become decisive for the design under earthquake action. Due to this fact, it is recommended to provide sufficient load reserves when neglecting higher eigenmodes in the pushover analysis. However, it is recommended to consider all relevant eigenmodes in order to avoid estimates of possibly required load reserves.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit der Erdbebensicherheit von Gründerzeitbauten. Dabei handelt es sich um Gebäude aus Mauerwerk mit Holzdecken, welche gegen Ende des 19. und Anfang des 20. Jahrhunderts in Wien errichtet worden sind. In den meisten kommerziellen Finite Elemente Programmen, welche sich speziell an Bauingenieure und Bauingenieurinnen richten, gibt es keine zufriedenstellende Lösung, um einen Nachweis der Erdbebensicherheit nach den Europäischen Normen zu erbringen. Meist sind lediglich die linear elastischen Nachweisformate der Antwortspektrummethoden implementiert, welche sich sehr gut zur Bemessung moderner Stahlbetonbauten eignen. Bei Gründerzeithäusern mit ihren Wandpfeilern aus Mauerwerk erweist sich der Nachweis der in der Norm geforderten Tragsicherheit meist als problematisch. Die Norm schlägt hier eine detailliertere Methode vor, welche Nichtlinearitäten berücksichtigt und somit weitere Tragreserven, welche bei einer linear elastischen Berechnung nicht genützt werden können, zu aktivieren vermag. Die sogenannte nichtlineare statische Pushover-Analyse ist somit in jüngerer Vergangenheit in den Fokus vieler Bauingenieure und Bauingenieurinnen gerückt, die sich mit Gründerzeitbauten und insbesondere dem beim Ausbau solcher Gebäude erforderlichen Nachweis der Erdbebensicherheit beschäftigen.

1.1 Zielsetzung und Forschungsfrage

In dieser Arbeit wird gezeigt, wie eine mögliche Anwendung der Pushover-Analyse auf Mauerwerksbauten sinnvoll möglich ist. Dazu werden verschiedene Gebäudemodelle mit der Pushover-Analyse untersucht und anschließend die Berechnungsergebnisse bewertet. Dabei soll auch insbesondere analysiert werden, wie sich der Einfluss einer ungünstigen Grundrissform des Gebäudes hinsichtlich Torsionsbeanspruchungen auf die Ergebnisse auswirkt.

Da, wie bereits anfangs angesprochen, keine vollumfassenden Lösungen zur Anwendung der Pushover-Analyse in bereits bestehender Baustatik-Software existiert, wurde hier ein eigener Ansatz verfolgt, der in den nachfolgenden Kapiteln umrissen wird.

1.2 Gliederung der Arbeit

Um bei der Berechnung einen direkten Zugang zu wesentlichen Parametern zu haben, stellt es sich als sinnvoll heraus, diese mit Hilfe der von Herrn Prof. Christian Bucher an der TU Wien entwickelten Scriptsprache slangTNG[17] durchzuführen. Diese beinhaltet eine Sammlung von Funktionen und Klassen, welche hauptsächlich für Strukturanalysen Anwendung finden. slangTNG basiert im Kern auf der Programmiersprache C++, welche automatisiert in die Scriptsprache Lua übersetzt wird. Sie ist quelloffen, steht unter BSD Lizenz und kann daher einfach für eigene Zwecke erweitert sowie angepasst werden. Für diese Arbeit wird das bestehende Modul **fem**, das eine Klasse zur Erstellung eines Strukturmodells mitsamt verschiedener Finite Elemente Typen und Materialdefinitionen enthält, um ein weiteres Finites Element und eine zusätzliche Materialdefinition erweitert.

Diese in den Kapiteln 2 und 3 beschriebene Erweiterung von slangTNG stellt den ersten großen Abschnitt dieser Masterarbeit dar. In Kapitel 2 erfolgt die Präsentation des neu implementierten Finiten Elements **RQuad**, dessen Name sich von der vorgegebenen rechteckigen Form ableitet. Dieses Element stellt ein vollwertiges ebenes Schalenelement dar, hat also eine Steifigkeit sowohl in Membrantragrichtung als auch in Plattenwirkrichtung.[3] Anschließend werden in Kapitel 3 numerische Beispiele berechnet, um die Genauigkeit des Elements zu überprüfen. Im daran anschließenden Kapitel 4 wird das dem Element zugrundegelegte Materialmodell näher vorgestellt. Ziel ist es, ein Modell zu implementieren, das elastisches sprödes Versagen abbilden kann, um später den Ausfall von Elementen bei Überbeanspruchung simulieren zu können. Dieses Materialmodell wird im Folgenden **ElasticFailure** bezeichnet, da es keine plastischen Verformungen zulässt, sondern es nach der Überschreitung von Grenzspannungen zu Ausfällen einzelner Integrationspunkte oder darauffolgend ganzer Elemente kommt. Auf dieser Basis kann in einem weiteren Schritt die Pushover-Analyse durchgeführt werden, da es mit dem Herabsetzen der Steifigkeit von Elementen beziehungsweise deren gänzlichem Ausfall zu einer nichtlinearen Beziehung der Kräfte und der Verformungen kommt, was essentiell für die Anwendung der Pushover-Analyse ist.

Im zweiten Hauptteil dieser Arbeit erfolgt nach einer kurzen theoretischen Einführung in die Erdbebennormung und im speziellen die Pushover-Analyse die Anwendung des im ersten Teil implementierten Elements RQuad unter Nutzung des Materialmodells ElasticFailure auf ausgewählte Gebäudemodelle. Es werden also verschiedene Strukturmodelle untersucht, beginnend mit einer einzelnen Wandscheibe bis zu realitätsnäheren Modellen vereinfachter Gebäude. Die Pushover-Analyse wird sowohl kraftgesteuert als auch verschiebungsgesteuert durchgeführt, um das zuvor definierte Element auf seine Tauglichkeit für derartige Berechnungen zu prüfen. Anschließend werden die Ergebnisse der verschiedenen analysierten Gebäudemodelle diskutiert und zusammenfassende Schlussfolgerungen gezogen. Zum Abschluss erfolgt ein Ausblick auf weitere in diesem Zusammenhang sinnvoll erscheinende offene Themen und Fragestellungen, deren Behandlung auch für die zukünftig vorgesehene normative Anwendung der Pushover-Analyse interessant erscheinen.

1.3 Erdbebennormung gemäß Eurocodes

Der Nachweis der Tragsicherheit von Gebäuden gegen Erdbebeneinwirkungen ist gemäß den Europäischen Baunormen (Eurocodes) mit Hilfe verschiedener mehr oder weniger exakter Nachweismethoden möglich. Das Spektrum reicht von stark vereinfachten Methoden, die gewisse Anforderungen an die Geometrie des Grundrisses und Aufrisses des Gebäudes bezüglich ihrer Regelmäßigkeit und Bauweise stellen, bis zu hochdetaillierten Nachweisformaten, die eine tatsächliche Simulation des Erdbebens über eine Zeitspanne umfassen. Die folgende Tabelle 1.1 soll einen kurzen Überblick dieser Methoden nach Eurocode 8 (folgend auch EC8) Auslegung von Bauwerken gegen Erdbeben[14, 15] geben.

Nachfolgend werden die Methoden kurz beschrieben, um ein grundlegendes Verständnis zu vemitteln. Im Rahmen dieser Arbeit wird es zur Anwendung der Methode der Pushover-Analyse kommen, welche dann sehr detailliert in Kapitel 4 beschrieben wird. Die Komplexität und der Berechnungsaufwand der Methoden in der Tabelle 1.1 nimmt von oben nach unten zu.

Vereinfachtes Antwortspektrumverfahren

Das vereinfachte Antwortspektrumverfahren setzt eine erste Eigenform des Gebäudes analog einer Kragstütze voraus. Deshalb gibt es strenge einzuhaltende Regeln bezüglich der Gebäudeform im Grundriss als auch im Aufriss, die eine Regelmäßigkeit fordern und diese Annahme somit sicherstellen sollen. Sind diese Bedingungen eingehalten, darf die auf das Bauwerk einwirkende

Tab. 1.1: Nachweisformate nach EC8

Methode	Linearität	Berechnung	Modellgenauigkeit
Vereinfachtes Antwortspektrumverfahren	linear	statisch	stark vereinfacht, eben, nur erste Eigenform berücksichtigt
Multimodales Antwortspektrumverfahren	linear	statisch/dynamisch	detaillierter, dreidimensional, maßgebende Eigenformen berücksichtigt
Nichtlineare statische Pushover-Analyse	nichtlinear	statisch	dreidimensional, Plastisches Materialverhalten oder Versagen berücksichtigt
Nichtlineare Zeitverlaufsberechnung	nichtlinear	dynamisch	dreidimensional, Plastisches Materialverhalten oder Versagen berücksichtigt

Gesamterdbebenkraft stark vereinfacht in der Form eines Dreiecks über die Geschoße aufgeteilt werden, wobei man je Geschoß eine Einzellast auf Deckenebene angreifen lässt. Die so ermittelten Schnittgrößen werden anschließend gemäß der Steifigkeiten der Wandscheiben auf diese aufgeteilt. Schließlich kann die Untersuchung für zwei ebene Modelle jeweils in eine der beiden Symmetrieachsen des Bauwerks erfolgen.

Multimodales Antwortspektrumverfahren

Das multimodale Antwortspektrumverfahren ist momentan die in der Norm vorgeschlagene Referenzmethode. Im Unterschied zum vereinfachten Antwortspektrumverfahren werden alle maßgebenden Eigenformen der Struktur berücksichtigt. So können auch die Gebäudegrundrisse, die zu Rotationseigenformen (Torsion) neigen, berechnet und nachgewiesen werden. Die Berechnung erfolgt wieder linear, wobei die Kräfte wie auch beim vereinfachten Verfahren mit einem Verhaltensbeiwert, der die nichtlineare Antwort des Materials abbilden soll, abgemindert werden dürfen. Dieser Beiwert wird in der Norm für die unterschiedlichen Materialien sowie Bauweisen definiert und stellt eine sehr grobe Vereinfachung dar. Man erhält für jede berechnete Eigenform Schnittgrößen, welche abschließend zu einer umhüllenden Schnittgrößenkombination überlagert werden.

Ergänzend zu diesen zwei Basismethoden werden im EC8 zwei weitere Nachweisformate vorgeschlagen.

Nichtlineare Pushover-Analyse

Die nichtlineare Pushover-Analyse wird meist verformungsbasiert durchgeführt. Das heißt, dass die gesamte Struktur in die Lage einer maßgebenden Eigenform versetzt wird. Anschließend werden die Verformungen schrittweise erhöht, wobei die Vertikallasten (Eigengewicht und Nutzlasten) konstant gehalten werden. Für jeden Berechnungsschritt wird die resultierende Kraft am Fuß

des Gebäudes der Verschiebung an einem zuvor definierten Kontrollknoten, der sinnvoll im obersten Geschoß gewählt wird, gegenübergestellt. Mit zunehmenden Verformungen kommt es zu stetig steigenden Kräften in den Bauteilen und durch die Definition plastischer Gelenke oder eines Versagensmodells für das Material selbst kommt es zu einem nichtlinearen Zusammenhang zwischen der resultierenden Kraft am Fuß und der Verschiebung im Referenzknoten. Es kommt also zu einer Abflachung der Kraft-Verformungskurve, auch Kapazitätskurve genannt, und im Fall eines simulierten Materialversagens auch zu einem kompletten Versagen der Struktur. Nun muss ein Zusammenhang zwischen der Kapazitätskurve und dem Antwortspektrum nach EC8 hergestellt werden, um einen Nachweis nach Norm erbringen zu können. Dies geschieht über den sogenannten Performance Point, der den Schnittpunkt der Kapazitätskurve mit dem linear elastischen Antwortspektrum darstellt. Nun kann an der Abszisse des Spektrums die Zielverschiebung abgelesen werden.

Nichtlineare Zeitverlaufsberechnung

Die nichtlineare Zeitverlaufsberechnung ist jene Nachweismethode, bei der die Modellierung des Bauwerks der Realität am nächsten kommt. In ihrem Rahmen kommen tatsächlich aufgezeichnete oder künstlich generierte Erdbebenschriebe, also Verläufe der Bodenbeschleunigung über ein gewisses Zeitintervall, zur Anwendung. Das Gebäudemodell wird diesen Bodenbeschleunigungen ausgesetzt und es kommt somit zu wechselden Belastungszyklen in den Bauteilen, wobei durch die im Materialmodell berücksichtigten Versagenskriterien einzelne Bauteile Risse bekommen oder gänzlich ausfallen können. Da der Modellierungsaufwand im Vergleich zu den oben beschriebenen Methoden erheblich höher ist, wird dieses Verfahren meist nur für besonders gefährdete und wichtige Bauwerke eingesetzt.

Während sich das vereinfachte Antwortsprektrumverfahren gut für neue regelmäßige Stahlbetonkonstruktionen eignet und das multimodale Antwortspektrumverfahren bei ebensolchen Gebäuden, welche jedoch auch maßgebende Torsionseigenformen besitzen, zur Anwendung kommt, ist es häufig der Fall, dass durch keine dieser beiden linear elastischen Referenzmethoden ein Nachweis bestehender Gründerzeitgebäude aus Mauerwerk möglich ist. Es ist also durchaus sinnvoll, das Gebäudemodell realitätsnäher abzubilden, indem ein Materialmodell zugrundegelegt wird, dass Nichtlinearitäten beziehungsweise Versagen von Bauteilen darstellen kann. Diese Vorgehensweise führt zur nichtlinearen statischen Pushover-Analyse, die in dieser Arbeit im Speziellen untersucht und angewendet wird.

1.4 Konstruktive Ausführung von Gründerzeithäusern

In diesem Abschnitt werden die grundlegenden Konstruktionselemente eines typischen Gründerzeithauses kurz beschrieben, die in weiterer Folge den untersuchten Gebäudemodellen dieser Arbeit zugrundeliegen. Die Abbildung 1.1 zeigt einen Schnitt durch eines dieser Gebäude, wie sie in Wien Ende des 19. beziehungsweise Anfang des 20. Jahrhunderts errichtet wurden. Die grundlegenden Informationen zur konstruktiven Ausführung von Gründerzeithäusern wurden aus [9] entnommen.

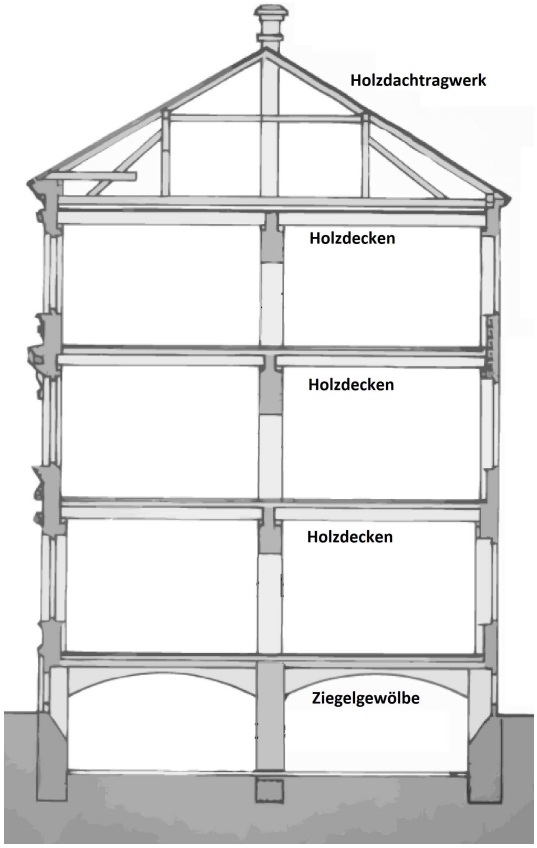

Abb. 1.1: Regelschnitt durch ein typisches Gründerzeithaus (adaptiert nach [10])

Im Folgenden werden die wesentlichen Tragelemente besprochen, die zum Lastabtrag sowohl in vertikaler als auch in horizontaler Richtung beitragen.

Die Wände wurden aus Ziegelmauerwerk im alten österreichischen Format ausgeführt. Tragende Außenwände und die Mittelmauer haben eine Mindeststärke von 30 cm, was zwei Ziegelreihen entspricht. Meist wurden die Wände in den unteren Geschoßen breiter ausgeführt und verjüngen sich dann nach oben hin. Die Querwände dienen aus statischer Sicht hauptsächlich der Aussteifung der Geschoße bei Horizontalbelastung, also unter Erdbebeneinwirkung. Diese bestehen zumeist aus einer einzigen Ziegelreihe und sind somit 15 cm stark.

Die Kellerdecken wurden in der Regel als Ziegelgewölbe ausgeführt. Dadurch haben sie eine aussteifende Wirkung. Die Kellergeschoße werden somit bei der Erdbebenberechnung nicht berücksichtigt, da die Decken als starrer Einspannhorizont betrachtet werden können.

Für die Geschoßdecken kamen zwei verschiedene Deckentypen zum Einsatz, wobei es sich bei beiden Varianten um Holzdecken handelt. In den Regelgeschoßen wurden meist Tramdecken hergestellt, bei denen einzelne Holzträme im Abstand von 60 cm bis 80 cm zwischen Außenmauer und Mittelmauer aufliegen. Diese Deckentypen sind aus statischer Sicht als nahezu schubweich zu betrachten. Im obersten Geschoß, also unter dem Dachboden, wurde meist eine sogenannte Dippelbaumdecke errichtet, die aus halben Baustämmen besteht, die nebeneinander gelegt mit Metalldübeln, daher der Name "Dippel", schubfest verbunden sind.

Kapitel 2

Formulierung des Finiten Elements RQuad

2.1 Grundlagen

Beim in dieser Arbeit implementierten Finiten Element handelt es sich um ein ebenes Schalenelement. Das bedeutet, dass es sowohl eine Scheibentragwirkung als auch eine Plattentragwirkung besitzt. Die klassischen finiten Scheibenelemente besitzen je Knotenpunkt zwei Freiheitsgrade in Elementebene, nämlich in den Verschiebungen $\mathbf{u}$ und $\mathbf{v}$. Plattenelemente besitzen einen Verschiebungsfreiheitsgrad $\mathbf{w}$ aus der Elementebene heraus und weiter zwei Verdrehungsfreiheitsgrade ϕ_x und ϕ_y. Bei Kombination eines Scheiben und eines Plattenelements durch Überlagerung der Steifigkeiten, wie es für ebene Elemente die übliche Vorgangsweise ist, erhält man ein ebenes Schalenelement. Es verbleibt jedoch der nicht definierte Freiheitsgrad ϕ_z, der Rotationen um den Normalvektor des Elements beschreibt. Dies kann zu numerischen Problemen führen, da die Steifigkeitsmatrix des Elements in der Zeile und Spalte für ϕ_z keine Einträge hat und somit Singularitäten auftreten können.[3]

Es wurde daher versucht, über bestimmte Annahmen geringe Steifigkeitswerte, welche nicht im Zusammenhang mit den tatsächlichen Verschiebungen stehen, zu definieren um dieses Problem zu umgehen. So beschäftigten sich seit Mitte der 1980er Jahre Wissenschaftler und Wissenschaftlerinnen im Fachbereich der Finiten Elemente mit der Frage, wie bei ebenen Membranproblemen mit dem Freiheitsgrad der Umdrehungen in der Ebene umzugehen sei [1, 5]. Ziel war es, eine Finite Elemente Formulierung mit vollen sechs Freiheitsgraden je Knotenpunkt zu erhalten, das heißt, für den Freiheitsgrad ϕ_z einen Verschiebungsansatz zu definieren. In den meisten wissenschaftlichen Beiträgen zu diesem Thema wird als erste Arbeit in diesem Gebiet auf jene von Allman [1] verwiesen, welche sich mit ebenen Dreieckselementen beschäftigt. Hier werden erstmals die beiden translatorischen Verschiebungsansätze mit einem Rotationsanatz, welcher in dieser Arbeit mit ω bezeichnet wird, kombiniert. So erhält man ein Scheibenelement mit drei vollen Freiheitsgraden je Knotenpunkt.

Dieses Dreieckselement wurde darauffolgend von Cook[4, 5] aufgegriffen und auf einen Viereckselement erweitert. Dieses bildet die Basis des im Rahmen dieser Arbeit definierten Elements. Eine sehr ausführliche übersichtliche Darstellung bezüglich der verwendeten Ansatzfunktionen und der Kombination der Freiheitsgrade von Scheiben und Plattenelement zum gewünschten ebenen Schalenelement wurden unterschiedlichen Publikationen entnommen. Kefal, Oterkus, Tessler und Spangler[16] bieten einen guten Überblick über die Freiheitsgrade und die Ansatzfunktionen für ein solches Element. Sie behandeln jedoch ein Element, welches Schubverzerrungen bei der Plattentragwirkung berücksichtigt. Dies wird vor allem bei Platten benötigt, deren Dicke nicht mehr klein gegenüber den Längen- und Breitenabmessungen ist. Da es sich jedoch bei den hier untersuchten Wandscheiben, für deren Modellierung das neue Element verwendet werden soll, um schlanke Bauteile handelt, wird von einer Berücksichtigung von Schubverzerrungen abgesehen. Bei Jin[7] entspricht der Scheibenanteil jenem des vorhin genannten Beitrags, beim Plattenanteil wird jedoch auf eine tradionelle Kirchhoff Platte gesetzt, welche für die in dieser Arbeit durchgeführten Berechnungen eine ausreichende Genauigkeit bietet. Weiterhin wird ausführlich auf die

einzelnen Ansatzfunktionen eingegangen und diese hergeleitet. Auf diesen Grundlagen baut die Formulierung des Finiten Elements **RQuad** auf, die im Folgenden näher beschrieben wird.

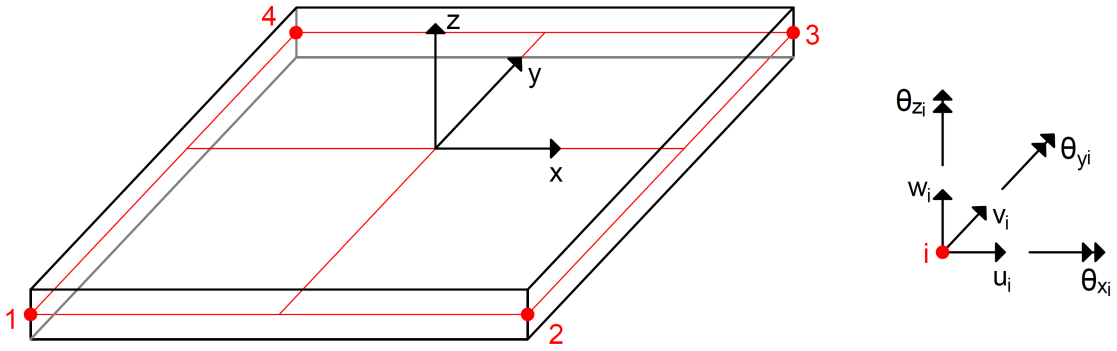

Abb. 2.1: Knotenfreiheitsgrade des Elements RQuad

2.2 Elementformulierung

Wie bereits eingangs erwähnt, wird auch für dieses Element der bewehrte Ansatz für ebene Schalenelemente verfolgt, bei dem die lokalen Membran- und Plattensteifigkeitsmatrizen berechnet und anschließend überlagert werden. Das Element besteht aus den vier Eckknoten, welche jeweils sechs Freiheitsgrade (drei Translations- und drei Rotationsfreiheitsgrade) aufweisen.

Ansatzfunktionen

Auf Grund der Tatsache, dass das Element im Rahmen dieser Arbeit zur Modellierung von Wandscheiben von Gebäuden vorgesehen wurde, erweist es sich als ausreichend, die Elementform auf Rechtecke zu beschränken. Dadurch bedingt wurde außerdem beschlossen, dass die Ansatzfunktionen, welche die Verschiebungen innerhalb des Elements interpolieren, nicht wie üblicherweise auf ein genormtes quadratisches Element mit den Eckpunktkoordinaten $\xi = \pm 1$ und $\eta = \pm 1$ bezogen werden, sondern in den tatsächlichen Elementabmessungen definiert werden. Auf diese Weise erspart man sich die dazu erforderliche Koordinatentransformation und die Berechnung kann etwas beschleunigt werden.

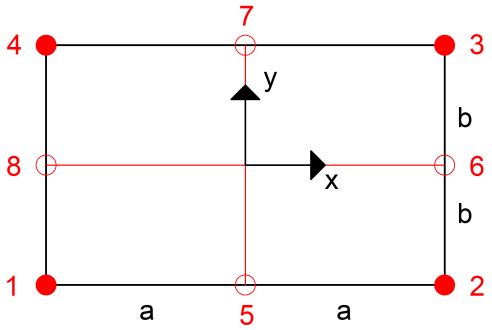

Abb. 2.2: Lokales Koordinatensystem des Elements RQuad

Nachfolgend werden die acht Ansatzfunktionen dargestellt. Die Länge **a** bezeichnet die halbe Elementbreite, bei der Länge **b** handelt es sich um die halbe Elementhöhe. Die ersten vier Funktionen 1-4 beziehen sich auf die vier Eckpunkte, während die Funktionen 5-8 die Seitenhalbierenden abbilden. Der Ursprung des lokalen Koordinatensystems wurde im Mittelpunkt des Rechtecks definiert, wobei die Achsen x und y wie in Abbildung 2.2 definiert wurden.

$$N_1 = \frac{(a-x)(b-y)}{2a2b} \tag{2.1}$$

$$N_2 = \frac{(a+x)(b-y)}{2a2b} \tag{2.2}$$

$$N_3 = \frac{(a+x)(b+y)}{2a2b} \tag{2.3}$$

$$N_4 = \frac{(a-x)(b+y)}{2a2b} \tag{2.4}$$

$$N_5 = \frac{(a^2-x^2)(b-y)}{2a^2b} \tag{2.5}$$

$$N_6 = \frac{(a+x)(b^2-y^2)}{2ab^2} \tag{2.6}$$

$$N_7 = \frac{(a^2-x^2)(b+y)}{2a^2b} \tag{2.7}$$

$$N_8 = \frac{(a-x)(b^2-y^2)}{2ab^2} \tag{2.8}$$

Auf Basis dieser Ansatzfunktionen werden die Funktionen $\mathbf{L_{1\text{-}4}}$ und $\mathbf{M_{1\text{-}4}}$ definiert, welche zur Interpolation des Rotationsfreiheitsgrades ϕ_z in Membranebene verwendet werden[16].

$$L_1 = \frac{1}{8}(y_{41}N_8 - y_{21}N_5) \tag{2.9}$$

$$L_2 = \frac{1}{8}(y_{12}N_5 - y_{32}N_6) \tag{2.10}$$

$$L_3 = \frac{1}{8}(y_{23}N_6 - y_{43}N_7) \tag{2.11}$$

$$L_4 = \frac{1}{8}(y_{34}N_7 - y_{14}N_8) \tag{2.12}$$

$$M_1 = \frac{1}{8}(x_{41}N_8 - x_{21}N_5) \tag{2.13}$$

$$M_2 = \frac{1}{8}(x_{12}N_5 - x_{32}N_6) \tag{2.14}$$

$$M_3 = \frac{1}{8}(x_{23}N_6 - x_{43}N_7) \tag{2.15}$$

$$M_4 = \frac{1}{8}(x_{34}N_7 - x_{14}N_8) \tag{2.16}$$

Zur Interpolation des Plattenanteils dienen die Funktionen $\mathbf{H_{x,1\text{-}12}}$ und $\mathbf{H_{y,1\text{-}12}}$, deren Herleitung ausführlich bei Jin[7] beschrieben ist. Für deren Berechnung werden die Ansatzfunktionen $\mathbf{N_{1-4}}$ durch die folgenden Beziehungen abgeändert, wobei der Index b für Biegung(bending) steht:

$$N_{1,b} = N_1 - \frac{1}{2}(N_8 + N_5) \tag{2.17}$$

$$N_{2,b} = N_2 - \frac{1}{2}(N_5 + N_6) \tag{2.18}$$

$$N_{3,b} = N_3 - \frac{1}{2}(N_6 + N_7) \tag{2.19}$$

$$N_{4,b} = N_4 - \frac{1}{2}(N_7 + N_8) \tag{2.20}$$

Die Funktionen $\mathbf{N_{5-8,b}}$ entsprechen den bereits oben definierten Ansatzfunktionen $\mathbf{N_{5-8}}$.

Weiters werden die nachstehend angeführten Hilfsgrößen berechnet, wobei die Indizes i und j die Eckknoten 1-4 und der Index k die Mittelpunkte 5-8 der vier Elementseiten bezeichnen:

$$l_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2} \tag{2.21}$$

$$b_k = \frac{\frac{1}{2}yij^2 - \frac{1}{4}x_{ij}^2}{l_{ij}^2} \tag{2.22}$$

$$c_k = \frac{-x_{ij}}{l_{ij}^2} \tag{2.23}$$

$$d_k = \frac{\frac{1}{2}xij^2 - \frac{1}{4}y_{ij}^2}{l_{ij}^2} \tag{2.24}$$

$$e_k = \frac{-y_{ij}}{l_{ij}^2} \tag{2.25}$$

Auf Basis der zuvor definierten Funktionen und Hilfsgrößen werden nun die eingangs erwähnten, direkt mit den Verschiebungsfreiheitsgraden verknüpften Ansatzfunktionen $\mathbf{H^x_{1-12}}$ und $\mathbf{H^y_{1-12}}$ definiert.

$$\begin{aligned}
H_1^x &= \frac{3}{2}(c_5 N_{5,b} - c_8 N_{8,b}) & H_1^y &= \frac{3}{2}(e_5 N_{5,b} - e_8 N_{8,b}) \\
H_2^x &= 0 & H_2^y &= N1,b + d_5 N_{5,b} + d_8 N_{8,b} \\
H_3^x &= -N1,b - b_5 N_{5,b} - b_8 N_{8,b} & H_3^y &= 0 \\
H_4^x &= \frac{3}{2}(c_6 N_{6,b} - c_5 N_{5,b}) & H_4^y &= \frac{3}{2}(e_6 N_{6,b} - e_5 N_{5,b}) \\
H_5^x &= 0 & H_5^y &= N2,b + d_6 N_{6,b} + d_5 N_{5,b} \\
H_6^x &= -N2,b - b_6 N_{6,b} - b_5 N_{5,b} & H_6^y &= 0 \\
H_7^x &= \frac{3}{2}(c_7 N_{7,b} - c_6 N_{6,b}) & H_7^y &= \frac{3}{2}(e_7 N_{7,b} - e_6 N_{6,b}) \\
H_8^x &= 0 & H_8^y &= N3,b + d_7 N_{7,b} + d_6 N_{6,b} \\
H_9^x &= -N3,b - b_7 N_{7,b} - b_6 N_{6,b} & H_9^y &= 0 \\
H_{10}^x &= \frac{3}{2}(c_8 N_{8,b} - c_7 N_{7,b}) & H_{10}^y &= \frac{3}{2}(e_8 N_{8,b} - e_7 N_{7,b}) \\
H_{11}^x &= 0 & H_{11}^y &= N4,b + d_8 N_{8,b} + d_7 N_{7,b} \\
H_{12}^x &= -N4,b - b_8 N_{8,b} - b_7 N_{7,b} & H_{12}^y &= 0
\end{aligned} \tag{2.26}$$

Diese Funktionen beziehen sich jeweils auf den Vektor der Verschiebungsgrößen des Elements für den Biegeanteil, der sich wie folgt zusammensetzt:

$$\mathbf{q} = (w_1,\theta_{x1},\theta_{y1},w_2,\theta_{x2},\theta_{y2},w_3,\theta_{x3},\theta_{y3},w_4,\theta_{x4},\theta_{y4})^T \tag{2.27}$$

Integrationspunkte

Da für den Biegeanteil des Elements die Kirchoffplattentheorie zugrundegelegt wird, bleiben die Querschnitte auch in der verformten Lage eben und somit sind die Dehnungen über die Querschnittsdicke linear. Zur Berücksichtigung der Plattensteifigkeit wurden die Integrationspunkte daher derart gewählt, dass zwei Ebenen von Integrationspunkten jeweils um den Abstand $\pm\frac{t}{2} * \frac{1}{\sqrt{3}}$ von der Mittelebene des Elements entfernt liegen, wobei $\mathbf{t}$ die gesamte Elementdicke bezeichnet. Beide dieser Ebenen setzen sich aus neun Integrationspunkten zusammen, die jeweils ein 3x3 Gauss Integrationsschema[3] bilden, wie in der Abbildung 2.3 übersichtlich dargestellt ist. Die Gewichte der Integrationspunkte für das 3x3 Integrationsschema und die Integration über die Elementdicke werden nachfolgend beschrieben.

Für Randpunkte gilt $\mathbf{w_c} = \frac{5}{9}$ und für den Mittelpunkt des Elements beträgt der Gewichtungsfaktor $\mathbf{w_m} = \frac{8}{9}$. Da für das Element RQuad die Integration nicht wie üblich über ein transformiertes quadratisches Element erfolgt, sondern vielmehr über die echte Elementgröße durchgeführt wird, müssen diese Gewichtsfaktoren mit der jeweiligen halben Seitenlänge $\mathbf{a}$ oder $\mathbf{b}$ multipliziert werden. Da die Integration über die Dicke des Elements einer Gauss-Quadratur mit zwei Stützpunkten entspricht, sind die Gewichte für beide Punkte jeweils $\mathbf{w_z} = \mathbf{1}$ und wirken sich daher nicht weiter auf die Berechnung aus. Je Integrationspunkt setzt sich der Gesamtge-

wichtsfaktor aus dem Produkt der Faktoren in die drei Integrationsrichtungen zusammen, wobei die Faktoren in **x** und **y** Richtung noch mit den tatsächlichen Elementabmessungen **a** und **b** skaliert werden.

$$\boldsymbol{W_i = w_x a \cdot w_y b \cdot w_z} \tag{2.28}$$

Exemplarisch wird die Berechnung für die Integrationspunkte **1**, **5** und **9** angegeben.

$$\begin{aligned} \boldsymbol{W_1} &= \boldsymbol{w_c a \cdot w_c * b \cdot w_z} = \frac{\mathbf{5}}{\mathbf{9}}\boldsymbol{a} \cdot \frac{\mathbf{5}}{\mathbf{9}}\boldsymbol{b} \cdot \mathbf{1} \\ \boldsymbol{W_5} &= \boldsymbol{w_m a \cdot w_c b \cdot w_z} = \frac{\mathbf{8}}{\mathbf{9}}\boldsymbol{a} \cdot \frac{\mathbf{5}}{\mathbf{9}}\boldsymbol{b} \cdot \mathbf{1} \\ \boldsymbol{W_9} &= \boldsymbol{w_m a \cdot w_m b \cdot w_z} = \frac{\mathbf{8}}{\mathbf{9}}\boldsymbol{a} \cdot \frac{\mathbf{8}}{\mathbf{9}}\boldsymbol{b} \cdot \mathbf{1} \end{aligned} \tag{2.29}$$

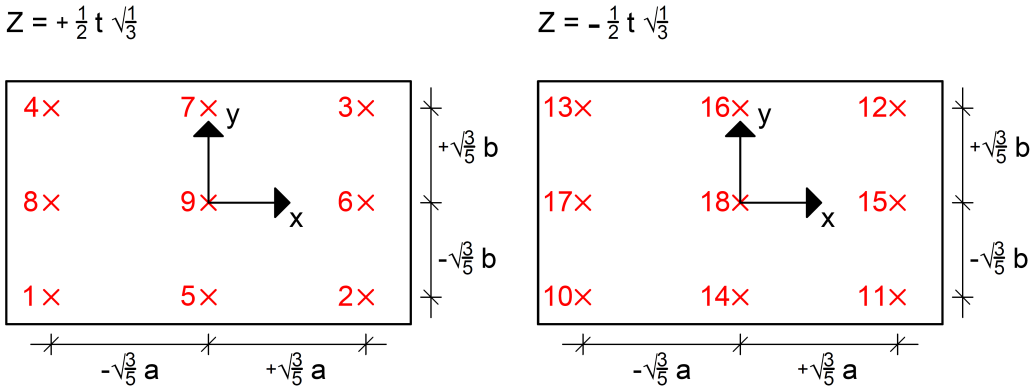

Abb. 2.3: Integrationspunkte des Elements RQuad

Neben diesen 18 Integrationspunkten verfügt das Element weiters über 8 Spannungspunkte, an welchen die Spannung ausgewertet wird. Diese sind mittig an den vier Elementrändern an der oberen und unteren Kante positioniert, wie aus Abbildung 2.4 hervorgeht.

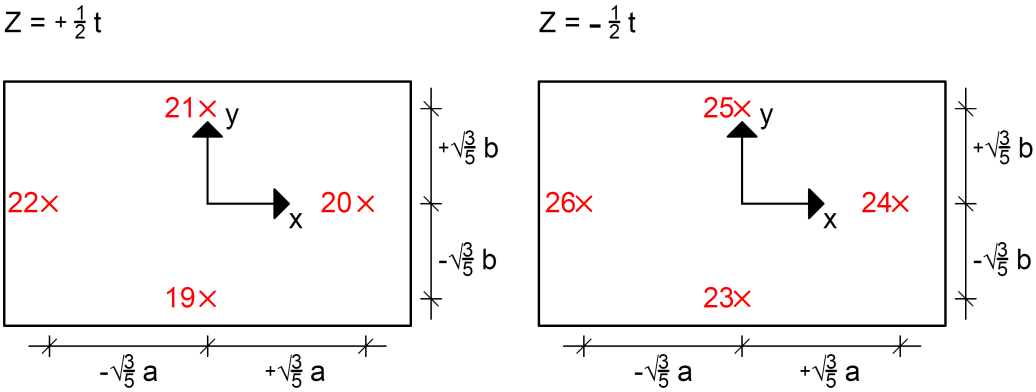

Abb. 2.4: Punkte zur Spannungsauswertung des Elements RQuad

Verschiebungsfeld und Dehnungs-Verschiebungsbeziehungen

Die nachfolgend angeführten Beziehungen wurden von [16] zuerst übernommen und anschließend die dort angesetzten Schubverzerrungen einer Mindlin-Reissner Platte durch die Annahmen der Kirchhoffplattentheorie ersetzt. Die z-Koordinate ist dabei, wie in Abbildung 2.1 ersichtlich, mit ihrem Ursprung in der Mittelfläche des Elements und den Extremwerten von $\pm\frac{t}{2}$ an der Ober- und Unterseite des Elements definiert. Somit lässt sich jeder Punkt des Elements durch die folgenden drei Komponenten des Verschiebungsvektors **u** eindeutig beschreiben:

$$u_x = u + z\theta_y = \sum_{i=1}^{4} N_i u_i + \sum_{i=1}^{4} L_i \theta_{zi} - z(\sum_{k=0}^{3} H^x_{1+3k} + \sum_{k=0}^{3} H^x_{2+3k} + \sum_{k=0}^{3} H^x_{3+3k}) \quad (2.30)$$

$$u_y = v + z\theta_x = \sum_{i=1}^{4} N_i v_i + \sum_{i=1}^{4} M_i \theta_{zi} - z(\sum_{k=0}^{3} H^y_{2+3k} + \sum_{k=0}^{3} H^y_{2+3k} + \sum_{k=0}^{3} H^y_{3+3k}) \quad (2.31)$$

$$u_z = w = \sum_{i=1}^{4} N_i w_i - \sum_{i=1}^{4} L_i \theta_{xi} - \sum_{i=1}^{4} M_i \theta_{yi} \quad (2.32)$$

Die linearen Dehnungs-Verschiebungsbeziehungen gemäß der linearen Elastizitätstheorie erhält man durch Ableiten der jeweiligen Komponenten des Verschiebungsvektors:

$$\varepsilon_{xx} = \frac{\partial u_x}{\partial x} = \frac{\partial u}{\partial x} + z\frac{\partial \theta_y}{\partial x} \quad (2.33)$$

$$\varepsilon_{yy} = \frac{\partial u_y}{\partial y} = \frac{\partial v}{\partial y} + z\frac{\partial \theta_x}{\partial y} \quad (2.34)$$

$$\gamma_{xy} = \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} = \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} + z(\frac{\partial \theta_y}{\partial y} + \frac{\partial \theta_x}{\partial x}) \quad (2.35)$$

Definition der H-Matrix und der B-Matrix

Die Basis eines jeden Finiten Elements bilden zwei Matrizen, die in der Literatur meist als H-Matrix und B-Matrix[3] bezeichnet werden. Erstere enthält alle Ansatzfunktionen für die zur Verfügung stehenden Freiheitsgrade sämtlicher Knoten. In der B-Matrix werden die Ableitungen der Ansatzfunktionen nach den Ableitungsrichtungen **dx**, **dy** und **dz** geordnet eingetragen. Mit Hilfe dieser Matrizen können später auf einfachem Weg Elementmatrizen wie die Elementsteifigkeitsmatrix und die Elementmassenmatrix berechnet werden. Die Basis dieser Matrizen bilden die im vorigen Unterpunkt näher beschriebenen Beziehungen. In den folgenden Gleichungen bezeichnet **q** den sogenannten Knotenverschiebungsvektor, dessen Komponenten die Freiheitsgrade der einzelnen Knoten sind. Somit ergeben sich für das gesamte Element eine H-Matrix der Dimension 3x24 und eine B-Matrix der Dimension 9x24. Um die Darstellung der B-Matrix möglichst übersichtlich zu halten, werden darin die Ableitungen der Ansatzfunktionen durch

einen hochgestellten Buchstaben ausgedrückt. $\frac{\partial N}{\partial x}$ wird beispielsweise als $\boldsymbol{N^x}$ bezeichnet, $\frac{\partial H^x}{\partial x}$ als $\boldsymbol{H^{xx}}$ angeschrieben.

$$\begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \mathbf{H}\cdot q = \begin{pmatrix} N_{1-4} & 0 & -zH^x_{1,4,7,10} & -zH^x_{2,5,8,11} & -zH^x_{3,6,9,12} & L_{1-4} \\ 0 & N_{1-4} & -zH^y_{1,4,7,10} & -zH^y_{2,5,8,11} & -zH^y_{3,6,9,12} & M_{1-4} \\ 0 & 0 & N_{1-4} & -L_{1-4} & -M_{1-4} & 0 \end{pmatrix} \cdot \begin{pmatrix} u_{1-4} \\ v_{1-4} \\ w_{1-4} \\ \theta_{x,1-4} \\ \theta_{y,1-4} \\ \theta_{z,1-4} \end{pmatrix} \tag{2.36}$$

$$\begin{pmatrix} \frac{\partial u_x}{\partial x} \\ \frac{\partial u_x}{\partial y} \\ \frac{\partial u_x}{\partial z} \\ \frac{\partial u_y}{\partial x} \\ \frac{\partial u_y}{\partial y} \\ \frac{\partial u_y}{\partial z} \\ \frac{\partial u_z}{\partial x} \\ \frac{\partial u_z}{\partial y} \\ \frac{\partial u_z}{\partial z} \end{pmatrix} = \mathbf{B}\cdot q = \begin{pmatrix} N^x_{1-4} & 0 & -zH^{xx}_{1,4,7,10} & -zH^{xx}_{2,5,8,11} & -zH^{xx}_{3,6,9,12} & L^x_{1-4} \\ N^y_{1-4} & 0 & -zH^{xy}_{1,4,7,10} & -zH^{xy}_{2,5,8,11} & -zH^{xy}_{3,6,9,12} & L^y_{1-4} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & N^x_{1-4} & -zH^{yx}_{1,4,7,10} & -zH^{yx}_{2,5,8,11} & -zH^{yx}_{3,6,9,12} & M^x_{1-4} \\ 0 & N^y_{1-4} & -zH^{yy}_{1,4,7,10} & -zH^{yy}_{2,5,8,11} & -zH^{yy}_{3,6,9,12} & M^y_{1-4} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & N^x_{1-4} & -L^x_{1-4} & -M^x_{1-4} & 0 \\ 0 & 0 & N^y_{1-4} & -L^x_{1-4} & -M^x_{1-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} u_{1-4} \\ v_{1-4} \\ w_{1-4} \\ \theta_{x,1-4} \\ \theta_{y,1-4} \\ \theta_{z,1-4} \end{pmatrix} \tag{2.37}$$

Berechnung der Elementsteifigkeitsmatrix

Um ein Finite Elemente System zu berechnen, werden im statischen Fall die bekannten äußeren Kräfte mit den gesuchten unbekannten Knotenpunktverschiebungen in eine Beziehung zueinander gesetzt. Diese Beziehung wird über die Steifigkeitsmatix **K** hergestellt. Es gilt folgende Gleichung, wobei der Vektor **U** die Knotenpunktverschiebungen enthält und der Vektor **F** die äußeren, an den Knoten angreifenden Kräfte.

$$\boldsymbol{K \cdot U = F} \tag{2.38}$$

Diese globale Steifigkeitsmatrix **K** setzt sich aus den einzelnen Elementsteifigkeitsmatrizen aller die Struktur bildenden Elemente zusammen. Nun soll die Berechnung dieser Elementsteifigkeitsmatrix für das hier beschriebene Element RQuad näher erläutert werden. Nachdem die oben übersichtlich abgebildeten H-Matrix und die B-Matrix definiert wurden, werden diese verwendet, um die weiteren Elementmatrizen zu bilden. Zuerst wird die Berechnung der Elementsteifigkeitsmatrix $\mathbf{K_{RQuad}}$ mit Hilfe der B-Matrix beschrieben. Die analytische Beziehung zur Ermittlung der Elementsteifigkeitsmatrix erfolgt durch die Integration über das Elementvolumen:

$$\boldsymbol{K_{RQuad} = \int_{V_e} B^T \cdot C \cdot B \cdot dV} \tag{2.39}$$

In der obigen Gleichung bezeichnet **C** den Elastizitätstensor, wessen Komponenten in Gleichung (2.40) ersichtlich sind.

$$\mathbf{C} = \begin{pmatrix} \frac{E}{(1-\nu^2)} & \frac{E\nu}{(1-\nu^2)} & 0 \\ \frac{E\nu}{(1-\nu^2)} & \frac{E}{(1-\nu^2)} & 0 \\ 0 & 0 & \frac{E(1-\nu)}{(1-\nu^2)} \end{pmatrix} \tag{2.40}$$

Im Rahmen der Finiten Elemente Methode wird der in Gleichung (2.39) vorhandene Integralausdruck diskretisiert, indem eine Summe der einzelnen Produkte $\boldsymbol{B^T \cdot C \cdot B}$ für jeden Integrationspunkt gebildet wird. Dafür müssen die beiden Matrizen $\mathbf{B_{Mem}}$ und $\mathbf{B_{Ben}}$ gebildet werden, welche die jeweiligen Membran- und Biegeanteile der B-Matrix enthalten. Diese Vorgehensweise rührt daher, weil die Steifigkeiten für den Membrananteil und den Biegeanteil getrennt berechnet und anschließend überlagert werden.

$$\mathbf{B_{Mem}} = \begin{pmatrix} \boldsymbol{N^x_{1-4}} & 0 & 0 & 0 & 0 & \boldsymbol{L^x_{1-4}} \\ 0 & \boldsymbol{N^y_{1-4}} & 0 & 0 & 0 & \boldsymbol{M^y_{1-4}} \\ \boldsymbol{N^y_{1-4}} & \boldsymbol{N^x_{1-4}} & 0 & 0 & 0 & \boldsymbol{L^y_{1-4} + M^x_{1-4}} \end{pmatrix} \tag{2.41}$$

$$\mathbf{B_{Ben}} = \begin{pmatrix} 0 & 0 & -\boldsymbol{zH^{xx}_{1,4,7,10}} & -\boldsymbol{zH^{xx}_{2,5,8,11}} & -\boldsymbol{zH^{xx}_{3,6,9,12}} & 0 \\ 0 & 0 & -\boldsymbol{zH^{yy}_{1,4,7,10}} & -\boldsymbol{zH^{yy}_{2,5,8,11}} & -\boldsymbol{zH^{yy}_{3,6,9,12}} & 0 \\ 0 & 0 & -\boldsymbol{zH^{xy}_{1,4,7,10}} - \boldsymbol{zH^{yx}_{1,4,7,10}} & -\boldsymbol{zH^{xy}_{2,5,8,11}} - \boldsymbol{zH^{yx}_{2,5,8,11}} & -\boldsymbol{zH^{xy}_{3,6,9,12}} - \boldsymbol{zH^{yx}_{3,6,9,12}} & 0 \end{pmatrix} \tag{2.42}$$

Nun kann die Membran- und Plattensteifigkeit berechnet werden. Anschließend werden, wie beschrieben, die Anteile der Scheibe und der Platte zur Elementsteifigkeitsmatrix $\mathbf{K_{RQuad}}$ überlagert.

$$\boldsymbol{K_{RQuad}} = \sum_{i=1}^{nIntPoints} (\boldsymbol{B^T_{Mem,i} \cdot C \cdot B_{Mem,i} \cdot \frac{t}{2} + B^T_{Ben,i} \cdot C \cdot B_{Ben,i} \cdot t) \cdot W_i} \tag{2.43}$$

In Gleichung 2.43 bezeichnet $\mathbf{t}$ die Elementdicke und W_i das Gewicht des jeweiligen Integrationspunktes. Da die Integrationspunkte wie in Abbildung 2.3 in zwei Ebenen angeordnet sind, muss der Membrananteil lediglich mit der halben Elementdicke multipliziert werden, um auf das vollständige Volumen zu kommen. Für den Biegeanteil erfolgt die Integration in z-Richtung, also der Dickenrichtung, wie erwähnt gemäß einer Gauß-Quadratur mit zwei Stützpunkten, weshalb dieser Anteil mit der gesamten Elementdicke multipliziert werden muss, um dem Elementvolumen zu entsprechen.

Berechnung der geometrischen Elementsteifigkeitsmatrix

Da im Rahmen dieser Arbeit Wandscheiben untersucht werden sollen, welche zu einem wesentlichem Teil in Scheibentragwirkung, also in der Ebene, beansprucht werden, erweist es sich als sinnvoll, auch die geometrische Elementsteifigkeitsmatrix $\mathbf{K_{Geo}}$ zu definieren. Durch diese ist es möglich, Stabilitätsversagen abzubilden und die Steifigkeit des Elements unter hohen Beanspruchungen in der Ebene abzumindern. Ein wesentlicher Punkt ist also, dass bei der Berechnung dieser Matrix im Unterschied zur Elementsteifigkeitsmatrix anstatt des Elastizitätstensors $\mathbf{C}$ ein Tensor, welcher den tatsächlich wirkenden Spannungszustand im Element beschreibt, berücksichtigt wird. Dieser wird folgend als $\mathbf{N}$ bezeichnet, was sinngemäß die Beanspruchung

durch Normalkräfte beschreiben soll. Aus diesem Grund muss bereits vor der Berechnung der geometrischen Steifigkeitsmatrix einmal die Berechnung der Struktur erfolgt sein, um den Deformationszustand des Elements zu kennen und so auch die im Element wirkenden Spannungen. Die Vorgehensweise erfolgt dabei in ähnlicher Weise wie bei Boutagouga und Djeghaba[6]. Der Tensor $\mathbf{N}$ setzt sich aus den vorhandenen Spannungen in Scheibentragwirkung zusammen.

$$\boldsymbol{N} = \begin{pmatrix} \boldsymbol{\sigma_{xx}} & \boldsymbol{\sigma_{xy}} \\ \boldsymbol{\sigma_{yx}} & \boldsymbol{\sigma_{yy}} \end{pmatrix} \tag{2.44}$$

Nun wird mit Hilfe der bereits bekannten B-Matrix die Matrix $\mathbf{B_{Geo}}$ gebildet, indem die Ableitungen der Ansatzfunktionen wiefolgt in eine 2x24 Matrix zusammengefasst werden.

$$\mathbf{B_{Geo}} = \begin{pmatrix} \boldsymbol{N^x_{1-4}} & \boldsymbol{N^x_{1-4}} & \boldsymbol{N^x_{1-4} - zH^{xx}_{1,4,7,10} - zH^{yx}_{1,4,7,10}} & \boldsymbol{-L^x_{1-4} - zH^{xx}_{2,5,8,11} - zH^{yx}_{2,5,8,11}} \\ \boldsymbol{N^y_{1-4}} & \boldsymbol{N^y_{1-4}} & \boldsymbol{N^y_{1-4} - zH^{xy}_{1,4,7,10} - zH^{yy}_{1,4,7,10}} & \boldsymbol{-L^y_{1-4} - zH^{xy}_{2,5,8,11} - zH^{yy}_{2,5,8,11}} \\ & & \boldsymbol{-M^x_{1-4} - zH^{xx}_{3,6,9,12} - zH^{yx}_{3,6,9,12}} & \boldsymbol{L^x_{1-4} + M^x_{1-4}} \\ & & \boldsymbol{-M^y_{1-4} - zH^{xy}_{3,6,9,12} - zH^{yy}_{3,6,9,12}} & \boldsymbol{L^y_{1-4} + M^y_{1-4}} \end{pmatrix} \tag{2.45}$$

Analog der Berechnung der Elementsteifigkeitsmatrix kann die geometrische Elementsteifigkeitsmatrix $\mathbf{K_{Geo}}$ bestimmt werden, indem wieder die Summe des Produkts $\boldsymbol{B^T_{Geo}} \cdot \boldsymbol{N} \cdot \boldsymbol{B_{Geo}}$ von allen Integrationspunkten gebildet wird.

$$\boldsymbol{K_{Geo}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{nIntPoints}} \boldsymbol{B^T_{Geo,i}} \cdot \boldsymbol{N} \cdot \boldsymbol{B_{Geo,i}} \cdot \frac{\boldsymbol{t}}{\boldsymbol{2}} \cdot \boldsymbol{W_i} \tag{2.46}$$

Bestimmung der Elementmassenmatrix

Die Massenmatrix wird in der Finite Elemente Methode benötigt, um Eigenformen einer Struktur zu berechnen, indem das nachfolgende Gleichungssystem gelöst wird.

$$(\boldsymbol{K} - \boldsymbol{\omega^2} \cdot \boldsymbol{M}) \cdot \boldsymbol{\varphi} = \mathbf{0} \tag{2.47}$$

In obiger Gleichung enthält der Vektor $\boldsymbol{\omega^2}$ die Eigenwerte, also die Eigenkreisfrequenzen $\boldsymbol{\omega}$ der Struktur, während in den Spalten der Matrix $\boldsymbol{\varphi}$ die Eigenformen enthalten sind. Es ist also ersichtlich, dass die Ermittlung der Elementmassenmatrix $\mathbf{M}$ genau wie die der Elementsteifigkeitsmatrix $\mathbf{K}$ unerlässlich für die Druchführung von Erdbebenberechnungen und insbesondere der hier näher betrachteten Pushover-Analyse ist.

Weiters ist die Massenmatrix erforderlich, um dynamische Berechnungen durchzuführen, also Berechnungen, in welchen die Kräfte in der Struktur durch die Beschleunigung der Elementmassen hervorgerufen werden. Hier erweitert sich die zuvor beschriebene Gleichung 2.38 um einen weiteren Anteil. In Gleichung 2.48 bezeichnet $\mathbf{M}$ die Massenmatrix und $\ddot{\mathbf{U}}$ die Knotenpunktbeschleunigungen, welche als zweite Ableitung der Knotenpunktverschiebungen definiert sind.

$$\boldsymbol{M} \cdot \boldsymbol{\ddot{U}} + \boldsymbol{K} \cdot \boldsymbol{U} = \boldsymbol{F} \tag{2.48}$$

Nachfolgend wird die Berechnung der Elementmassenmatrix $\mathbf{M_{RQuad}}$ beschrieben. Die Masse eines Körpers entspricht bei analystischer Berechnung der Integration der Dichte des Körpers über das Volumen. Um die Massenmatrix $\mathbf{M_{RQuad}}$ des Elements zu berechnen, bedient man sich der zuvor bestimmten H-Matrix[3].

$$\boldsymbol{M = \int_{V_e} \rho \cdot H^T \cdot H dV} \tag{2.49}$$

$$\boldsymbol{M_{RQuad} = \sum_{i=1}^{nIntPoints} \rho \cdot H^T \cdot H \cdot \frac{t}{2} \cdot W_i} \tag{2.50}$$

Wie beim Membrananteil der Elementsteifigkeitsmatrix erfolgt die Multiplikation mit der halben Elementdicke, um die beiden Ebenen an Integrationspunkten zu berücksichtigen und das gesamte Elementvolumen korrekt zu erfassen.

Kapitel 3

Numerische Beispiele und Vergleich mit analytischen Ergebnissen

In diesem Abschnitt sollen verschiedene Strukturen, welche durch Elemente des Typs $\mathbf{R_{Quad}}$ diskretisiert wurden, untersucht werden. Es sollen die Berechnungsergebnisse, meist in anschaulicher Form die Verformung an einer bestimmten Stelle, mit den analytischen Ergebnissen verglichen werden. Die Berechnung in diesem Abschnitt erfolgt nach der linearen Elastizitätstheorie. Die **slangTNG** Skripte der hier vorgestellten Beispiele können dem Anhang D entnommen werden.

3.1 Membrantragwirkung

In diesem Unterpunkt soll überprüft werden, wie sich das Element unter reiner Scheibenbeanspruchung verhält. Dazu wird ein Kragarm bestehend aus vier quadratischen Elementen modelliert, die jeweils die Abmessungen $\mathbf{2 \cdot a = 12}$ und $\mathbf{2 \cdot b = 12}$ aufweisen. Die Elementdicke wird zu $\boldsymbol{t = 1}$ festgelegt. Weitere Parameter sind aus der nachstehenden Tabelle 3.1 beziehungsweise aus Abbildung 3.1 zu entnehmen. Die einwirkende Kraft wird je zur halben Größe auf den oberen und unteren Knoten am Ende des Kragarms aufgebracht.

Tab. 3.1: Berechnungsparameter

Balkenlänge	$\boldsymbol{L = 48}$
Querschnittsbreite	$\boldsymbol{B = 1}$
Querschnittshöhe	$\boldsymbol{H = 12}$
Elastizitätsmodul	$\boldsymbol{E = 30000}$
Querdehnungszahl	$\boldsymbol{\nu = 0}$
Einwirkende Kraft	$\boldsymbol{F = 40}$

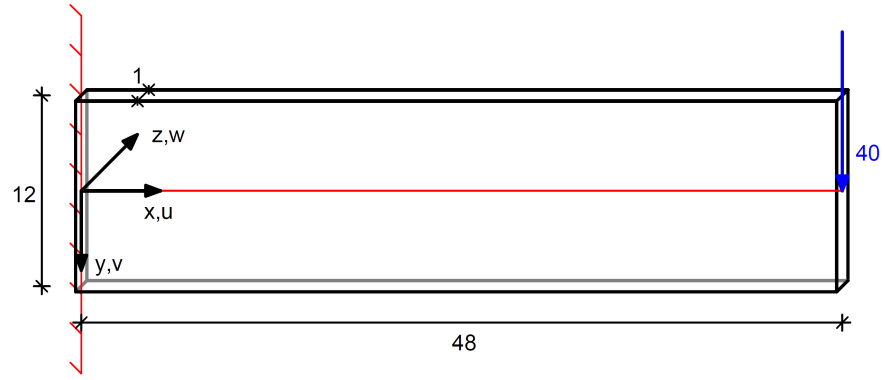

Abb. 3.1: Statisches System - Kragbalken unter Membrantragwirkung

Die analytische Lösung für die Durchbiegung am Ende eines Kragarms unter Einwirkung einer Einzellast wird nachfolgend berechnet.

$$\begin{aligned} I &= \frac{B \cdot H^3}{12} = \frac{1 \cdot 12^3}{12} = 144 \\ v &= \frac{F \cdot L^3}{3 \cdot E \cdot I} = \frac{40 \cdot 48^3}{3 \cdot 30000 \cdot 144} = 0{,}341 \end{aligned} \tag{3.1}$$

Nun erfolgt die numerische Berechnung der Durchbiegung mittels **slangTNG**[17] und dem darin neu implementierten Element $\mathbf{R_{Quad}}$. Die folgenden Abbildungen zeigen die Struktur einmal in unverformter und anschließend in verformter Lage. Die vertikale Durchbiegung zufolge der numerischen Berechnung beträgt $\boldsymbol{v = 0.347}$ und stimmt damit sehr gut mit der analystischen Lösung überein. Bei einer dichteren Vernetzung des Kragbalkens kann davon ausgegangen werden, dass die analystische Lösung nahezu exakt erreicht wird.

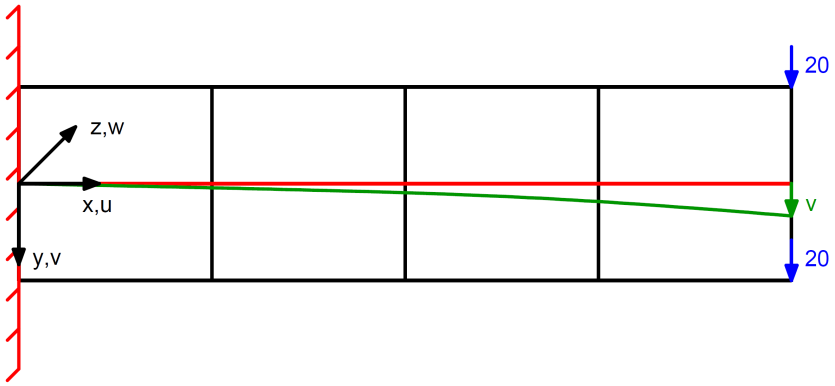

Abb. 3.2: Elementdiskretisierung - 4x $\mathbf{R_{Quad}}$

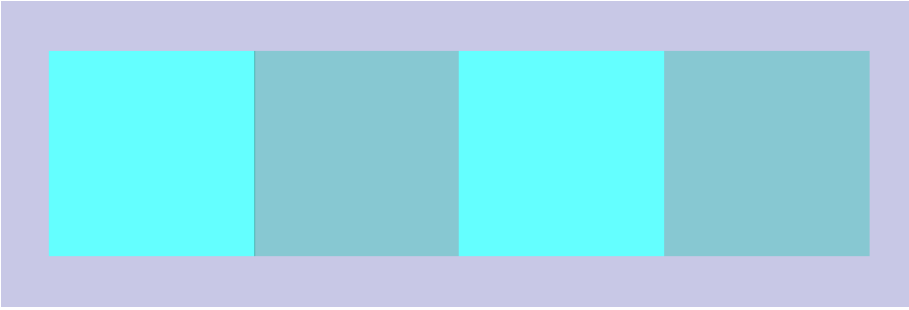

Abb. 3.3: Unverformte Lage

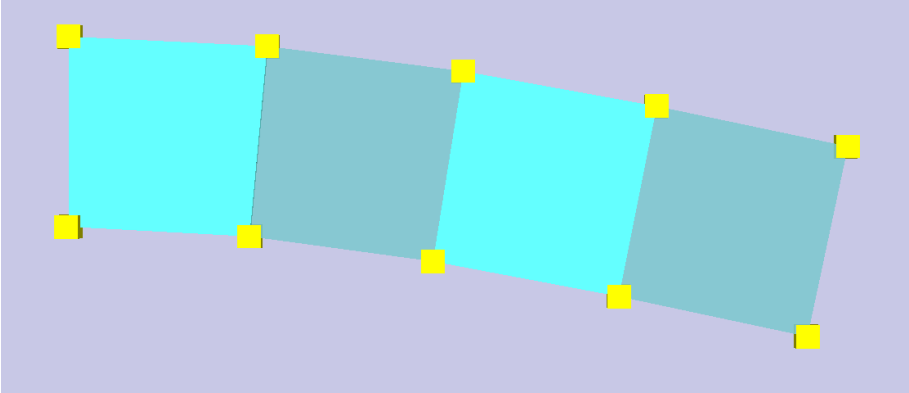

Abb. 3.4: Verformte Lage unter Scheibenbeanspruchung

3.2 Plattentragwirkung

Wie beim Beispiel zur Membrantragwirkung wird auch zur Überprüfung der Plattentragwirkung der im vorigen Abschnitt beschriebene Kragarm verwendet. Sämtliche Parameter aus Tabelle 3.1 bleiben unverändert, lediglich die Belastung wird nun normal zur Elementoberfläche, wie in Abbildung 3.5 dargestellt, aufgebracht. Es erfolgt zuerst die Berechnung der analytischen Lösung.

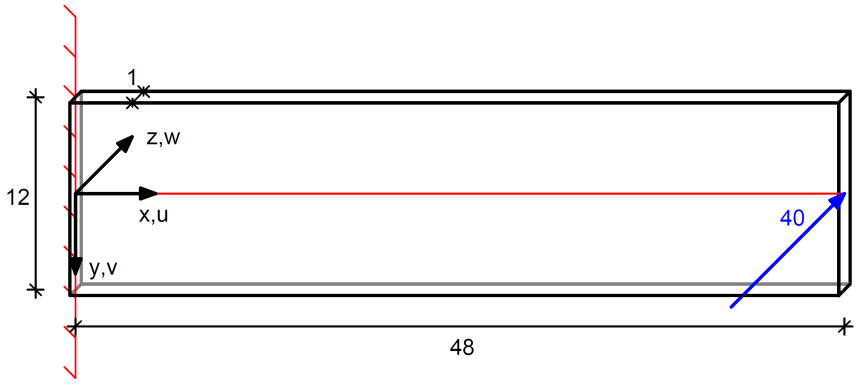

Abb. 3.5: Statisches System - Kragbalken unter Plattentragwirkung

$$I = \frac{H \cdot B^3}{12} = \frac{12 \cdot 1^3}{12} = 1$$
$$w = \frac{F \cdot L^3}{3 \cdot E \cdot I} = \frac{40 \cdot 48^3}{3 \cdot 30000 \cdot 1} = 49{,}15 \tag{3.2}$$

Die numerische Berechnung erfolgt für das identische System wie oben. Die Durchbiegung beträgt nun $w = 55{,}30$, was ebenfalls in guter Näherung zur analytischen Lösung liegt. Auch hier lässt sich durch eine Verdichtung des Netzes die Genauigkeit noch weiter steigern.

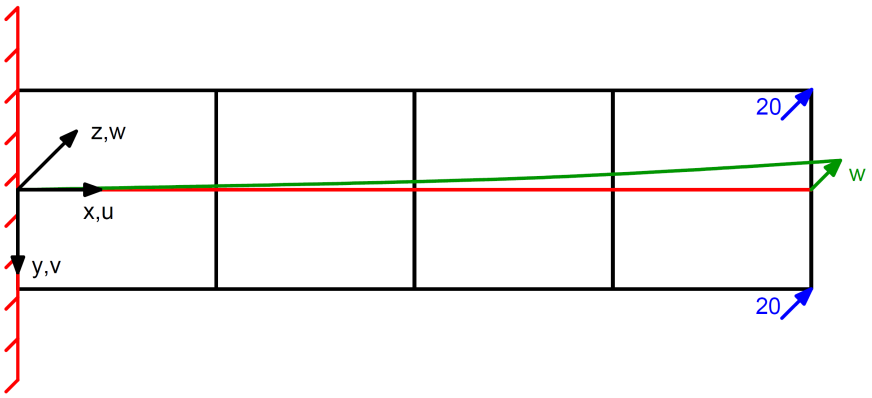

Abb. 3.6: Elementdiskretisierung - 4x $\mathbf{R_{Quad}}$

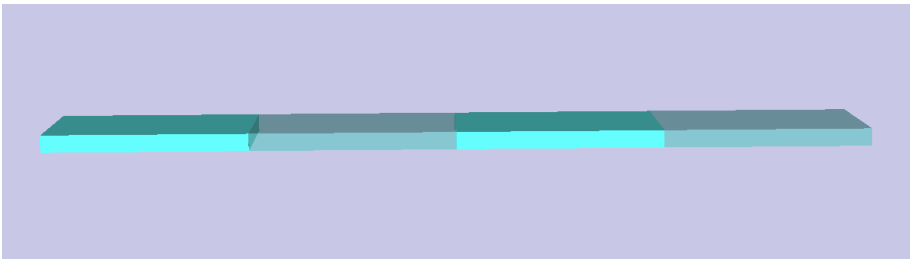

Abb. 3.7: Unverformte Lage

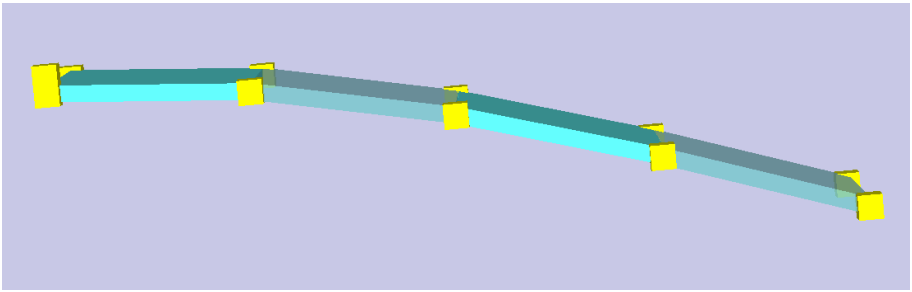

Abb. 3.8: Verformte Lage unter Plattenbeanspruchung

3.3 Eigenfrequenzen und Eigenschwingungsformen

Nach den ersten beiden Unterkapiteln, die im Grunde der Verifizierung der Elementsteifigkeitsmatrix **K** dienten, soll in diesem Unterpunkt die Massenmatrix **M** überprüft werden. Dies geschieht auf nachvollziehbarem Wege durch Berechnung der Eigenfrequenzen einer Struktur gemäß Formel 2.47, da für einfache Systeme auch analytische Lösungen für die Eigenfrquenzen existieren. Da die Steifigkeitsmatrix durch die bereits erbrachten Vergleiche als ausreichend genau betrachtet werden kann, hat lediglich die Massenmatrix einen Einfluss auf die gesuchten Eigenfrequenzen. Als System zur Kontrolle kommt abermals ein Kragarm zum Einsatz. Es wird die erste Eigenfrequenz zuerst mittels Formel und anschließend durch Finite Elemente in **slangTNG** berechnet. Als zusätzlicher Berechnungsparameter fließt hier nun auch die Dichte des Materials mit der Größe $\boldsymbol{\rho = 1}$ ein. Die Grundeigenfrequenz eines Kragarms lässt sich für den Balken unter reiner Eigenlast durch folgende Näherungsformel berechnen, wobei $\boldsymbol{m}$ die Masse je Längeneinheit des Balkens bezeichnet.

$$\boldsymbol{f_1 = 0{,}560 \cdot \sqrt{\frac{E \cdot I}{m \cdot L^4}} = 0{,}560 \cdot \sqrt{\frac{30000 \cdot 1}{1 \cdot 12 \cdot 1 \cdot 48^4}} = 0{,}01215} \tag{3.3}$$

Die Lösung der Gleichung 3.4 für den Vektor $\boldsymbol{\omega}$ führt zu den Eigenkreisfrequenzen. Diese werden anschließend durch $\boldsymbol{2 \cdot \pi}$ geteilt, um die Eigenfrequenzen der Struktur zu erhalten. Der Rechengang ist nachfolgend dargestellt. Die Matrix $\boldsymbol{\varphi}$ enthält Spaltenweise die der jeweiligen Eigenkreisfrequenz zugehörigen Vektoren der Eigenformen.

$$\boldsymbol{(K_{RQuad} - \omega^2 \cdot M_{RQuad}) \cdot \varphi = 0} \tag{3.4}$$

$$\boldsymbol{\det(K_{RQuad} - \omega^2 \cdot M_{RQuad}) \cdot = 0} \tag{3.5}$$

$$\boldsymbol{\omega_1 = 0{,}0715} \tag{3.6}$$

$$\boldsymbol{f_1 = \frac{\omega_1}{2 \cdot \pi} = \frac{0{,}0715}{2 \cdot \pi} = 0{,}01138} \tag{3.7}$$

Durch den Vergleich der beiden Lösungen ergibt sich eine Abweichung von $\boldsymbol{1 - \frac{0{,}01138}{0{,}01215} = 0{,}06}$. Bei dichterer Vernetzung kann auch hier davon ausgegangen werden, dass sich das Finite Elemente Ergebnis weiter der analytischen Lösung angleicht. Für den Zweck der Überprüfung der Massenmatrix ist diese geringe Abweichung vernachlässigbar und es kann somit bestätigt werden, dass die Elementmassenmatrix $\mathbf{M_{RQuad}}$ eine ausreichende Genauigkeit aufweist. Die nachfolgende Abbildung zeigt die zur berechneten Eigenfrequenz zugehörige erste Eigenform $\boldsymbol{\varphi_1}$.

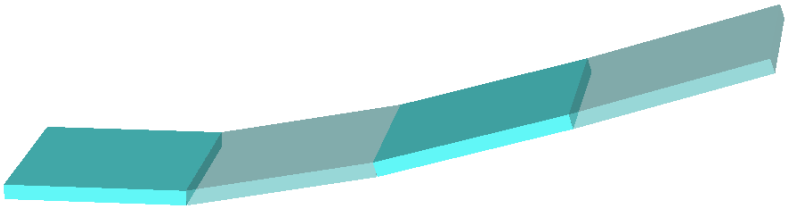

Abb. 3.9: Erste Eigenform des untersuchten Kragarms

3.4 Stabilität - Beulen

Als letzte der in Kapitel 2 beschriebenen Elementmatrizen soll die geometrische Steifigkeitsmatrix $\mathbf{K_{Geo}}$ überprüft werden. Dazu wird wieder ein Kragarm untersucht, welcher diesmal von einer Normalkraft auf Druck beansprucht wird. Es wird die Knicklast des Systems, welches dem ersten Eulerfall entspricht, berechnet. Die Abmessungen des Kragarms und die Materialparameter wurden aus den vorgehenden Beispielen unverändert übernommen. Die analytisch exakte Knicklast eines druckbeaspruchten Kragträgers ist wie in Gleichung 3.8 zu ermitteln. In dieser Gleichung bezeichnet $\boldsymbol{L_k}$ die Knicklänge des Stabes, welche für einen Kragarm der doppelten Systemlänge, also $\mathbf{2 \cdot L}$, entspricht. Maßgebend ist die Knickfigur um die schwache Querschnittsachse.

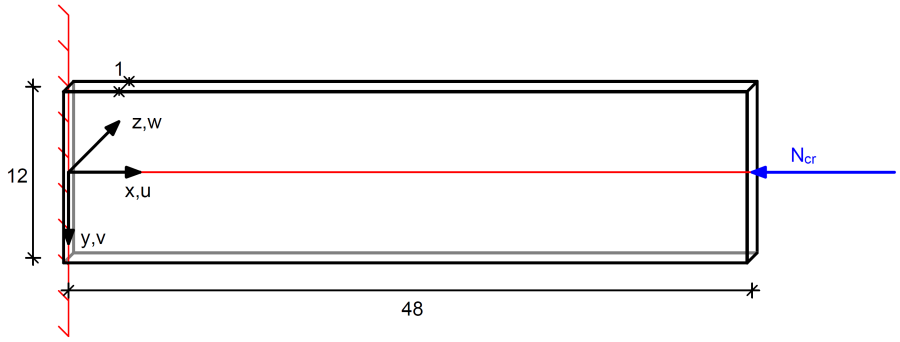

Abb. 3.10: Statisches System - Kragbalken unter Einwirkung einer zentrischen Knicklast

$$\boldsymbol{N_{cr}} = (\frac{\boldsymbol{\pi}}{\boldsymbol{L_k}})^{\mathbf{2}} \cdot \boldsymbol{E} \cdot \boldsymbol{I} = (\frac{\boldsymbol{\pi}}{\mathbf{2 \cdot 48}})^{\mathbf{2}} \cdot \mathbf{30000 \cdot 1 = 32{,}13} \tag{3.8}$$

Nun soll zum Vergleich die Knicklast mit Hilfe des Finiten Elements $\mathbf{R_{Quad}}$ berechnet werden. Im Unterschied zu den vorhergehenden Beispielen wird der Kragarm nun in zehn Elemente unterteilt, um grafisch eine bessere Darstellung der Knickform zu ermöglichen. Die numerische Vorgehensweise zur Ermittlung der Knicklast ist durch das gegebene Skript File in Anhang D vollständig dargestellt. Für den hier untersuchten Kragarm ergibt sich die Knicklast zu $\boldsymbol{N_{cr,RQuad}} = \mathbf{25{,}50}$, was in guter Näherung zur analytischen Lösung steht. Die numerische Lösung liegt auf jeden Fall unterhalb der analytischen Knicklast, da in der Finite Elemente Berechnung ein Ausknicken um beide Querschnittsachsen berücksichtigt wurde, während in der analytischen Vergleichsrechnung ausschließlich das Knicken um die schwache Querschnittsachse berechnet wurde.

Es wird darauf hingewiesen, dass die erreichte Genauigkeit der numerischen Berechnung von der Feinheit des Netzes der Finiten Elemente und dem Verhältnis der Querschnittsabmessungen abhängt. Für schlanke Querschnitte ist die erreichte Genauigkeit wesentlich höher als für Querschnitte, deren Dicke groß im Verhältnis zu ihrer Höhe ist.

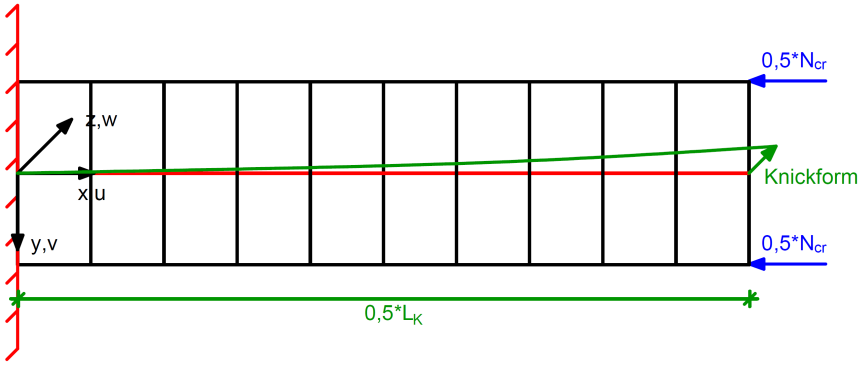

Abb. 3.11: Elementdiskretisierung - 10x $\mathbf{R_{Quad}}$

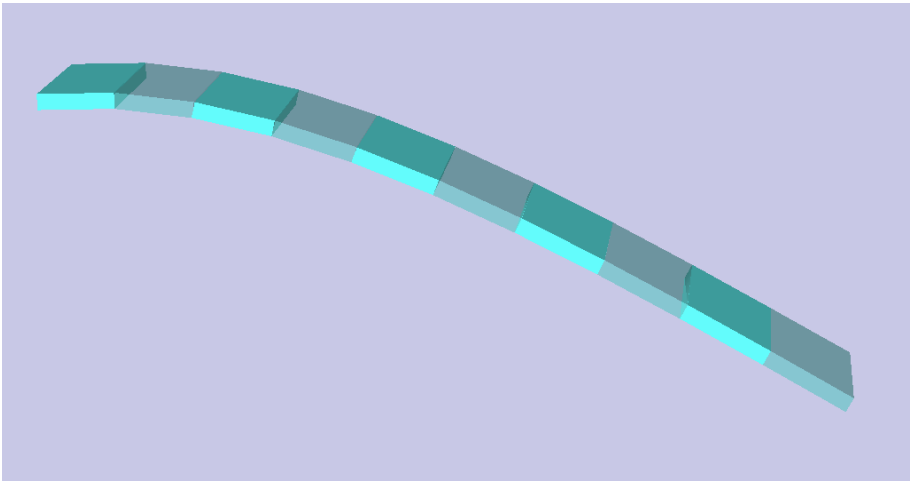

Abb. 3.12: Knickfigur - Unterteilung des Stabes in zehn Elemente

Kapitel 4

Definition des Materialmodells ElasticFailure

Nachdem in den vorangegangenen Abschnitten das Element $\mathbf{R_{Quad}}$ vorgestellt und dessen Genauigkeit durch verschiedene Beispiele überprüft wurde, wird sich dieses Kapitel damit beschäftigen, ein übersichtliches Materialmodell zu implementieren, das elastisches, sprödes Versagen abbilden kann. Dies ist von Notwendigkeit, da im Anschluss, wie eingangs erwähnt, Pushover-Analysen durchgeführt werden sollen, für welche zwingend ein Materialgesetz erforderlich ist, das ein Abmindern der Elementsteifigkeiten oder gar Ausfallen von ganzen Elementen zulässt.

4.1 Formulierung des Materialmodells

Um das Versagensmodell möglichst allgemein zu halten, wurde entschieden, zwei zulässige Grenzspannungen zu definieren, eine für eine maximal zulässige Zugspannung $\boldsymbol{f_t}$ und eine für eine maximal zulässige Druckspannung $\boldsymbol{f_c}$. Nach erstmaliger Berechnung des Finite Elemente Systems sind die vorhandenen Spannungen $\boldsymbol{\sigma}$ in allen Elementen und all deren Integrationspunkten durch die Beziehung in Gleichung 4.1 bekannt.

$$\boldsymbol{\sigma} = \boldsymbol{C} \cdot \boldsymbol{\varepsilon} \tag{4.1}$$

Die Matrix $\mathbf{C}$ ist der bereits in den vorigen Kapiteln definierte Elastizitätstensor. Die 3x3 Matrix $\boldsymbol{\varepsilon}$ wird Verzerrungstensor genannt und enthält die im Zuge der Finite Elemente Berechnung ermittelten Verezerrungen. Der Spannungstensor $\boldsymbol{\sigma}$ ist aus Gleichung 4.2 ersichtlich.

$$\boldsymbol{\sigma} = \begin{pmatrix} \boldsymbol{\sigma_{xx}} & \boldsymbol{\sigma_{xy}} & \boldsymbol{\sigma_{xz}} \\ \boldsymbol{\sigma_{yx}} & \boldsymbol{\sigma_{yy}} & \boldsymbol{\sigma_{yz}} \\ \boldsymbol{\sigma_{zx}} & \boldsymbol{\sigma_{zy}} & \boldsymbol{\sigma_{zz}} \end{pmatrix} \tag{4.2}$$

Allgemein ist der Spannungstensor $\boldsymbol{\sigma}$ im xyz-Koordinatensystem definiert. Um die vorhandenen Spannungen mit den Grenzspannung vergleichen zu können, wird ein Koordinatensystem gewählt, in welchem keine Schubspannungen wirken. Dies geschieht durch die Transformation des Spannungstensors in ein Koordinatensystem, in welchem die Koordinatenachsen in die gleichen Wirkungsrichtungen wie die Normalspannungen zeigen und somit lediglich Einträge in der Hauptdiagonale des Spannungstensors verbleiben. Diese drei verbleibenen Einträge im Spannungstensor $\boldsymbol{\sigma_I}$, $\boldsymbol{\sigma_{II}}$ und $\boldsymbol{\sigma_{III}}$ werden deshalb Hauptnormalspannungen genannt. Die Berechnung erfolgt durch die Lösung des Eigenwertproblems gemäß Gleichung 4.3.

$$\boldsymbol{det(\sigma - \sigma_{I,II,III} \cdot I)} = \boldsymbol{det}\left(\begin{pmatrix} \boldsymbol{\sigma_{xx}} & \boldsymbol{\sigma_{xy}} & \boldsymbol{\sigma_{xz}} \\ \boldsymbol{\sigma_{yx}} & \boldsymbol{\sigma_{yy}} & \boldsymbol{\sigma_{yz}} \\ \boldsymbol{\sigma_{zx}} & \boldsymbol{\sigma_{zy}} & \boldsymbol{\sigma_{zz}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{\sigma_I} \\ \boldsymbol{\sigma_{II}} \\ \boldsymbol{\sigma_{III}} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix} \right) = \mathbf{0} \tag{4.3}$$

Im Fall des ebenen Schalenelements reduzieren sich die von **0** verschiedenen Einträge des Spannungstensors wie in Gleichung 4.4 angegeben.

$$\boldsymbol{\sigma} = \begin{pmatrix} \boldsymbol{\sigma_{xx}} & \boldsymbol{\sigma_{xy}} & \mathbf{0} \\ \boldsymbol{\sigma_{yx}} & \boldsymbol{\sigma_{yy}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{4.4}$$

Somit werden lediglich die Hauptnormalspannungen $\boldsymbol{\sigma_I}$ und $\boldsymbol{\sigma_{II}}$ benötigt. Die ermittelten Hauptnormalspannungen werden mit den eingangs definierten zulässigen Spannungen $\boldsymbol{f_t}$ und $\boldsymbol{f_c}$ verglichen. Zugspannungen sind hierbei positiv und Druckspannungen negativ definiert.

$$\begin{aligned} \boldsymbol{\sigma_I} &> \boldsymbol{f_t} \\ \boldsymbol{\sigma_{II}} &> \boldsymbol{f_t} \\ \boldsymbol{\sigma_I} &< \boldsymbol{f_c} \\ \boldsymbol{\sigma_{II}} &< \boldsymbol{f_c} \end{aligned} \tag{4.5}$$

Kommt es in einem Integrationspunkt zu einer Überschreitung beziehungsweise Unterschreitung eines der beiden zulässigen Werte, wird die Steifigkeit für diesen Integrationspunkt **0** gesetzt und er fällt somit aus[12]. Das bedeutet, dass dieser Punkt nicht mehr zur Elementsteifigkeit beiträgt. Dies wird durch die Annahme des Elastizitätsmoduls von $\boldsymbol{E} = \mathbf{0}$ umgesetzt, wodurch die gesamte Steifigkeitsmatrix für diesen Integrationspunkt $\boldsymbol{K_{IntPoint}} = \mathbf{0}$ wird. Im Materialgesetz **ElasticFailure** wird unterschieden, ob das Materialversagen durch ein Überschreiten der zulässigen Zugspannung oder ein Unterschreiten der zulässigen Druckspannung ausgelöst wird. Für jedes Element existiert je eine Matrix für Zug- und Druckversagen, aus der hervorgeht, in welchen Integrationspunkten es zu Zug- oder Druckversagen gekommen ist. Diese Matrix wird nachfolgend **Versagensmatrix** bezeichnet. Näheres dazu ist in den Anwendungsbeispielen der Kapitel 4.2 und 4.3 ersichtlich.

Es muss so lange eine Iteration durchgeführt werden, bis keine neuen Integrationspunkte mehr ausfallen und sich ein Gleichgewicht einstellt. Kann durch diese Vorgehensweise kein Gleichgewicht gefunden werden, das heißt fallen mit jedem Iterationsschritt immer mehr Punkte und Elemente aus, kommt es schließlich zu einer Singularität der Steifigkeitsmatrix **K**. Folglich kann die grundlegende Gleichung $\boldsymbol{U} \cdot \boldsymbol{K} = \boldsymbol{F}$ nicht mehr gelöst werden.

4.2 Mauerwerkswand unter reiner Scheibenbeanspruchung

Zur Demonstration des zuvor vorgestellten Materialmodells **ElasticFailure** wird nun ein Anwendungsbeispiel präsentiert. Dieses Beispiel zeigt anschaulich, wie sich die finiten Elemente nach überschreiten beziehungsweise unterschreiten der definierten zulässigen Grenzspannungen verhalten.

In diesem Beispiel wird eine Wandscheibe, welche am unteren Ende gelagert ist, am oberen freien Seite durch eine Querlast beansprucht. Das statische System ist in Abbildung 4.1 ersichtlich. Die Diskretisierung erfolgt durch 16 finite Elemente, wie aus Abbildung 4.2 hervorgeht. Alle Materialparameter und die Geometrie der Wandscheibe können der nachfolgenden Tabelle entnommen werden. Die Berechnung erfolgt weiters unter Vernachlässigung des Eigengewichts.

Tab. 4.1: Berechnungsparameter

Wandhöhe	$\boldsymbol{H = 3{,}6}$
Wandlänge	$\boldsymbol{L = 6}$
Wandstärke	$\boldsymbol{D = 0{,}3}$
Elastizitätsmodul	$\boldsymbol{E = 1{,}6e + 9}$
Querdehnungszahl	$\boldsymbol{\nu = 0{,}2}$
Zulässige Druckspannung	$\boldsymbol{f_c = -500000}$
Zulässige Zugspannung	$\boldsymbol{f_t = 20000}$
Einwirkende Kraft Start	$\boldsymbol{F = 20000}$
Laststeigerung je Lastschritt	$\boldsymbol{\Delta F = 500}$

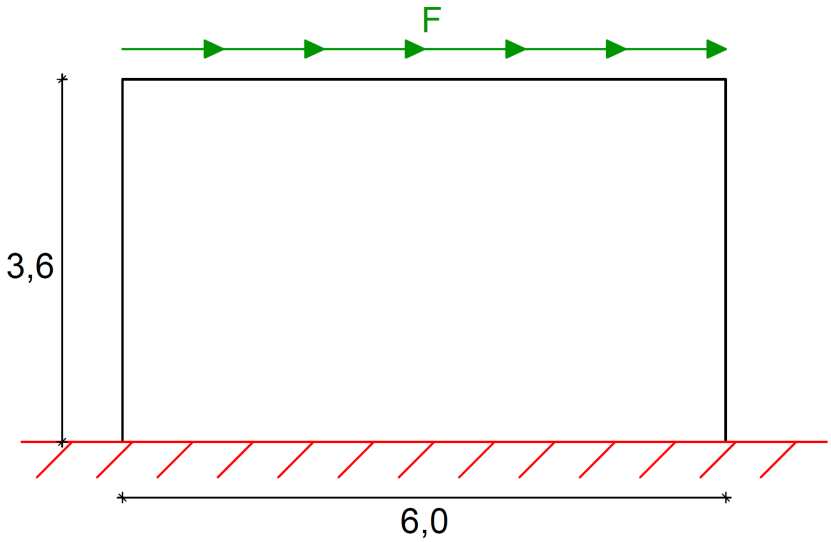

Abb. 4.1: Statisches System - Wandscheibe unter Schubbeanspruchung

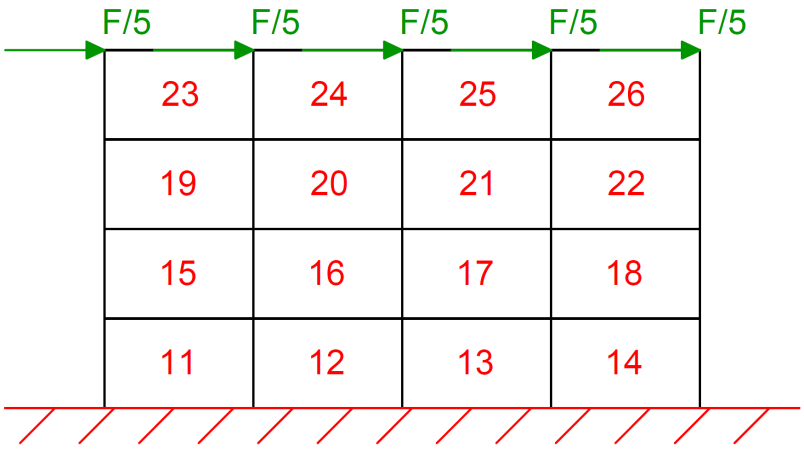

Abb. 4.2: Elementdiskretisierung - 16x $\mathbf{R_{Quad}}$

In einem ersten Schritt wird eine Last **F** anteilig auf alle Knoten des oberen Randes der Wandscheibe aufgebracht, welche anschließend mit jedem weiteren Iterationsschritt um die Größe **ΔF** gesteigert wird. Für jeden Berechnungsschritt werden die aufgebrachte Kraftgröße und die Verformung im oberen mittleren Knoten der Wandscheibe in einem Diagramm gegenübergestellt. Diese Iteration erfolgt so lange, bis in den Integrationspunkten der Elemente die maximal zulässigen Spannungen erreicht werden und diese Punkte somit ausfallen. Somit kommt es zu einem Abflachen der Kurve der Kraft-Verformungsbeziehung. Bei jeder Steigerung der Kraft um **ΔF** kommt es also zu jeweils immer größeren Verformungen. Weiters wird angemerkt, dass bei jedem neuen Lastniveau die Berechnung der Struktur so lange erfolgt, bis keine weiteren Integrationspunkte mehr ausfallen und sich somit ein Gleichgewicht im System einstellt. Das diesem Beispiel zugrunde liegende Skript file ist vollständig in Anhang D ersichtlich. Die folgende Abbildung 4.3 zeigt die Wandscheibe in unverformter Lage.

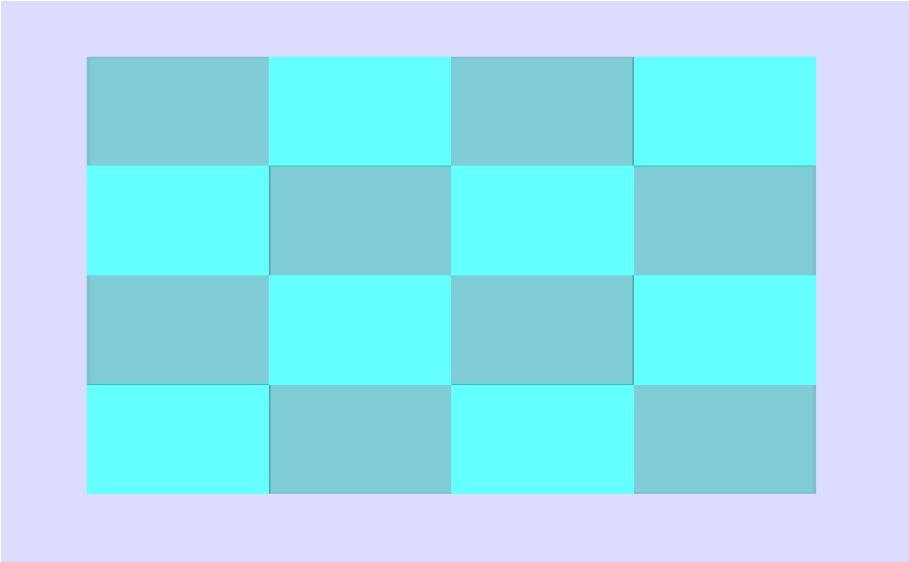

Abb. 4.3: Wandscheibe in unverformter Lage

Nach erreichen des halben endgültigen Lastniveaus sind noch keine Integrationspunkte ausgefallen. Die Berechnung erfolgte bis dahin linear elastisch. Dies geht aus dem Kraft-Verformungsdiagramm aus Abbildung 4.4 hervor. Die verformte Lage zu diesem Zeitpunkt ist in Abbildung 4.5 abgebildet.

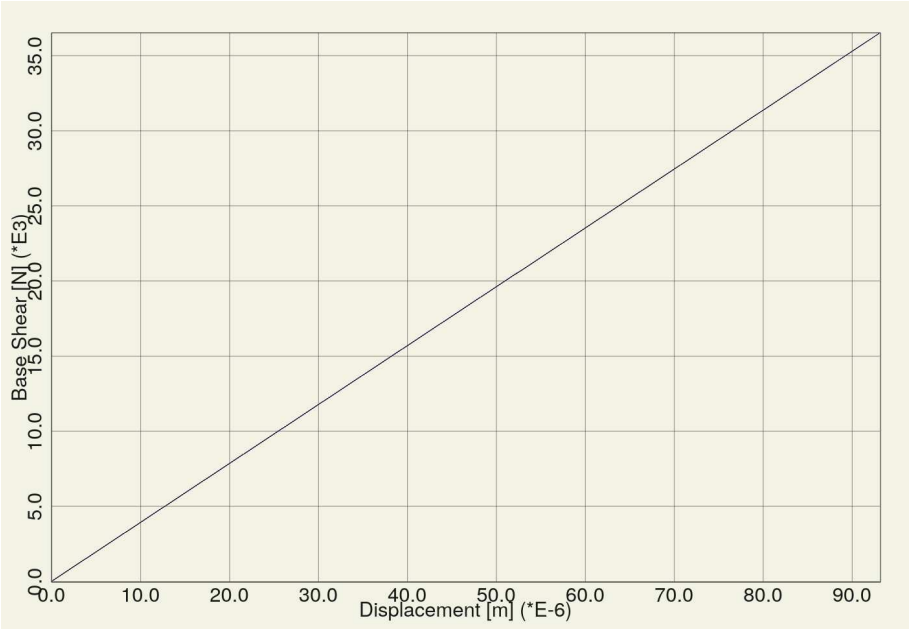

Abb. 4.4: Last- Verformungsdiagramm bis zur halben Versagenslast

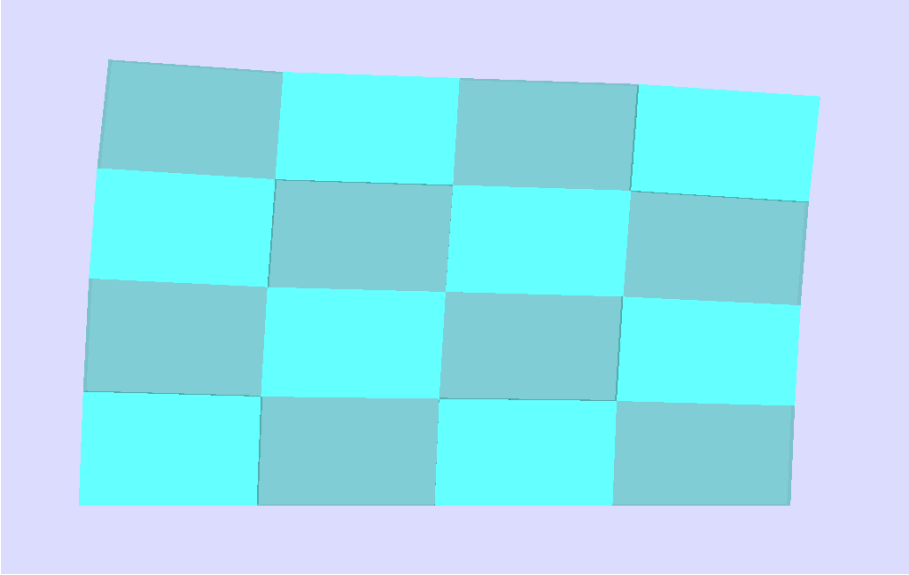

Abb. 4.5: Wandscheibe in linear elastisch verformter Lage bei halber Versagenslast

Bei weiterer Steigerung der Last kommt es in ersten Elementen zu einem Teilversagen und somit Ausfall von Integrationspunkten. Dies ist deutlich in der Simulation erkennbar und geht auch eindeutig aus dem Graph von Abbildung 4.6 hervor.

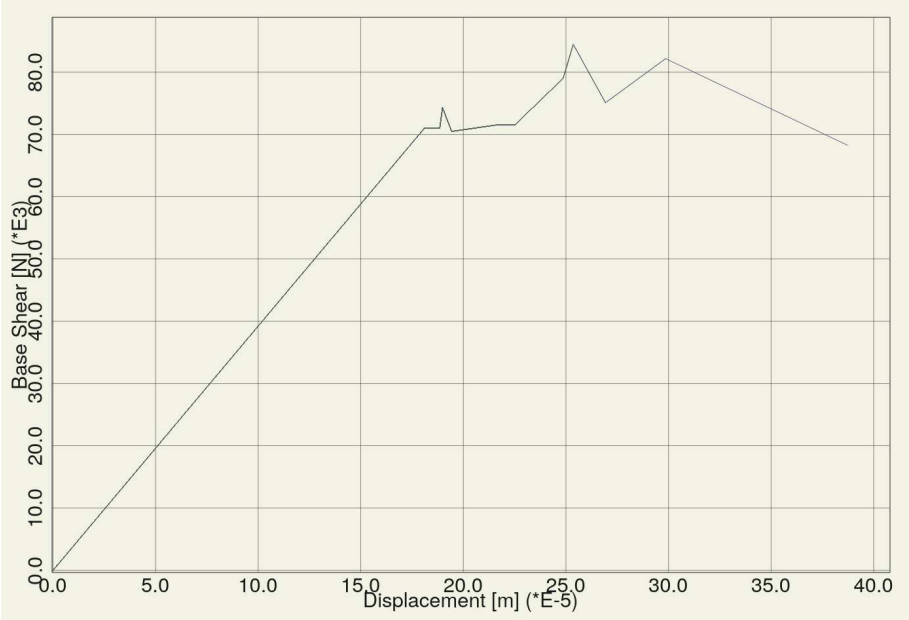

Abb. 4.6: Last- Verformungsdiagramm bis zum Versagen der Wandscheibe

In der untenstehenden Abbildung 4.7 ist bereits Versagen in dutzenden Integrationspunkten eingetreten. Es erfolgt eine Iteration innerhalb des Lastschrittes, um nach einem Gleichgewicht zu suchen. Fallen bei gleich bleibender Last trotzdem mit jedem Iterationsschritt weitere Integrationspunkte aus, sodass es schließlich zu einer Singularität der Steifigkeitsmatrix **K** kommt, ist das Ende der Simultion erreicht. Die Berechnung kann nicht mehr weiter fortgefahren werden. Es kommt zum Versagen der Struktur.

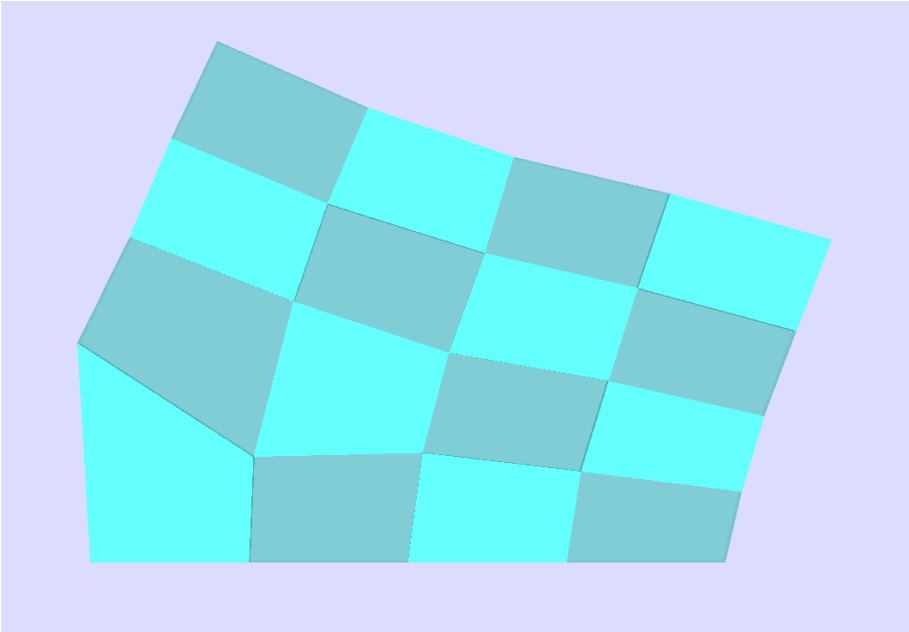

Abb. 4.7: Wandscheibe einen Berechnungsschritt vor dem Versagen

Wie in der nachfolgenden **Versagensmatrix** 4.6 dargestellt ist, existiert für alle Elemente für sämtliche Integraionspunkte ein Eintrag, der entweder den Wert **0** auweist, das heißt es kam

zu keinem Versagen in diesem Integrationspunkt, oder den Wert **1** besitzt, was bedeutet, dass dieser Integrationspunkt für weitere Berechnungen nicht in der Steifigkeitsmatrix des Elements berücksichtigt wird. Folgend ist der Zustand dieser Matrix unmittelbar vor dem Versagen der Struktur abgebildet, wobei jede Zeile ein Element repräsentiert, welches jeweils 18 Einträge für seine 18 Integrationspunkte besitzt. Die erste Spalte in der Gleichung unten, welche die Elementnummerierung enthält, dient nur informativen Zwecken und ist nicht Teil der tatsächlichen Versagensmatrix.

$$
\boldsymbol{Failure} = \begin{pmatrix}
\boldsymbol{Element\ 11} & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
\boldsymbol{Element\ 12} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\boldsymbol{Element\ 13} & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
\boldsymbol{Element\ 14} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 15} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 17} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 18} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 19} & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
\boldsymbol{Element\ 20} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 21} & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\boldsymbol{Element\ 22} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 24} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 25} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{Element\ 26} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \tag{4.6}
$$

Die erste Zeile der obigen Matrix enthält die Informationen zum Element links unten der Wandscheibe. Wie zu erwarten war, erfährt dieses die höchste Zugbeanspruchung und infolge dessen fallen in diesem Element die meisten Integrationspunkte aus. Mit Hilfe der in Abbildung 4.2 beschrifteten Elemente kann durch die **Versagensmatrix** rasch identifiziert werden, in welchen Elementen es zu einem Versagen kommt.

4.3 Gebäudemodell unter kraftgesteuerter Pushover-Analyse

Das Modell der einzelnen Wandscheibe des vorigen Kapitels wird nun erweitert. Es soll ein vereinfachtes, regelmäßiges Gebäudemodell untersucht werden. Die Materialparameter der Wände bleiben unverändert. Die Decken werden als Holzdecken, genauer gesagt als Dippelbaumdecken wie in der Einleitung kurz erwähnt, modelliert. Dabei wird ein Zusammenwirken der einzelnen Holzbalken aufgrund ihrer Verdübelung als Deckenplatte vorausgesetzt. Auf der sicheren Seite liegend wird der Elastizitätsmodul dieser Deckenplatte gemäß Holz quer zur Faserrichtung beansprucht gewählt. Da das Ziel dieser Arbeit die anschauliche Darstellung von Pushover-Kurven ist, wurde auf eine Definition eines orthotropen Materialmodells verzichtet. Für eine realitätsnähere Abbildung der Holzdecken würde sich auf jeden Fall die Implementierung eines solchen Modells anbieten. In der nachfolgenden Tabelle 4.2 sind die neu hinzukommenden Berechnungsparameter für dieses Beispiel zusammengefasst.

Nun wird eine horizontale Belastung auf das Modell aufgebracht. Wie in [14] vorgesehen, wird die Last in der Gebäudehöhe als dreiecksförmig verteilt angenommen, was in guter Näherung zur

Tab. 4.2: Berechnungsparameter

Gebäudehöhe	$H_{Gesamt} = 9{,}0$
Geschoßhöhe	$H_{Geschosst} = 3{,}0$
Gebäudelänge	$L = 8{,}0$
Gebäudebreite	$B = 6{,}0$
Wandstärke	$D_{wall} = 0{,}3$
Deckenstärke	$D_{slab} = 0{,}16$
Wand Elastizitätsmodul	$E_{wall} = 1{,}6e + 9$
Wand Querdehnungszahl	$\nu_{wall} = 0{,}2$
Wand Zulässige Druckspannung	$f_{c,wall} = -500000$
Wand Zulässige Zugspannung	$f_{t,wall} = 20000$
Decke Elastizitätsmodul	$E_{slab} = 3{,}7e + 8$
Decke Querdehnungszahl	$\nu_{slab} = 0{,}3$
Decke Zulässige Druckspannung	$f_{c,slab} = -360000$
Decke Zulässige Zugspannung	$f_{t,slab} = 550000$
Einwirkende Kraft Start	$F = 1000$
Laststeigerung je Lastschritt	$\Delta F = 1000$

ersten Eigenform der Struktur steht. Das untersuchte statische System inklusive der Belastung ist in Abbildung 4.8 dargestellt.

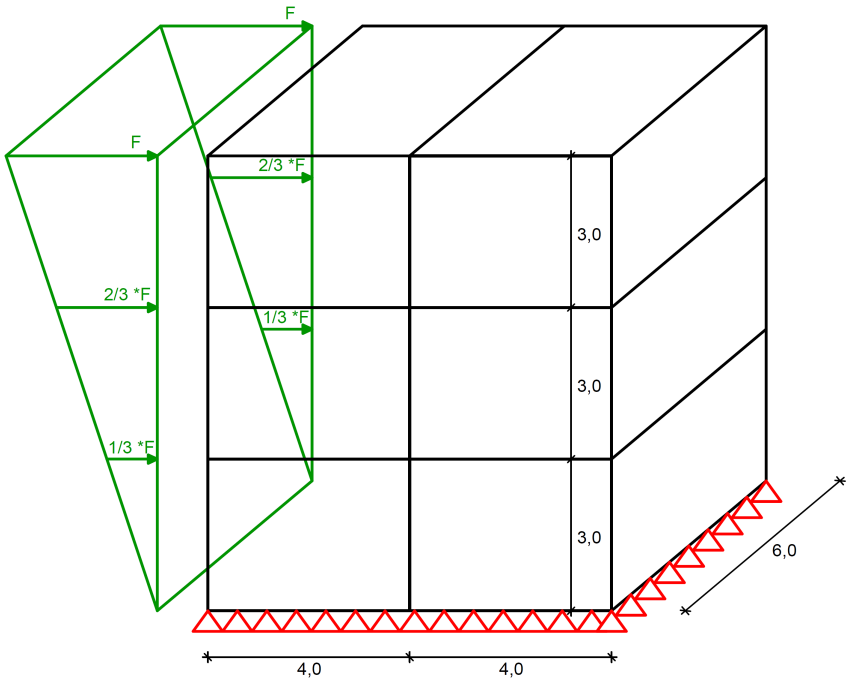

Abb. 4.8: Statisches System - Struktur unter dreieckig verteilter Horizontalbelastung

Dieses System wird nun durch finite Elemente diskretisiert, wie aus Abbildung 4.9 hervorgeht. Dabei erfolgt die Lagerung in den 6 Knoten der untersten Ebene gelenkig, das bedeutet lediglich die Verschiebungsfreiheitsgrade u_x, u_y und u_z sind gesperrt. In jedem Geschoß wirkt die horizontale Belastung als Einzellast in Gebäudelängsrichtung auf alle Knoten des Geschoßes ein.

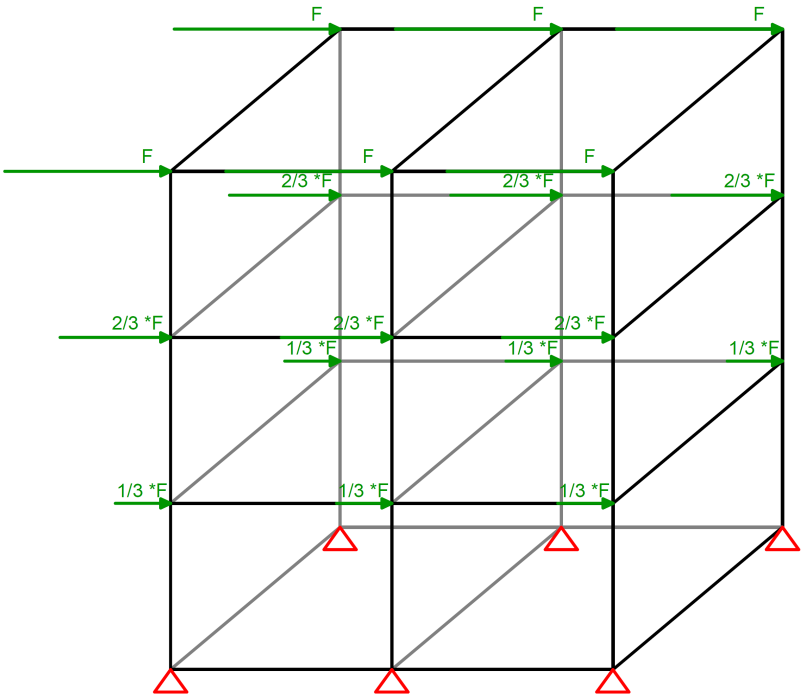

Abb. 4.9: Elementdiskretisierung - 24x $\mathbf{R_{Quad}}$

Die folgende Abbildung 4.10 zeigt die Struktur in unverformter Lage.

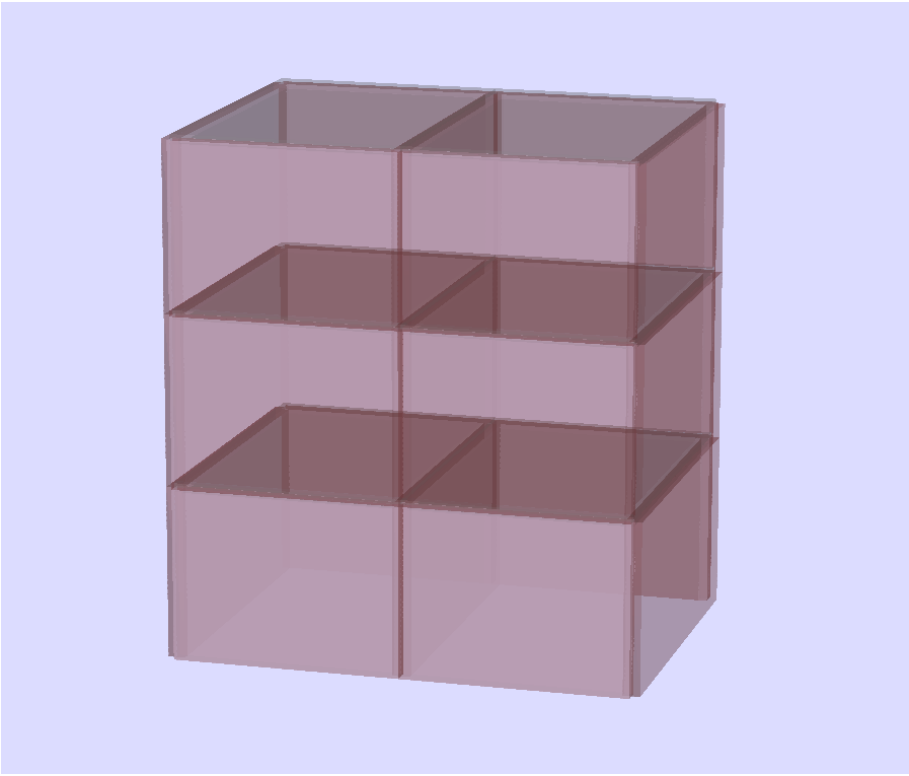

Abb. 4.10: Struktur in unverformter Lage

Anschließend erfolgt eine Laststeigerung, wobei innerhalb jedes Steigerungsschrittes die Berechnung mit gleichbleibender Last solange wiederholt wird, bis es zu keiner Änderung der globalen Resultierenden mehr kommt, sich also Gleichgewicht in der Gesamtstruktur eigestellt hat. Die Verschiebung in x-Richtung des vorderen linken Knotens in der Dachebene der Struktur wird nun in einem Diagramm der resultierenden Kraft in x-Richtung gegenübergestellt. Das Kraft-Verformungsdiagramm gemäß Abbildung 4.11 zeigt den Zustand der Kurve bis zum erreichen der halben Versagenslast. Die verformte Lage im gleichen Berechnungsschritt ist in Abbildung 4.12 abgebildet.

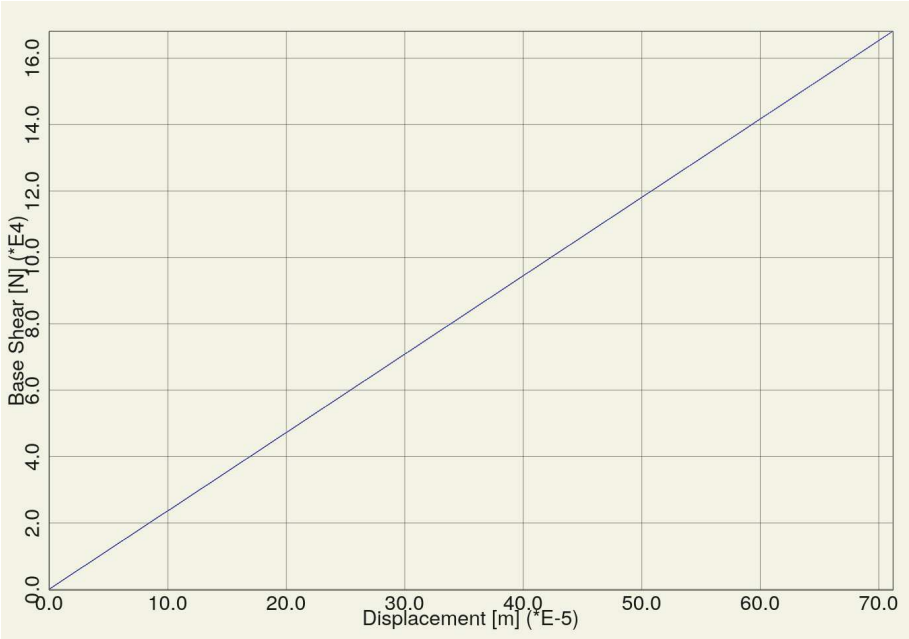

Abb. 4.11: Last- Verformungsdiagramm bis zur halben Versagenslast

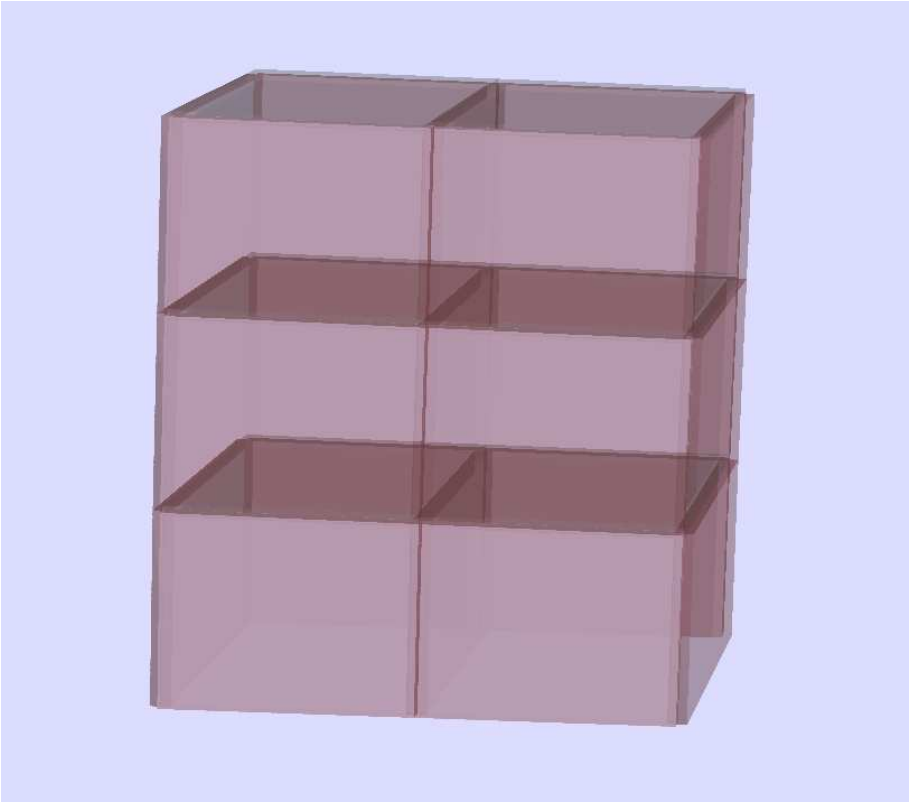

Abb. 4.12: Struktur in linear elastisch verformter Lage bei halber Versagenslast

Im nachfolgenden Diagramm 4.13 ist es bereits zu einer nichtlinearen Kraft-Verfprmungsbeziehung gekommen. Die Verformungen im Knotrollknoten wachsen überpropoertinal stark im Vergleich zur Laststeigerung an.

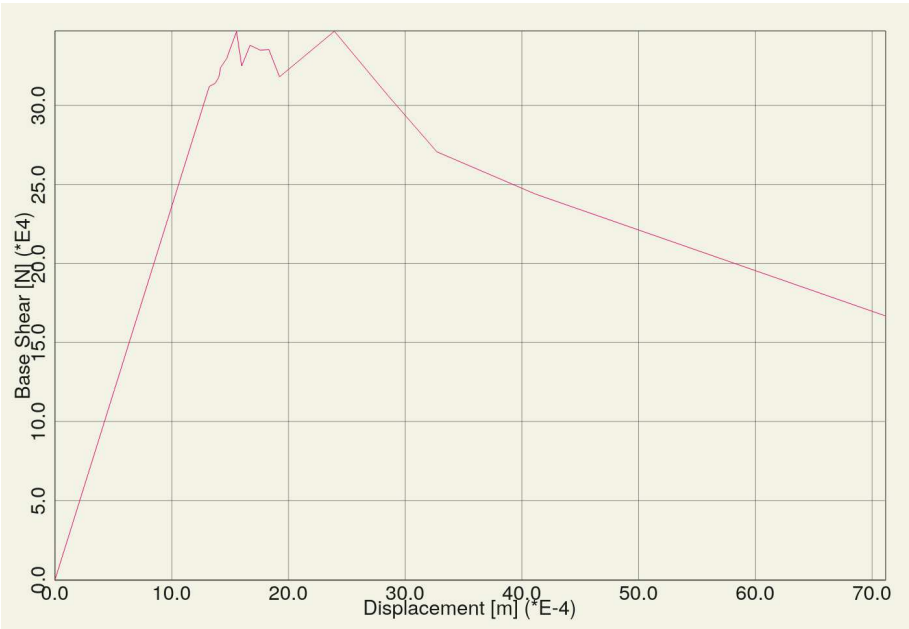

Abb. 4.13: Last- Verformungsdiagramm bis zum Versagen der Struktur

Schließlich kommt es in einigen Wandscheiben zu einer Überschreitung der zulässigen Spannungen und infolge dessen zum Herabsetzen der Steifigkeit der betroffenen Elemente. Dies geschieht solange, bis es zu einem Ausfall von einer zu großen Anzahl an Elementen gekommen ist, welcher schlussendlich zu einem Versagen der gesamten Struktur führt. Die untenstehende Abbildung 4.14 zeigt den Zustand der Struktur einen Berechnungsschritt, bevor ein globales Versagen eintritt.

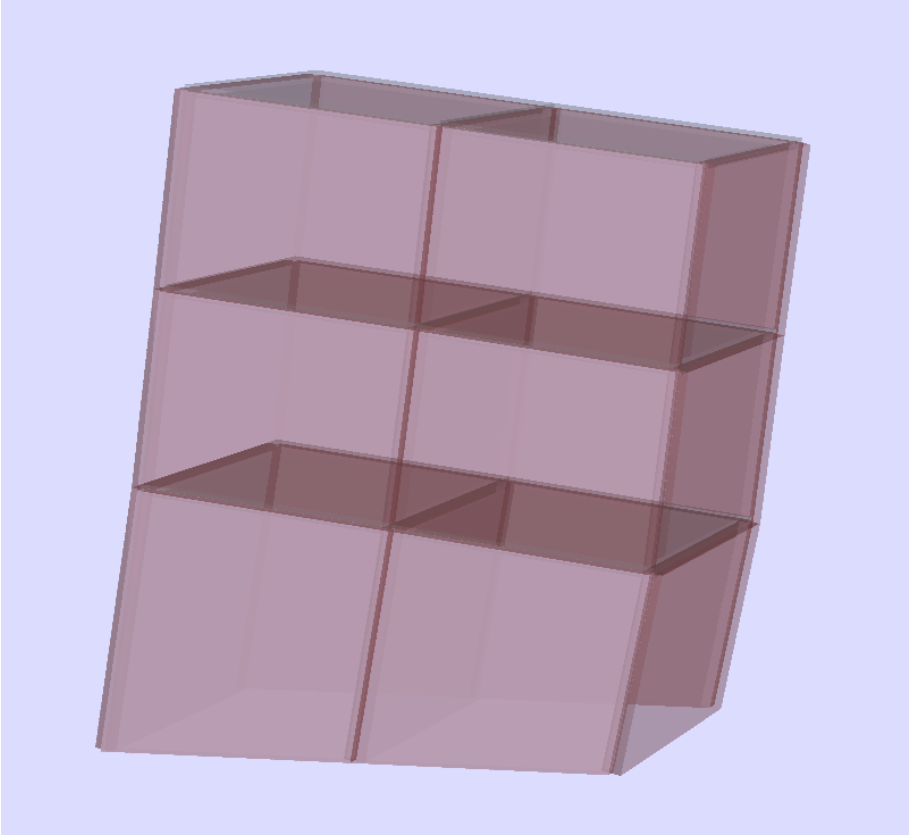

Abb. 4.14: Struktur einen Berechnungsschritt vor dem Versagen

Die obige Berechnung basiert auf zwei slangTNG files. Im file modell_test.tng wird das Strukturmodell definiert, im file push_modell.tng erfolgt die oben beschriebene Berechnung. Die vollständigen slangTNG files können dem Anhang D entnommen werden.

Kapitel 5

Theoretische Grundlagen

Dieses Kapitel gliedert sich in drei Abschnitte. Anfangs erfolgt die Einführung in später benötigte normative Begriffe und Parameter nach Eurocode 8. Diese sind Voraussetzung für das Verständis der später durchgeführten Berechnungen. Anschließend folgt eine weitere Vertiefung in die Materie der in der Einleitung dieser Arbeit bereits vorgestellten Pushover-Analyse. Schließlich wird erläutert, wie im nachfolgenden Kapitel 7, in welchem die Berechnungsbeispiele beschrieben werden, vorgegeangen wird, um Aussagen bezüglich der Tragfähigkeit der untersuchten Gebäudemodelle zu treffen.

5.1 Grundlagen und Begriffe nach Eurocode 8

In diesem Unterkapitel sollen die im folgenden verwendeten Begriffe, welche in die Ermittlung der Erdbebeneinwirkung eingehen, näher beschrieben werden. Gemäß Eurocode 8 und dem entprechenden nationalen Anwendungsdokument ist Österreich in mehrere Erdbebenzonen unterteilt. Diese Zoneneinteilung reicht von Zone 0 bis Zone 4, wobei in Zone 4 die Erdbebengefährdung am höchsten ist. Weiters ist eine Liste angegeben, in welcher jedem Bezirk eine Referenzbodenbeschleunigung a_{gR} zugeordnet ist. Am Beginn einer jeden Berechnung muss also der Standort des untersuchten Gebäudes definiert werden, um die normgemäße Bodenbeschleunigung festzulegen.

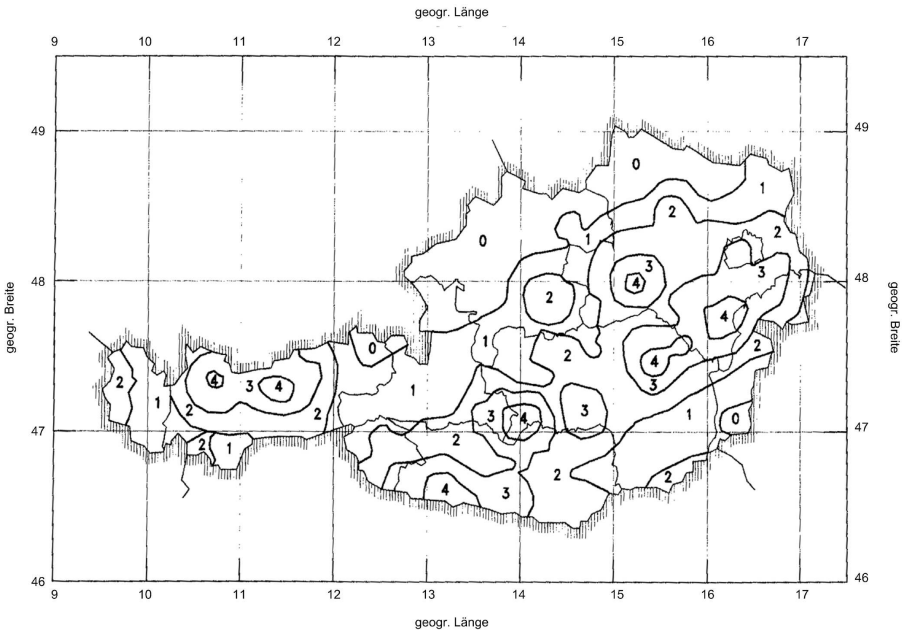

Abb. 5.1: Erdbebenzonen gemäß Eurocode 8 NA [13]

Neben dem Ort ist für die Größe der Einwirkung auch die Baugrundbeschaffenheit ausschlaggebend. Diese wird durch die sogenannten Baugrundklassen gemäß der folgenden Abbildung beschrieben.

Baugrund-klasse	Beschreibung des stratigraphischen Profils	Parameter		
		$v_{s,30}$ (m/s)	N_{SPT} (Schläge/30 cm)	c_u (kPa)
A	Fels oder andere felsähnliche geologische Formation, mit höchstens 5 m weicherem Material an der Oberfläche	> 800	—	—
B	Ablagerungen von sehr dichtem Sand, Kies oder sehr steifem Ton, mit einer Dicke von mindestens einigen zehn Metern, gekennzeichnet durch einen allmählichen Anstieg der mechanischen Eigenschaften mit der Tiefe	360–800	> 50	> 250
C	Tiefe Ablagerungen von dichtem oder mitteldichtem Sand, Kies oder steifem Ton, mit Dicken von einigen zehn bis mehreren hundert Metern	180–360	15–50	70–250
D	Ablagerungen von lockerem bis mitteldichtem kohäsionslosem Boden (mit oder ohne einige weiche kohäsive Schichten), oder von vorwiegend weichem bis steifem kohäsivem Boden	< 180	< 15	< 70
E	Ein Bodenprofil, bestehend aus einer Oberflächen-Alluvialschicht mit v_s-Werten nach C oder D und veränderlicher Dicke zwischen etwa 5 m und 20 m über steiferem Bodenmaterial mit v_s > 800 m/s			
S_1	Ablagerungen, bestehend aus (oder enthaltend) eine(r) mindestens 10 m dicke(n) Schicht weicher Tone oder Schluffe mit hohem Plastizitätsindex (PI > 40) und hohem Wassergehalt	< 100 (indikativ)	—	10–20
S_2	Ablagerungen von verflüssigbaren Böden, empfindlichen Tonen oder jedes andere Bodenprofil, das nicht in den Klassen A bis E oder S_1 enthalten ist			

Abb. 5.2: Baugrundklassen gemäß Eurocode 8 [14]

Ein weiterer Eingangsparameter ist die Bedeutungskategorie des Bauwerks. Diese ist abhängig von der Art des Gebäudes und der geplanten Nutzung. Beispielsweise haben Krankenhäuser, Schulen oder auch Kraftwerke eine höhere Bedeutungskategorie als Lagerhallen oder andere weniger von Menschen frequentierte oder zur Versorgungssicherheit des Staates relevante Bauwerke. Diese Bedeutungskategorien werden zwischen I und IV festgelegt, wobei IV die höchste Bedeutungskategorie besitzt.

Bedeutungs-kategorie	Bauwerke
I	Bauwerke von geringer Bedeutung für die öffentliche Sicherheit, z. B. landwirtschaftliche Bauten usw.
II	Gewöhnliche Bauwerke, die nicht unter die anderen Kategorien fallen
III	Bauwerke, deren Widerstand gegen Erdbeben wichtig ist im Hinblick auf die mit einem Einsturz verbundenen Folgen, z. B. Schulen, Versammlungsräume, kulturelle Einrichtungen usw.
IV	Bauwerke, deren Unversehrtheit während Erdbeben von höchster Wichtigkeit für den Schutz der Bevölkerung ist, z. B. Krankenhäuser, Feuerwachen, Kraftwerke usw.

Abb. 5.3: Baugrundklassen gemäß Eurocode 8 [14]

Basierend auf der festgelegten Bedeutungskategorie und der Baugrundklasse kann nun der Bedeutungsbeiwert γ_I mit Hilfe folgender Tabelle ermittelt werden.

Zonengruppe	Bedeutungskategorie γ_I			
	I	II	III	IV
0	0,8	1,0	1,0	1,0
1	0,8	1,0	1,0	1,0
2	0,8	1,0	1,1	1,2
3	0,8	1,0	1,2	1,4
4	0,8	1,0	1,4	1,4

Abb. 5.4: Bedeutungsbeiwert gemäß Eurocode 8 NA [13]

Nun wird die Referenzbodenbeschleunigung $\boldsymbol{a_{gR}}$ durch den Bedeutungsbeiwert $\boldsymbol{\gamma_I}$ modifiziert. Dadurch kann es sowohl zu einer Vergrößerung als auch zu einer Abminderung der anzusetzenden Bemessungsbodenbeschleunigung $\boldsymbol{a_g}$ kommen.

$$\boldsymbol{a_g = \gamma_I \cdot a_{gR}} \tag{5.1}$$

Zur Ermittlung der Kurve des elastischen Antwortspektrums sind weiters noch der Bodenparameter $\boldsymbol{S}$ und die Werte der Schwingungsdauer $\boldsymbol{T_B}$, $\boldsymbol{T_C}$ und $\boldsymbol{T_D}$ erforderlich, welche die Punkte zwischen den Abschnittsweise verschiedenen Kurven des Spektrums darstellen, erforderlich. Diese Parameter stehen alle in Abhängigkeit zur anfangs festgelegten Baugrundklasse. In Österreich ist laut dem nationalen Anwendungsdokument von Eurocode 8 ausschließlich die Spektralform Typ 1 anzuwenden, welche auf folgender Tabelle basiert.

Baugrundklasse	S	T_B (s)	T_C (s)	T_D (s)
A	1,0	0,15	0,4	2,0
B	1,2	0,15	0,5	2,0
C	1,15	0,20	0,6	2,0
D	1,35	0,20	0,8	2,0
E	1,4	0,15	0,5	2,0

Abb. 5.5: Eingangsparameter für Antwortspektrum gemäß Eurocode 8 [14]

Basierend auf den zuvor berechneten und definierten Parametern ist nun die Ermittlung des elastischen Antwortspektrum möglich. Dieses beschreibt die Reaktion von gedämpften Einmassenschwingern mit unterschiedlichen Schwingungsdauern (Perioden) $\boldsymbol{T}$ auf die Beschleunigungsanregung am Fußpunkt. Auf der Abszisse wird die Schwingdauer $\boldsymbol{T}$ aufgetragen. Jeder Schwingdauer ist auf der Ordniate die zugehörige Bemessungseinwirkung $\boldsymbol{S_d}$ zugeordnet. Folgend wird die Ermittlung der Ordinate $\boldsymbol{S_d}$ des Bemessungsspektrum angegeben. Abhängig davon, in welchem Bereich des Bemessungsspektrums sich die Periode $\boldsymbol{T_1}$ befindet, gelten die jeweiligen Beziehungen für $\boldsymbol{S_d}$, wobei für die viskose Dämpfung ein Wert von $\boldsymbol{\zeta = 5\%}$ angenommen wurde.

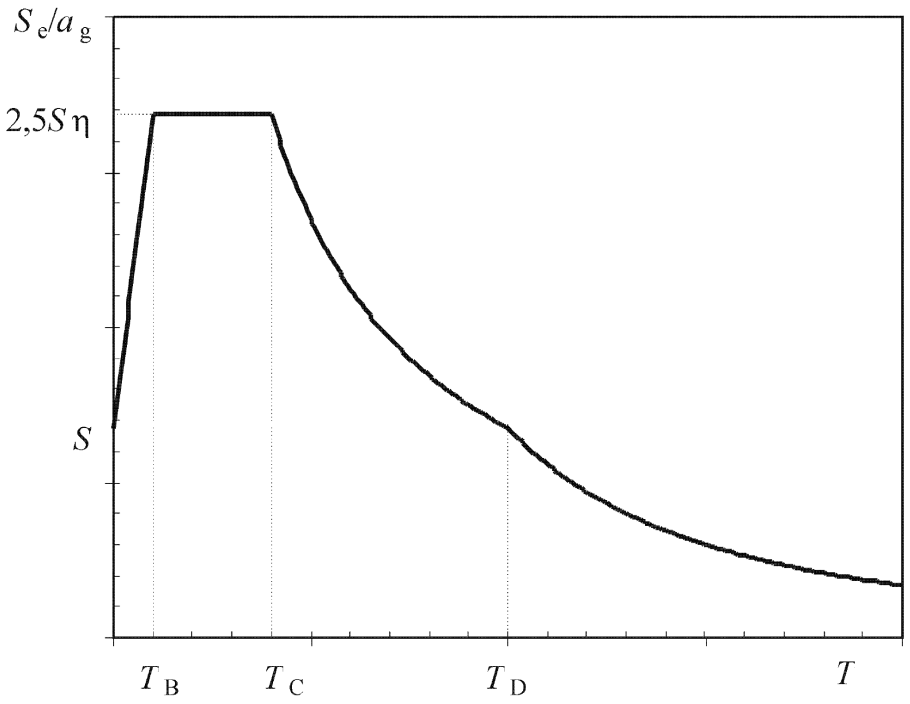

Abb. 5.6: Antwortspektrum gemäß Eurocode 8 [14]

$$
\begin{aligned}
\mathbf{0 \le T \le T_B:} \quad & \boldsymbol{S_d(T) = \frac{a_g \cdot S \cdot (1 + \frac{T}{T_B} \cdot (\eta \cdot 2{,}5 - 1))}{q}} \\
\mathbf{T_B \le T \le T_C:} \quad & \boldsymbol{S_d(T) = \frac{a_g \cdot S \cdot \eta \cdot 2{,}5}{q}} \\
\mathbf{T_C \le T \le T_D:} \quad & \boldsymbol{S_d(T) = \frac{a_g \cdot S \cdot \eta \cdot 2{,}5 \cdot \frac{T_C}{T}}{q}} \\
\mathbf{T_D \le T \le 4s:} \quad & \boldsymbol{S_d(T) = \frac{a_g \cdot S \cdot \eta \cdot 2{,}5 \cdot \frac{T_C \cdot T_D}{T^2}}{q}}
\end{aligned}
\tag{5.2}
$$

Der Verhaltensbeiwert, welcher nichtlineares, duktiles Materialverhalten berücksichtigen soll, wird zu $\boldsymbol{q = 1}$ angenommen, da sämtliche Nichtlinearitäten bei den folgenden Berechnungsbeispielen bereits auf der Widerstandsseite innerhalb des Materials berücksichtigt werden. Der von der Dämpfung abhängige Wert $\boldsymbol{\eta}$ ist wie folgt definiert.

$$\boldsymbol{\eta = \sqrt{\frac{10}{5 + \zeta}} \ge 0{,}55} \tag{5.3}$$

Das untersuchte Gebäude besitzt eine Vielzahl an Eigenschwingungsformen mit der jeweiligen Eigenschwingungsdauer $\boldsymbol{T_i}$, welche in der Regel nicht alle für die Bemessung relevant sind. Jede dieser Eigenformen hat einen unterschiedlich starken Einfluss auf die Größe der hervorgerufenen Belastungen des Bauwerks. Die Grundeigenform mit der zugehörigen Periode $\boldsymbol{T_1}$ hat in der Regel den größten Einfluss. Die Einflussgröße jeder einzelnen Eigenform wird durch die Einführung der effektiven modalen Massen $\boldsymbol{M_{k,eff}}$ anschaulich ausgedrückt. Nachfolgend werden die Berechnungsschritte zu deren Ermittlung erklärt. Zu allererst sind die Eigenfrequenzen $\boldsymbol{\omega_k}$

und Eigenformen φ_k des Gebäudemodells zu bestimmen. Dies geschieht durch die Lösung des Eigenwertproblems wie bereits in Kapitel 3 erklärt. Anschließend können die modalen Massen M_k für jede Eigenform k bestimmt werden.

$$M_k = \varphi_k^T \cdot M \cdot \varphi \tag{5.4}$$

Die Matrix M bezeichnet hierbei die Massenmatrix der Struktur. Weiters werden die Partizipationsfaktoren L_k berechnet.

$$L_k = \varphi_k^T \cdot M \cdot I \tag{5.5}$$

Der Vektor I in der obigen Gleichung bezeichnet einen Einheitsvektor, bei welchem alle Einträge den Wert 1 aufweisen. Mit Hilfe der ermittelten modalen Massen M_k und Partizipationsfaktoren L_k kann nun für jede Eigenform deren modale effektive Masse $M_{k,eff}$ bestimmt werden.

$$M_{k,eff} = \frac{L_k^2}{M_k} \tag{5.6}$$

Auf diese Weise kann rasch abgeschätzt werden, wie groß der Einfluss E_k der untersuchten Eigenform ist, in dem durch die Gesamtmasse des Gebäudes dividiert wird.

$$E_k = \frac{M_{k,eff}}{m} \tag{5.7}$$

Die Summe aller effektiven modalen Massen $M_{k,eff}$ entspricht hierbei immer der gesamten Masse m der Struktur. Bei regelmäßigen Gebäuden im Grundriss und Aufriss kann der Modalbeitrag der Grundeigenform $M_{1,eff}$ circa 85% erreichen.

5.2 Die Pushover-Analyse

Die Grundlagen der Pushover-Analyse werden in diesem Abschnitt nochmals zusammengefasst. Weiters wird auf die später angewendete Methode näher eingegangen und diese erläutert.

Wie bereits in der Einleitung beschrieben, bietet sich die Pushover-Analyse als Berechnungsmethode zum Nachweis der Erdbebensicherheit von Gebäuden an, da sie im Unterschied zu linearen Methoden, welche die einwirkenden Horizontallasten elastisch auf die einzelnen Bauteile aufteilen, dem Rechenmodell zusätzliche nichtlineare Tragreserven der Strukur zugänglich macht. Dies geschieht entweder durch das Einführen von platsichen Gelenken im finite Elemente Modell, oder das Definieren eines Versagensmodells für das Material, sodass Kraftumlagerungen möglich werden. Somit erhöht sich die Tragfähigkeit des untersuchten Strukturmodells, da im Fall der linear elastischen Berechnung bereits bei einer Spannungsüberschreitung in einem Bauteil von einem Versagen ausgegangen werden muss, währenddessen das System bei der Pushover-Analyse an Steifigkeit verliert und dadurch immer größere Verformungen bei gleichzeitig gleichmäßger Laststeigerung möglich werden. Diese Kraft-Verformungsbeziehung wird durch die sogenannte Kapazitätskurve gemäß Abbildung 5.7 ausgedrückt. Dabei wird die Verschiebung in einem anfangs definierten Kontrollknoten der resultierenden Horizontalbelastung (Querkraft) auf Höhe der Einspannebene des Bauwerks gegenübergestellt.

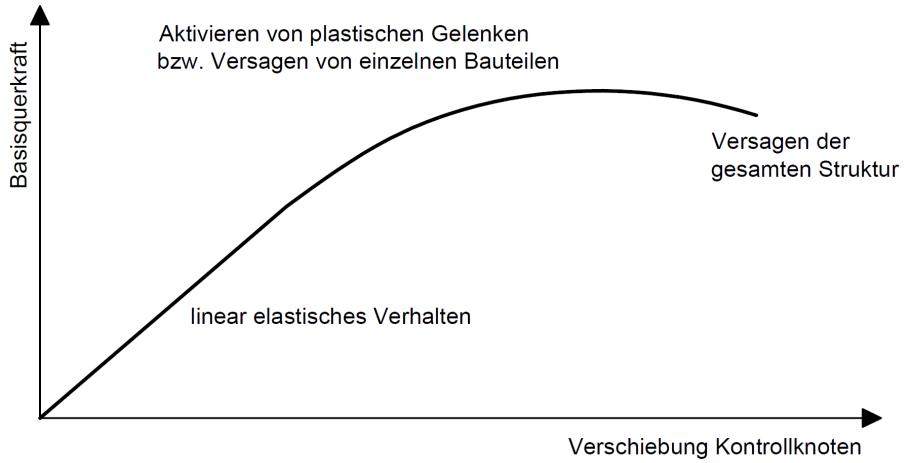

Abb. 5.7: Kapazitätskurve (Pushover-Kurve)

Anfangs werden die Vertikallasten auf das Modell aufgebracht und konstant gehalten. Anschließend erfolgt das Aufbringen der Horizontalbelastung, welche monoton gesteigert wird[11]. Diese Belastung wird solange gesteigert, bis die Kapazitätskurve nicht mehr wächst und eine zuvor definierte Zielverschiebung erreicht wird, oder es im Fall von berücksichtigtem Materialversagen zu einem Einsturz des Bauwerks kommt. Als Verformungsfigur wird in der Regel die Grundeigenform verwendet, welche näherungsweise einer dreiecksförmigen Verteilung der Horizontallasten über die Gebäudehöhe entspricht[11]. Die Horizontalbelastung kann bei vereinfachten analytischen Rechenmodellen, bei welchen die Massen der Struktur konzentriert in den einzelnen Geschoßen angenommen werden, in jedem Berechnungsschritt erhöht werden. Man spricht von einer kraftgesteuerten Berechnung wie in Abbildung 5.8 dargestellt.

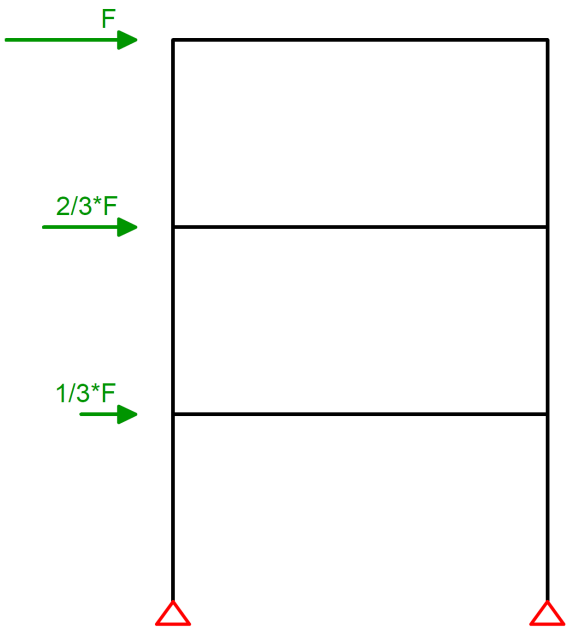

Abb. 5.8: Kraftgesteuerte Pushover-Analyse

Bei der Modellierung des Gebäudemodells mit einer Vielzahl an finiten Elementen erweist es sich als sinnvoll, keine äußeren Horizontalbelastungen an den Knoten aufzubringen. Vielmehr kann eine starke Vereinfachung der Berechnung erfolgen, indem die Pushover-Analyse weggesteuert durchgeführt wird. Dadurch müssen keine Näherungen für die genaue Aufteilung der Kräfte auf die Knoten gefunden werden. Die Vorgehensweise stellt sich wiefolgt dar: Es werden zu allererst die maßgebenden Eigenformen der untersuchten Struktur berechnet. Bei regelmäßigen Gebäuden stellen die ersten beiden Eigenformen in der Regel eine Verschiebung in Gebäudelängsrichtung und in Gebäudequerrichtung dar. Nun wird der Kontrollknoten defniert und die Eigenformen auf diesen Knoten normiert und um einen Faktor abgemindert. Anschließend wird das gesamte Modell in die Lage der untersuchten Eigenform versetzt, wie in Abbildung 5.9 schematisch gezeigt wird. Dadurch entstehen innerhalb der Struktur wirkende Kräfte durch die von außen zwangsweise aufgebrachte Verformung. Schließlich erfolgt mit jedem Berechnungsschritt eine Steigerung der Eigenform um einen Faktor, wobei während jedem Schritt die Kapazitätskurve gezeichnet wird. Dies führt wiederum ab einem gewissen Punkt zu Materialversagen innerhalb von einzelnen Bauteilen und somit Kraftumlagerungen. In den hier untersuchten Beispielen erfolgt die Skalierung der Eigenform so lange, bis es zu einem Versagen der Struktur kommt.

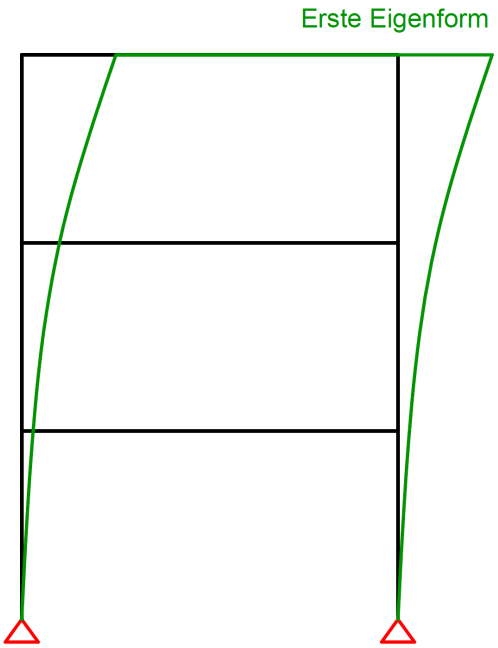

Abb. 5.9: Weggesteuerte Pushover-Analyse

5.3 Vorgehensweise im Rahmen der untersuchten Beispiele

Im Rahmen dieser Arbeit wird kein normgerechter Nachweis auf Basis der Pushover-Analyse präsentiert. Es werden somit weder Teilsicherheitsbeiwerte auf der Einwirkungsseite, noch auf der Materialwiderstandsseite berücksichtigt. Vielmehr sollen hier die Grundlagen präsentiert werden, auf deren Basis sich ein Nachweiskonzept erarbeiten ließe. Um dennoch einen Zusammenhang mit mit den in Eurocode 8[14] angegebenen standortabhängigen Parametern, welche die Erdbebeneinwirkung definieren, herzustellen, wird die im Zuge der Berechnung der später vorgestellten Beispiele ermittelte Querkraft auf Höhe der Einspannebene, im folgenden als Basisquerkraft $\boldsymbol{Q_b}$ bezeichnet, der laut Norm definierten Gesamterdbebenkraft $\boldsymbol{F_b}$ gegenübergestellt. Auf diese

Weise lassen sich sehr übersichtlich die Erdbebeneinwirkung und der Erdbebenwiderstand des Gebäudes gegenüberstellen. Folgend wird die Berechnung von $\boldsymbol{F_b}$ nach Eurocode 8 angegeben.

$$\boldsymbol{F_b = S_d(T_1) \cdot m \cdot \lambda} \tag{5.8}$$

In der obigen Gleichung bezeichnet $\boldsymbol{S_d(T_1)}$ die Ordinate des Bemessungsspektrums an der Stelle der Periode $\boldsymbol{T_1}$. Zur Abschätzung der Periode $\boldsymbol{T_1}$ der Grundeigenform sind in Eurocode 8 verschiedene Nähereungsformeln angegeben. Alternativ kann auch auf der sicheren Seite liegend der Wert für $\boldsymbol{S_d}$ am Plateau des Bemessungsspektrums unabhängig von der Periode angesetzt werden. Für die hier untersuchten Beispiele bietet sich die Ermittlung direkt aus dem finite Elemente Modell an. Da die Pushover-Analyse weggesteuert berechnet wird, wird für die Periode $\boldsymbol{T}$ in Gleichung 5.8 der Wert der der Pushover-Analyse zugrundegelegten Eigenform eingesetzt. Die Gesamtmasse des Gebäudes wird im folgenden $\boldsymbol{m}$ bezeichnet. Der Korrekturbeiwert gemäß Eurocode 8 wird bei $\boldsymbol{\lambda = 1{,}0}$ belassen. Dies bedeutet, dass die komplette Gesamterdbebenkraft der untersuchten Eigenform zugeschrieben wird.

Die Gesamterdbebenkraft $\boldsymbol{F_b}$ bezeichnet somit jene Kraft, die bei einer vorgegebenen Referenzbodenbeschleunigung $\boldsymbol{a_{gR}}$ an einem bestimmten Standort und Baugrund durch die Trägheit des Gebäudes an dessen Einspannhorizont wirkt.

Die aufnehmbare Basisquerkraft wird nun im Zuge der Pushover-Analyse des jeweils untersuchten Gebäudemodells ermittelt. Die horizontale Belastung wird so lange gesteigert, bis es zu einem Teilversagen einzelner Elemente kommt. Dadurch flacht die Kapazitätskurve ab und bei steigenden Verformungen kann die Basisquerkraft $\boldsymbol{Q_b}$ nicht weiter erhöht werden. Dieses Anwachsen der Verformungen ist solange möglich, bis es schließlich zu einem Versagen der Struktur kommt, da keine weiteren Lastumlagerungen mehr möglich sind. Der Wert der Basisquerkraft, welcher als aufnehmbare Belastung des Gebäudes der einwirkenden Gesamterdbebenkraft $\boldsymbol{F_b}$ gegenübergestellt wird, kann als maximal erreichter Wert der Kapazitätskurve interpretiert werden. Dieser wird im folgenden als Basisquerkraftwiderstand $\boldsymbol{Q_{b,R}}$ bezeichnet.

Kapitel 6

Untersuchung ausgewählter Gebäudemodelle

In Kapitel 2 wurde das finite Element **RQuad** vorgestellt und in Kapitel 3 anhand einfacher Berechnungsbeispiele dessen Zuverlässigkeit demonstriert. In Kapitel 4 wurde schließlich das Materialmodell **ElasticFailure** eingeführt. Auch hier wurde im Anschluss die grundlegende Funktionsweise mittels Beispielen anschaulich dargestellt. Somit wurde eine Basis aus einem Finiten Element und einem Materialmodell, welches in der Lage ist, Nichtlinearitäten abzubilden, gelegt, die für die in diesem Kapitel vorgesehenen Untersuchungen zwingend erforderlich sind. In Kapitel 5 folgte schließlich ein Überblick über die Pushover-Analyse, um die notwendigen Grundkenntnisse für die folgende Anwendung dieser zu vermitteln.

In diesem Kapitel wird schließlich das im Rahmen dieser Arbeit implementierte finite Element **RQuad** und das Materialmodell **ElasticFailure** an praktischen Beispielen angewendet. Es werden zwei unterschiedliche Gebäudemodelle, eines mit regelmäßigem Grundriss und ein unregelmäßiges mit einem L-förmigen Grundriss, untersucht. Dadurch soll zuerst anhand des regelmäßigen Gebäudes das zuvor beschriebene Nachweiskonzept praktisch angewendet werden. Anschließend wird dieses für das unregelmäßige Gebäude dahingehend erweitert, um bei der Pushover-Analyse auch Rotationseigenformen zu berücksichtigen.

Erdbebeneinwirkung

Die nachfolgende Tabelle 6.1 fasst die den Berechnungsbeispielen zugrundeliegenden Eingangsparameter der Erdbebeneinwirkung zusammen. Diese wurden dem Eurocode 8[14][13] entnommen.

Tab. 6.1: Erdbebenparameter

Standort	Wien
Erdbebenzone	3
Baugrundklasse	B
Bedeutungskategorie	II
Referenzbodenbeschleunigung a_{gR}	0,8
Dämpfung ζ	0,05

Mit Hilfe dieser Parameter kann der Bedeutungsbeiwert $\boldsymbol{\gamma_I = 1{,}0}$ bestimmt werden. Weiters ergeben sich der Bodenparameter und die Schwingungsdauern des Antwortspektrums wie folgt:

$$\begin{aligned} \boldsymbol{S &= 1{,}2} \\ \boldsymbol{T_B &= 0{,}15} \\ \boldsymbol{T_C &= 0{,}50} \\ \boldsymbol{T_D &= 2{,}00} \end{aligned} \tag{6.1}$$

Das linear elastische Antwortspektrum kann somit berechnet werden. Dieses dient später der Einordnung der Perioden $\boldsymbol{T}$ der untersuchten Gebäudemodelle zur Ermittlung der Ordinate des Bemessungsspektrum $\boldsymbol{S_d}$.

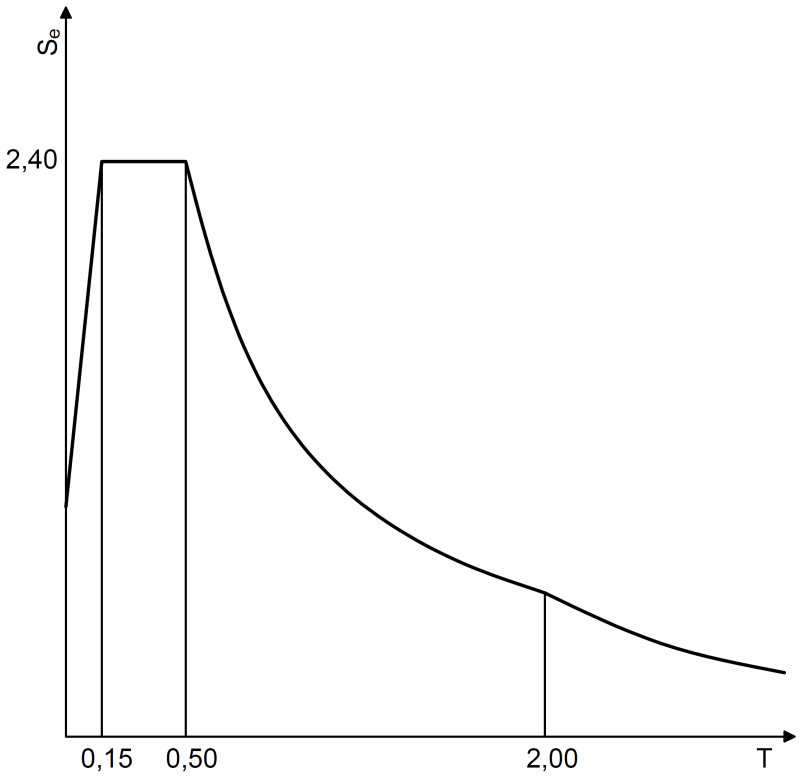

Abb. 6.1: Linear elastisches Antwortspektrum für die definierten Erdbebenparameter

Gebäudemodelle

Sowohl das regelmäßige als auch das unregelmäßige L-förmige Gebäudemodell bestehen aus drei Stockwerken. Sämtliche Modellparameter, welche für beide untersuchten Gebäude identisch sind, werden in der unten stehenden Tabelle 6.2 angeführt. Im Rahmen der Pushover-Analyse ist es üblich, mit den Mittelwerten der Materialfestigkeiten zu rechnen. Die zulässige Zugspannung der Wandscheiben wurde im Vergleich zu den vorangegangenen Beispielen erhöht, um ein höheres Verformungsvermögen des Bauwerks zuzulassen. Es wird an dieser Stelle nochmals darauf hingewiesen, dass es sich beim implementierten Materialmodell **ElasticFailure** um ein isotropes Materialgesetz handelt. Dies ist nur in begrenztem Ausmaß in der Lage, das komplexe Materialverhalten von Mauerwerk abzubilden. Die Entwicklung eines komplexeren Materialmodells, welches alle Versagensmuster von Mauerwerk berücksichtigen würde, übersteigt den Umfang dieser Arbeit, ist jedoch auf jeden Fall für weitere Forschungsarbeiten von Interesse.

Die Materialparameter für das Mauerwerk wurden [8] entnommen.

6.1 Regelmäßiges Gebäudemodell

Das symmetrische Gebäudemodell besteht aus beidseitig jeweils drei Wandpfeilern. Diese bilden zusammen den Haupttragwiderstand des Bauwerks gegen Erdbebeneinwirkungen in Gebäudelängsrichtung. Orthogonal dazu finden sich die beiden außen liegenden Feuermauern und mittig

Tab. 6.2: Modellparameter

Gebäudehöhe	$H_{Gesamt} = 11{,}7$
Geschoßhöhe	$H_{Geschosst} = 3{,}9$
Gebäudelänge	$L = 10{,}0$
Gebäudebreite	$B = 6{,}0$
Wandstärke Längswand	$D_{wall} = 0{,}45$
Wandstärke Querwand	$D_{wall} = 0{,}30$
Deckenstärke	$D_{slab} = 0{,}16$
Wand Elastizitätsmodul	$E_{wall} = 1{,}6e + 9$
Wand Querdehnungszahl	$\nu_{wall} = 0{,}2$
Wand Zulässige Druckspannung	$f_{c,wall} = -500000$
Wand Zulässige Zugspannung	$f_{t,wall} = 40000$
Decke Elastizitätsmodul	$E_{slab} = 3{,}7e + 8$
Decke Querdehnungszahl	$\nu_{slab} = 0{,}3$
Decke Zulässige Druckspannung	$f_{c,slab} = -360000$
Decke Zulässige Zugspannung	$f_{t,slab} = 550000$

zwei Schubwände, welche in der Tabelle im vorigen Abschnitt auch als Querwände bezeichnet wurden. Die genauen Abmessungen der einzelnen Elemente können dem file modell_sym.tng in Anhang E entnommen werden. Der Wahl der Lage des Kontrollknotens kommt bei der hier verwendeten Methode nicht die gleich wichtige Bedeutung zu, wie beim gewöhnlich verwendeten Nachweisformat, wo das Ende der Kapazitätskurve durch eine maximal zulässige Grenzverschiebung definiert wird. Durch das hier verwendete Versagensmodell kommt es zwangsläufig an einem gewissen Last- beziehungsweise Verformungsniveau der Struktur zu einem Versagen. Die Kapazitätskurve endet somit an dieser Stelle. Der Kontrollknoten (siehe Abbildung 6.2) dient somit vielmehr zur Überprüfung der Verformungen an eben diesem Punkt während der Pushover-Ananlyse.

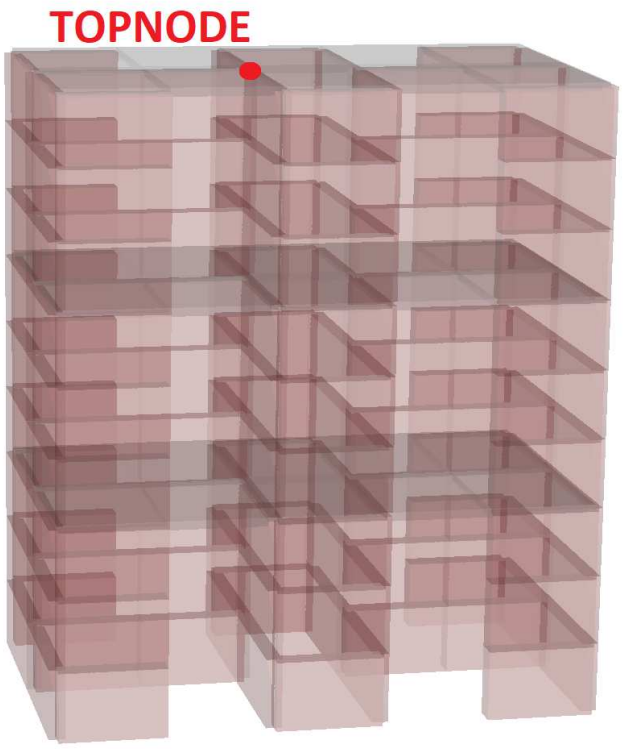

Abb. 6.2: Regelmäßiges Gebäudemodell in unverformter Lage - Position des Kontrollknotens

Die Wandnummerierung gemäß Abbildung 6.3 bezieht sich auf die Wände im Erdgeschoß. Übereinander liegende Wandscheiben unterscheiden sich durch die erste Ziffer der Nummerierung. Die untersten Wände des Gebäudes haben folglich die Ziffer **0** am Beginn.

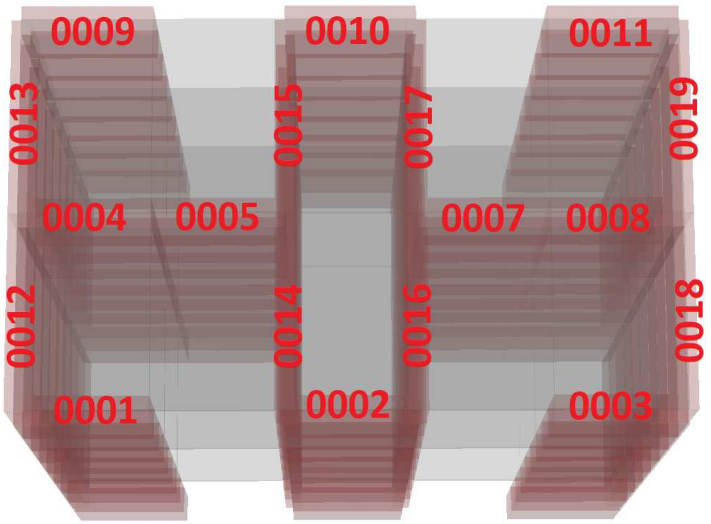

Abb. 6.3: Nummerierung der Wandscheiben

Das Gebäudemodell wird nun in die Richtungen beider Hauptsymmetrieachsen einer weggesteuerten Pushover-Analyse unterzogen. Dazu wird eine maximal zulässige Verschiebung definiert, welche mit 4‰der Gebäudehöhe, also $\boldsymbol{u_{max} = 0{,}004 \cdot H_{Gesamt} = 0{,}0468m}$, festgelegt wird. Dieser Wert wird in der Norm[15] als maximal zulässige gegenseitige Stockwerksverschiebung angenommen. Da die Decken, welche die einzelnen Wandpfeiler verbinden, nur eine geringe Schubfestigkeit aufweisen, kann als Stockwerk in diesem Falle die Gebäudehöhe beziehungsweise die Höhe der einzelnen Wandpfeiler, betrachtet werden[8]. Nun erfolgt die Berechnung einiger Eigenformen des Gebäudes. Auf Grund der den Eigenformen zugeordneten modalen Massen kann abgeschätzt werden, wie groß der jeweilige Einfluss dieser auf die Erdbebensicherheit ist. Die Eigenformen können graphisch ausgegeben werden um so rasch zu identifizieren, ob es sich um Translations- beziehungsweise Rotationsformen handelt. Für das regelmäßige Gebäudemodell ist es zweckmäßig, eine Pushover-Analyse für die ersten beiden Eigenformen, welche eine Verschiebung in die Gebäudehauptachsen beschreiben, durchzuführen.

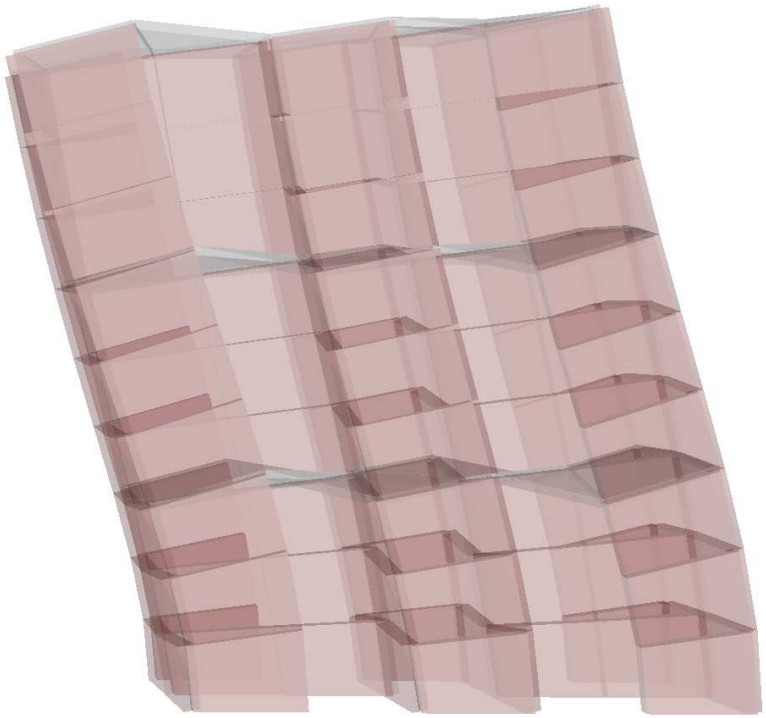

Abb. 6.4: Erste Eigenform des regelmäßigen Gebäudemodells

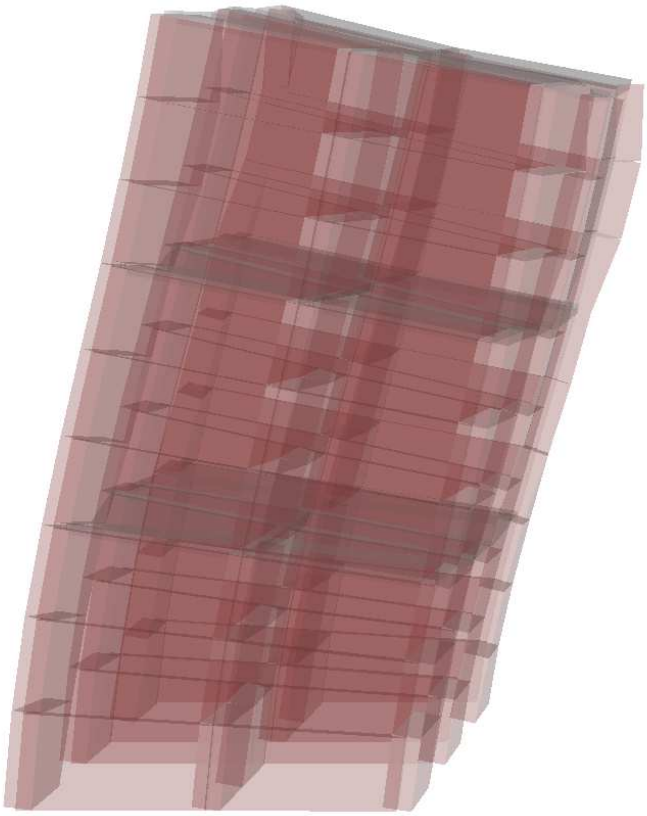

Abb. 6.5: Zweite Eigenform des regelmäßigen Gebäudemodells

Zu allererst wird das Modell unter die Einwirkung seiner Eigenlast versetzt. Nun wird die jeweilig untersuchte Eigenform auf den maximalen Koeffizienten des Eigenvektors normiert, sodass dieser den Wert 1 besitzt. Anschließend wird der Vektor mit der oben erwähnten zulässigen Verschiebung skaliert. Schließlich soll eine Iteration erfolgen, wobei eingangs die skalierte Eigenform durch die Anzahl der Iterationsschritte $\boldsymbol{N}$ geteilt wird. Anschließend werden in jedem Iterationsschritt die Verformungen schrittweise erhöht, bis entweder nach $\boldsymbol{N}$ Berechnungsschritten die zulässige Grenzverschiebung wieder erreicht wird, oder es aber bereits zuvor zu einem Versagen des Bauwerks kommt. Innerhalb jedes Iterationsschritt wird die Steifigkeitsmatrix $\boldsymbol{K}$ des Modells auf Singularität getestet. Ist diese singulär, kann also kein Gleichgewicht mehr gefunden werden, da eine zu große Anzahl an Elementen bereits versagt haben und keine Lastumlagerungen innerhalb der Struktur mehr möglich sind. Die Iteration wird dadurch abgebrochen und dieser Punkt markiert den letzten Punkt auf der Kapazitätskurve, bevor Versagen eintritt.

Für die Pushover-Analyse in Gebäudelängsrichtung (erste Eigenform) wird das Versagensbild der Struktur näher beschrieben. In der Abbildung 6.6 ist das Versagen des Bauwerks gut erkennbar. Da die aufnehmbaren Zugspannungen des definierten Material vielfach geringer als die aufnehmbaren Druckspannungen sind, kommt es somit im Erdgeschoß zu einem immer weiter fortlaufenden Ausfall der auf Zug beanspruchten Wandscheiben. Dieser Vorgang setzt sich solange fort, bis die äußersten Wandscheiben, welche noch auf Druck beansprucht waren, auch an einem Rand einen Zugausfall erleiden. Im nächsten Berechnungsschritt kommt es schließlich zu einem Versagen der gesamten Struktur. Im Unterschied zur Figur der ersten Eigenform sind

in der entsprechenden Versagensfigur deutliche Unstetigkeiten bei den Wänden im Erdgeschoß erkennbar, welche auf das Versagen einer großen Zahl von Integrationspunkten innerhalb der betroffenen Elemente schließen lassen.

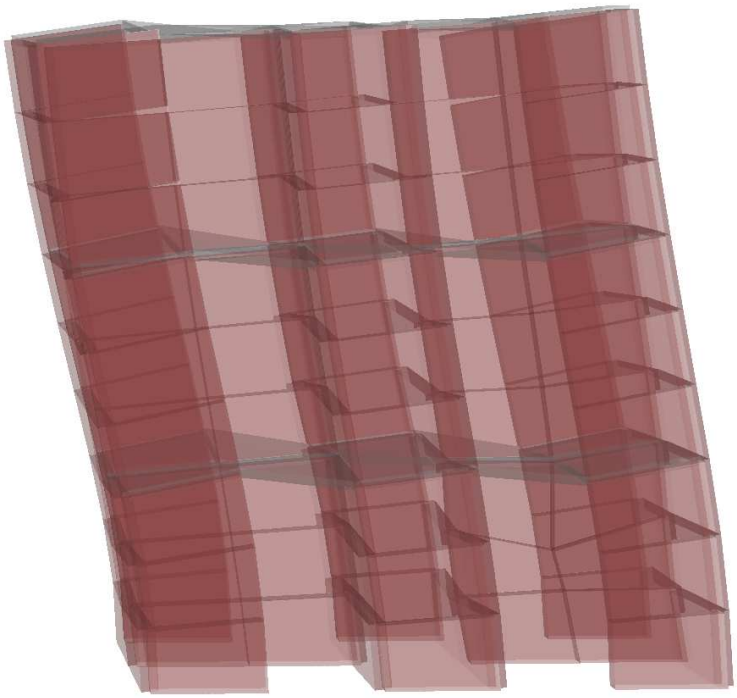

Abb. 6.6: Zustand unmittelbar vor Versagen - Pushover-Analyse gemäß erster Eigenform

Dies ist sehr deutlich in der **Versagensmatrix** erkennbar. Der hier abgebildete Ausschnitt dieser Matrix zeigt den Versagenszustand aller Wandscheiben im Erdgeschoß nach Überschreitung der zulässigen Zugbeanspruchung. Die Elemente 0001-0011 bezeichnen die Wandscheiben in Gebäudelängsrichtung, die Elemente 0012-0019 jene Wandscheiben orthogonal dazu.

$$
\boldsymbol{Failure} = \begin{pmatrix}
\textit{Element} & 0001 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
\textit{Element} & 0002 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\textit{Element} & 0003 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\textit{Element} & 0004 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\textit{Element} & 0005 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\textit{Element} & 0007 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\textit{Element} & 0008 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
\textit{Element} & 0009 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\textit{Element} & 0010 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
\textit{Element} & 0011 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
\textit{Element} & 0012 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\textit{Element} & 0013 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\textit{Element} & 0014 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\textit{Element} & 0015 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\textit{Element} & 0016 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\textit{Element} & 0017 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\textit{Element} & 0018 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
\textit{Element} & 0019 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{6.2}
$$

Die Kapazitätskurve für die erste Eigenform stellt nun wie beschrieben die Verschiebung des Kontrollknotens in Gebäudelängsrichtung der Basisquerkraft in ebenjene Richtung gegenüber.

Abb. 6.7: Kapazitätskurve - Pushover-Analyse gemäß erster Eigenform

Für die Pushover-Analyse in Richtung der zweiten Eigenform wird ebenfalls die entsprechende Abbildung zum Versagen sowie die Kapazitätskurve angegeben.

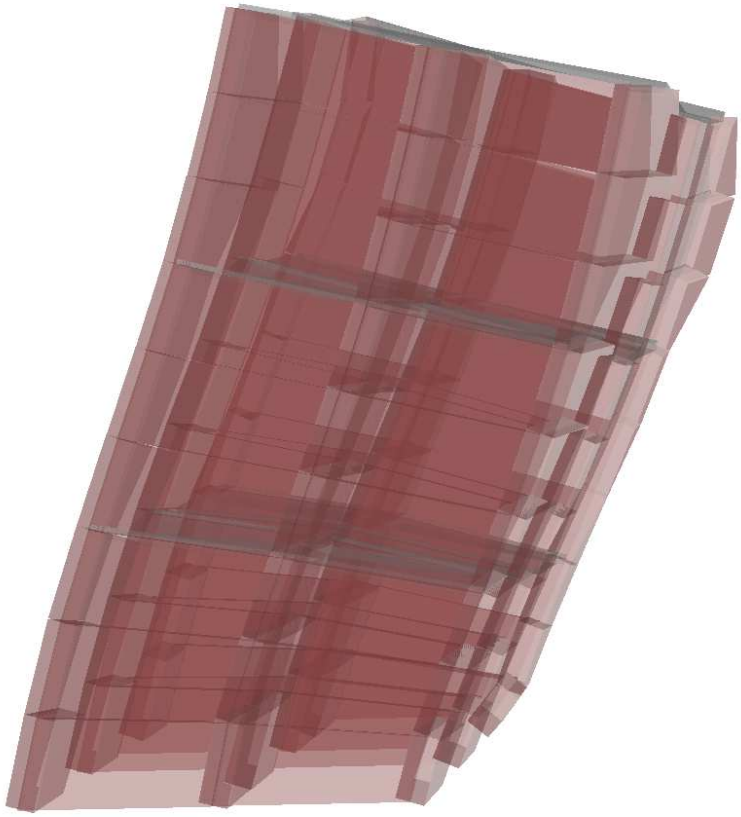

Abb. 6.8: Zustand unmittelbar vor Versagen - Pushover-Analyse gemäß zweiter Eigenform

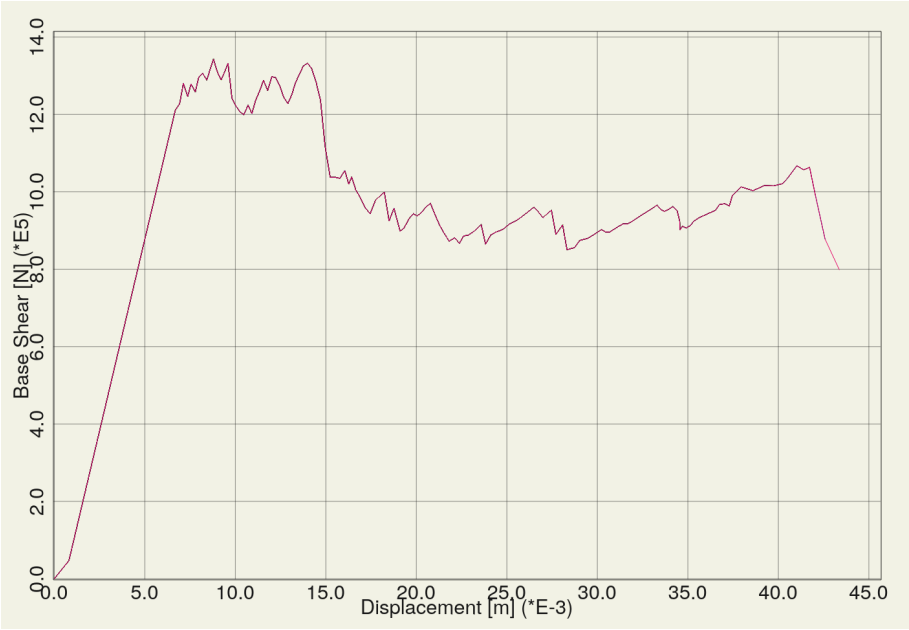

Abb. 6.9: Kapazitätskurve - Pushover-Analyse gemäß zweiter Eigenform

Die genaue numerische Vorgehensweise ist im file push_weg.tng in Anhang E ersichtlich.

Aus der Kapazitätskurve kann nun der Basisquerkraftwiderstand $\boldsymbol{Q_{b,R}}$ für beide untersuchten Richtungen abgelesen werden. Dieser entspricht wie in Kapitel 5 erwähnt dem maximal erreichtem Wert der jeweiligen Kurve. Der Index 1 beziehungsweise 2 steht in den folgenden Gleichungen für die Wirkung der Kraft in Gebäudelängs- beziehungsweise Gebäudequerrichtung.

$$\begin{aligned} \boldsymbol{Q_{b,1,R}} &= \boldsymbol{1358248 = 1358kN} \\ \boldsymbol{Q_{b,2,R}} &= \boldsymbol{1332437 = 1332kN} \end{aligned} \tag{6.3}$$

Im Anschluss soll nun die einwirkende Erdbebenbeanspruchung in Form der Gesamterdbebenkraft $\boldsymbol{F_b}$ ermittelt werden, um die einwirkende Kraft mit dem aufnehmbaren Widerstand zu vergleichen und eine Aussage bezüglich der Erdbebensicherheit der Struktur treffen zu können. Die Perioden ergeben sich aus der numerischen Berechnung.

$$\begin{aligned} \boldsymbol{T_1} &= \boldsymbol{0{,}186s} \\ \boldsymbol{T_2} &= \boldsymbol{0{,}176s} \end{aligned} \tag{6.4}$$

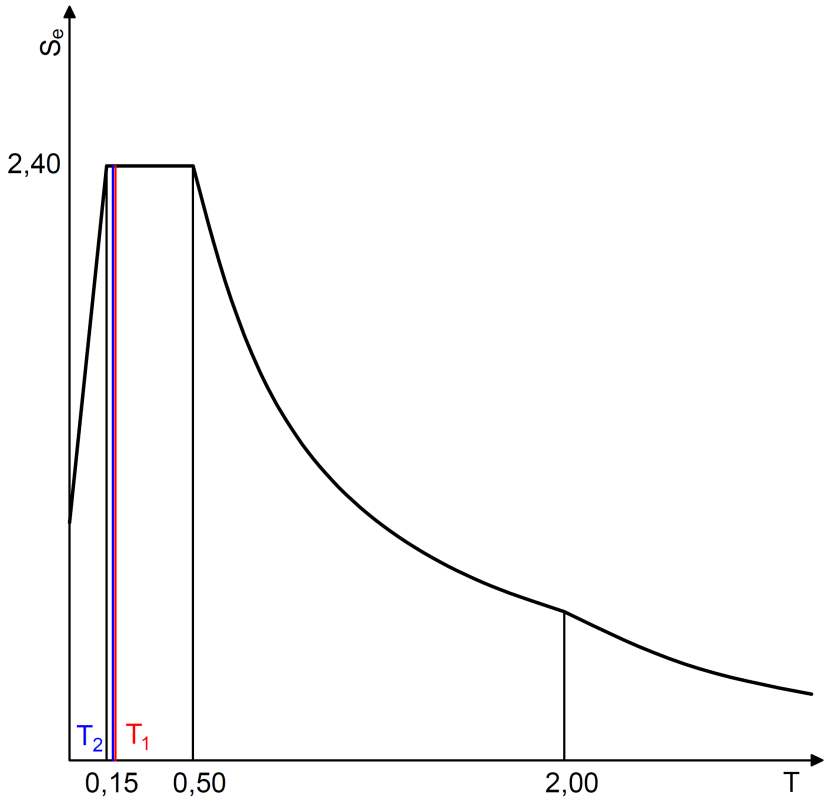

Abb. 6.10: Perioden des regelmäßigen Gebäudemodells im linear elastischen Antwortspektrum

Auch die Gesamtmasse des Bauwerks wurde numerisch ermittelt.

$$\boldsymbol{m = 315278kg} \tag{6.5}$$

$$\begin{aligned} F_{b,1} &= S_d(T_1) \cdot m = 2{,}4 \cdot 315278 = 756667 = 757kN \\ F_{b,2} &= S_d(T_2) \cdot m = 2{,}4 \cdot 315278 = 756667 = 757kN \end{aligned} \tag{6.6}$$

Nun kann der Ausnutzungsgrad der Gesamterdbebensicherheit wiefolgt angegeben werden.

$$\eta = max(\frac{F_{b,i}}{Q_{b,R,i}}) = max(\frac{757}{1358}, \frac{757}{1332}) = 0{,}57 \leq 1{,}00 \tag{6.7}$$

6.2 Unregelmäßiges Gebäudemodell

In diesem Kapitel wird das zuvor beschriebene Gebäudemodell derart erweitert, sodass es einen L-förmigen Grundriss erhält. Somit handelt es sich gemäß den in Eurocode 8[14] getroffenen Annahmen um ein unregelmäßiges Bauwerk. Der Kontrollknoten wird gemäß folgender Abbildung nahe dem Massenmittelpunkt des Bauwerks gewählt.

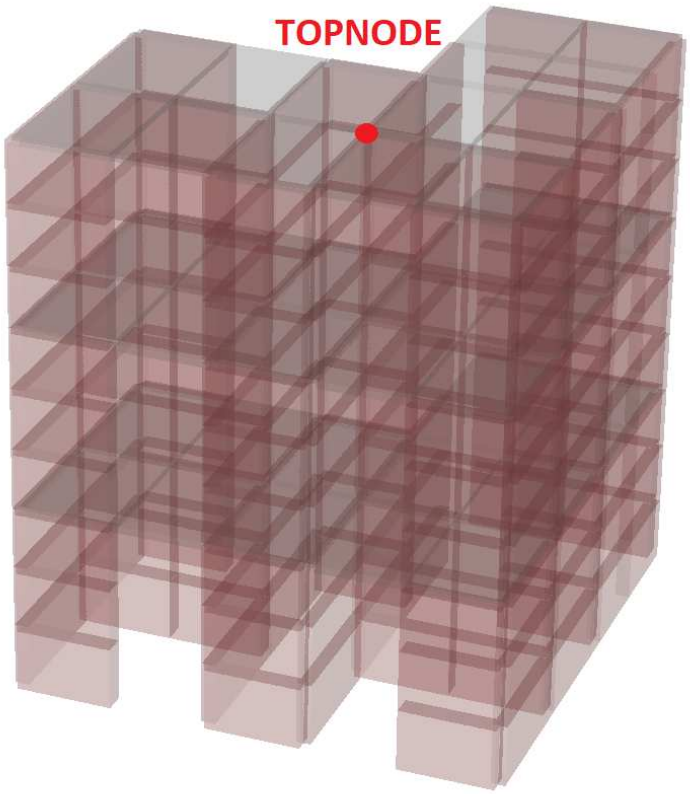

Abb. 6.11: Unregelmäßiges Gebäudemodell in unverformter Lage - Position des Kontrollknotens

Die nachfolgenden Abbildungen zeigen die ersten vier Eigenformen des Gebäudemodells. Bei Eigenform 1 und Eigenform 2 sind die Translationsanteile in x-Richtung beziehungsweise y-Richtung noch erkennbar, ab der dritten Eigenform sind starke Rotationsanteile ersichtlich.

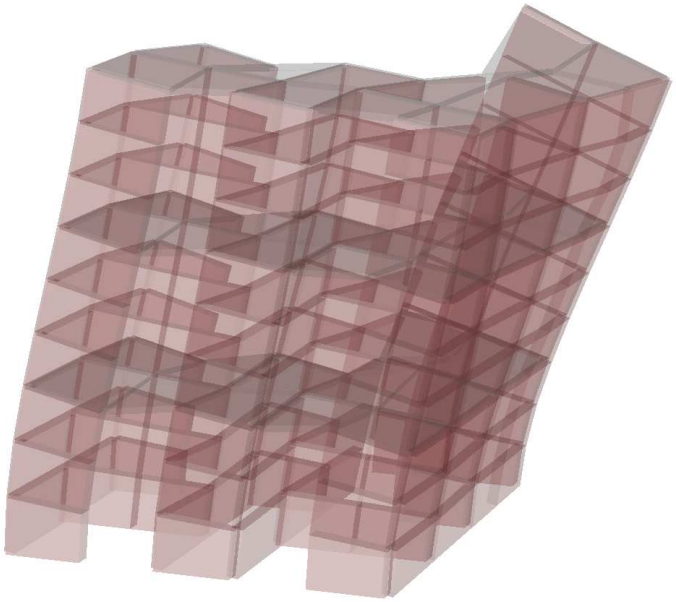

Abb. 6.12: Erste Eigenform des unregelmäßigen Gebäudemodells

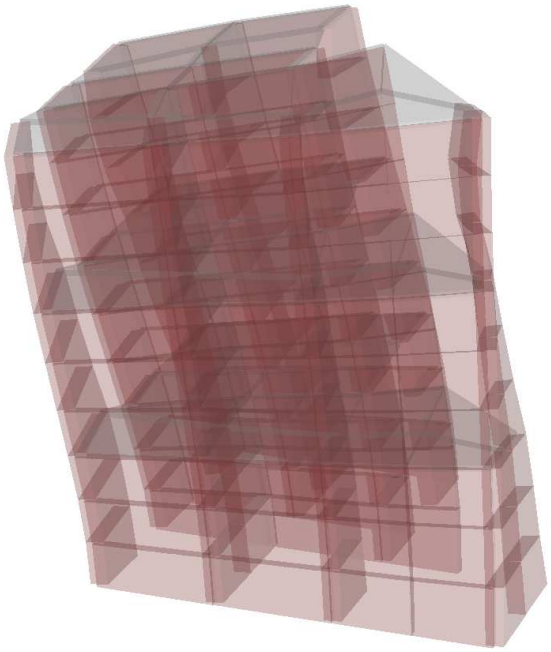

Abb. 6.13: Zweite Eigenform des unregelmäßigen Gebäudemodells

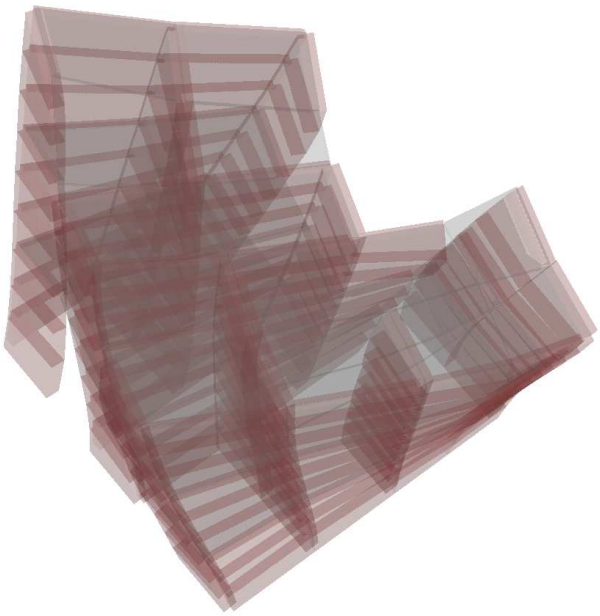

Abb. 6.14: Dritte Eigenform des unregelmäßigen Gebäudemodells

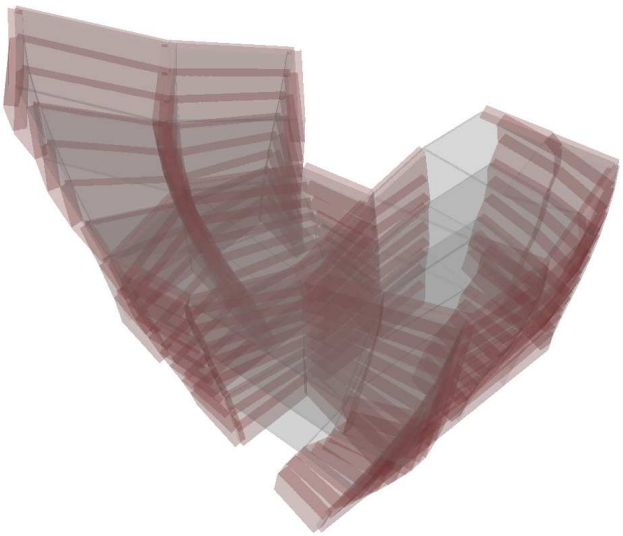

Abb. 6.15: Vierte Eigenform des unregelmäßigen Gebäudemodells

Bei der Untersuchung des regelmäßigen Gebäudes war jeder der ersten beiden Eigenformen eine Verschiebungsrichtung zugeordnet. Deshalb wurde für Eigenform 1 ausschließlich eine Pushover-Analyse in x-Richtung und für Eigenform 2 in y-Richtung durchgeführt. Beim gegenständlichen unregelmäßigem Gebäudemodell ist diese Zuordnung nicht mehr gegeben. Bereits ab der ersten Eigenform sind Rotationsanteile in der Verformungsfigur erkennbar. Deshalb wird der Ansatz des Nachweiskonzeptes erweitert. Für jede Eigenform erfolgt eine Pushover-Analyse in

x-Richtung und in y-Richtung. Dadurch erhält man für jede Richtung den maximal erreichbaren Basisquerkraftwiderstand $\boldsymbol{Q_{b,i,x,R}}$ und $\boldsymbol{Q_{b,i,y,R}}$.

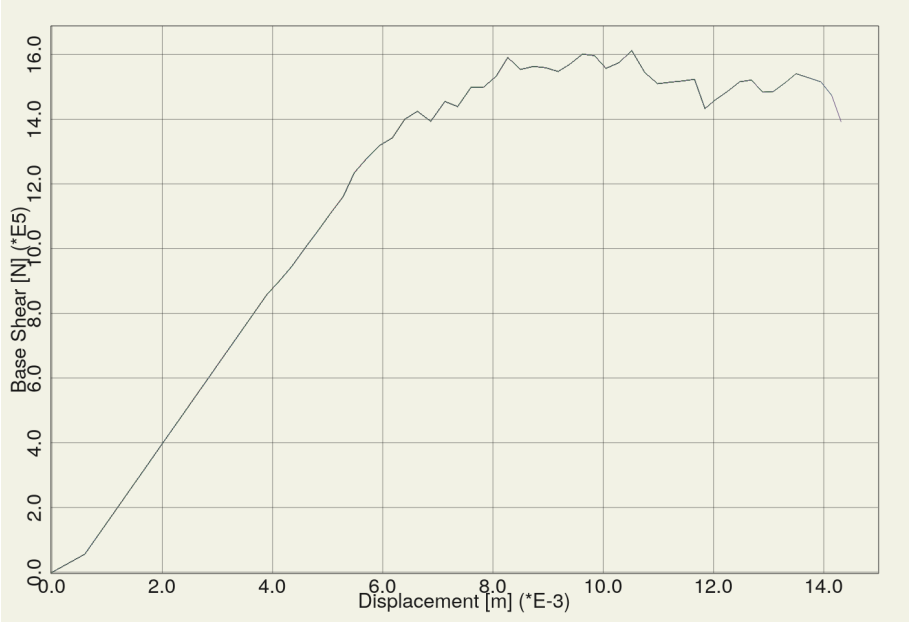

Abb. 6.16: Pushover-Analyse gemäß erster Eigenform - Basisquerkraft in x-Richtung $\boldsymbol{Q_{b,1,x,R}}$

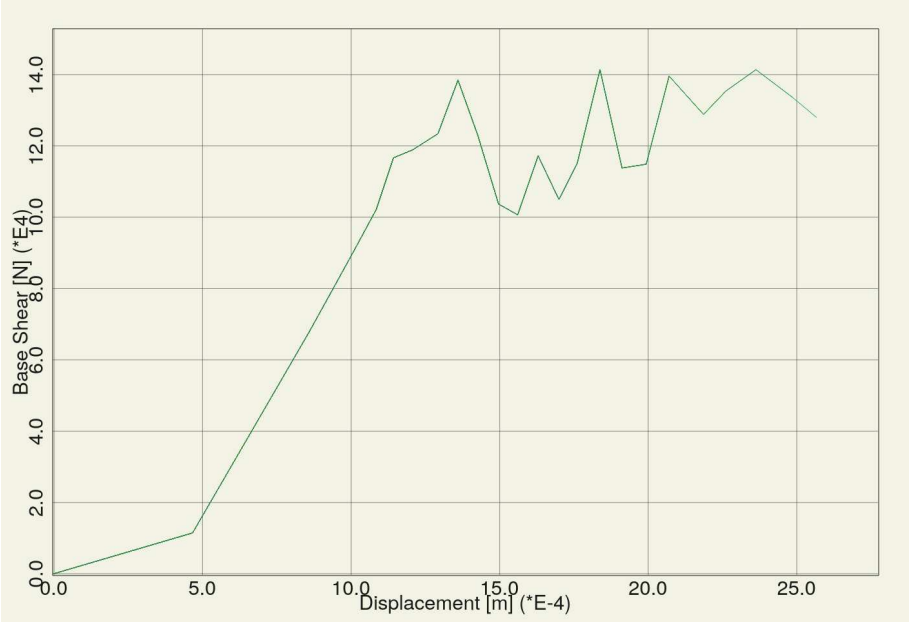

Abb. 6.17: Pushover-Analyse gemäß erster Eigenform - Basisquerkraft in y-Richtung $\boldsymbol{Q_{b,1,y,R}}$

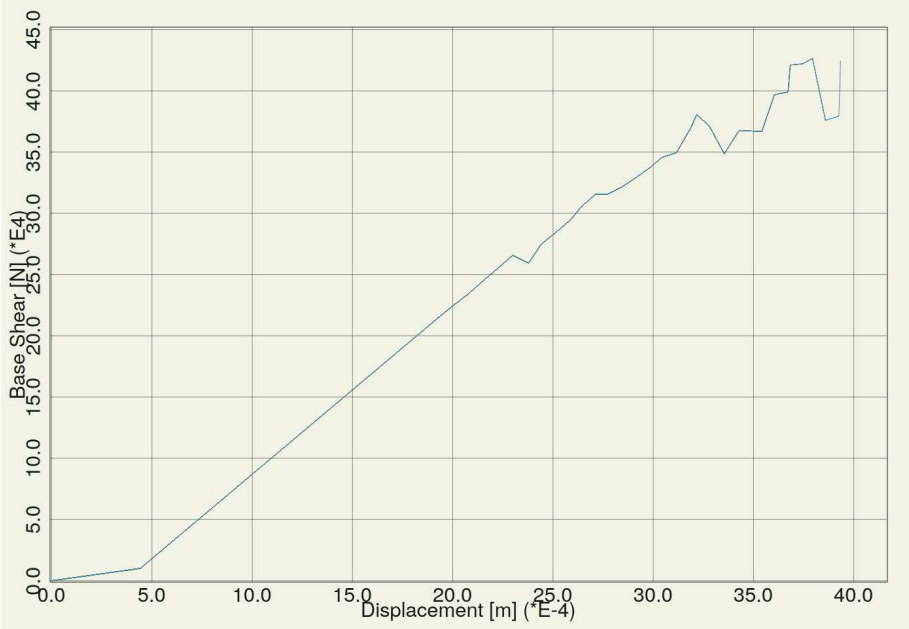

Abb. 6.18: Pushover-Analyse gemäß zweiter Eigenform - Basisquerkraft in x-Richtung $Q_{b,2,x,R}$

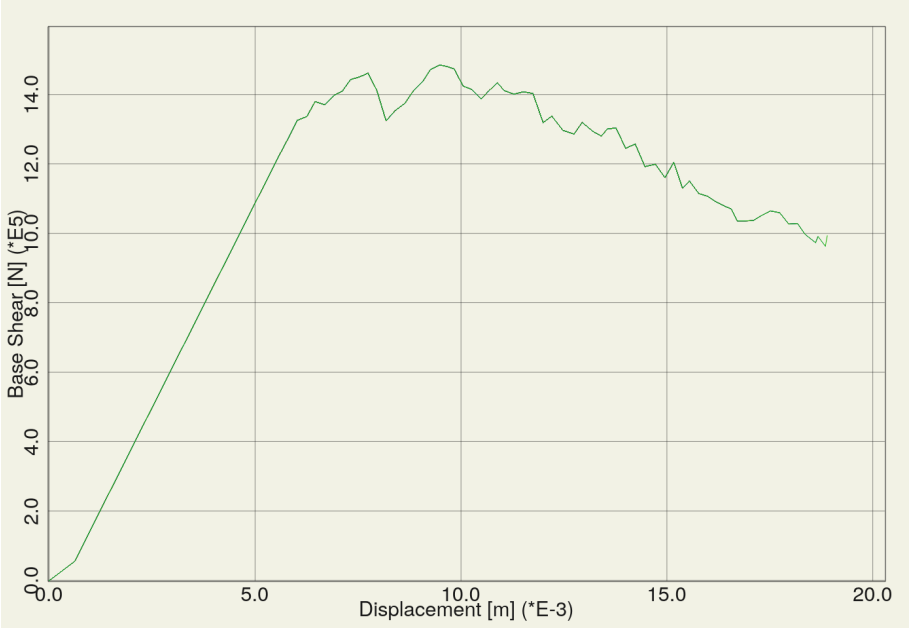

Abb. 6.19: Pushover-Analyse gemäß zweiter Eigenform - Basisquerkraft in y-Richtung $Q_{b,2,y,R}$

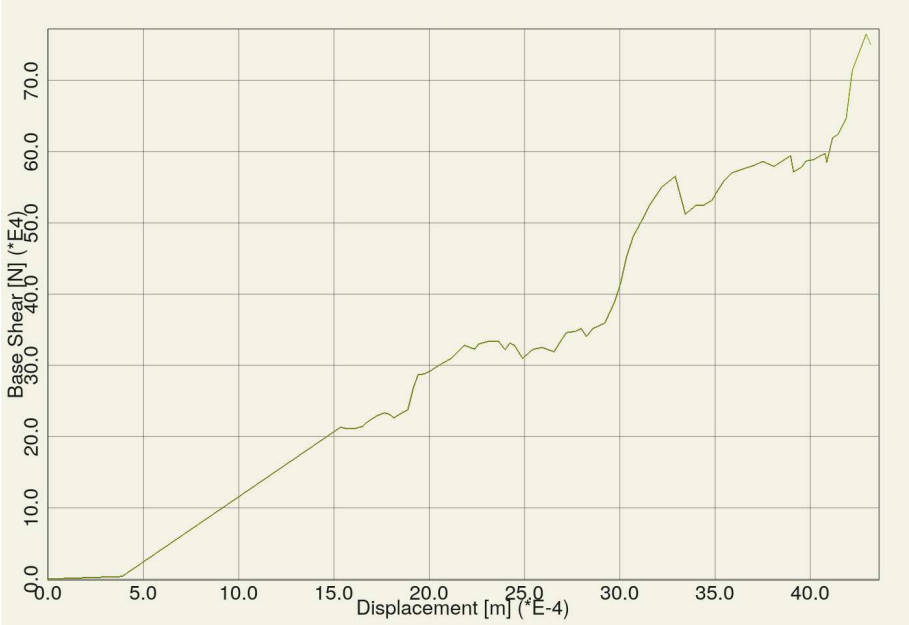

Abb. 6.20: Pushover-Analyse gemäß dritter Eigenform - Basisquerkraft in x-Richtung $Q_{b,3,x,R}$

Abb. 6.21: Pushover-Analyse gemäß dritter Eigenform - Basisquerkraft in y-Richtung $Q_{b,3,y,R}$

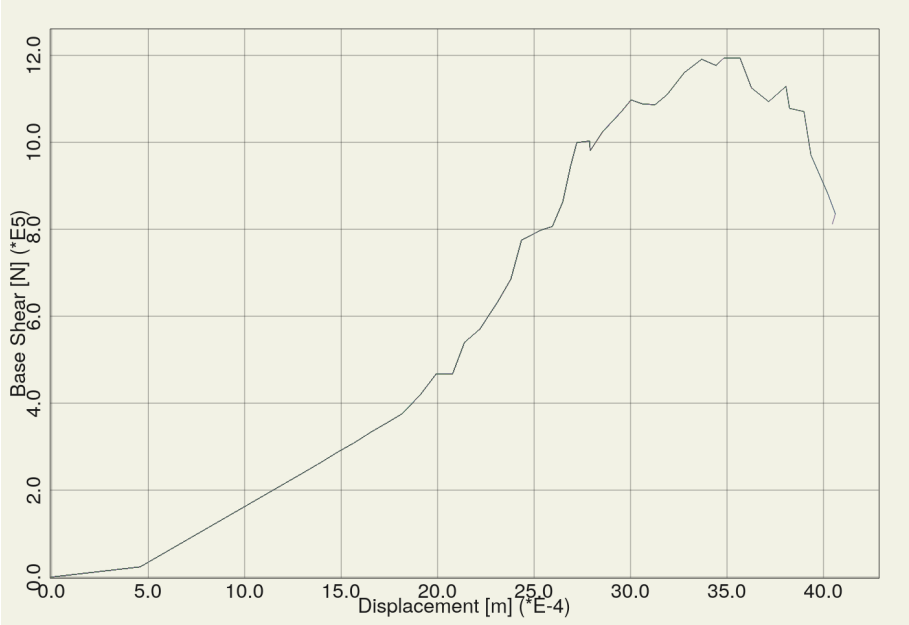

Abb. 6.22: Pushover-Analyse gemäß vierter Eigenform - Basisquerkraft in x-Richtung $Q_{b,4,x,R}$

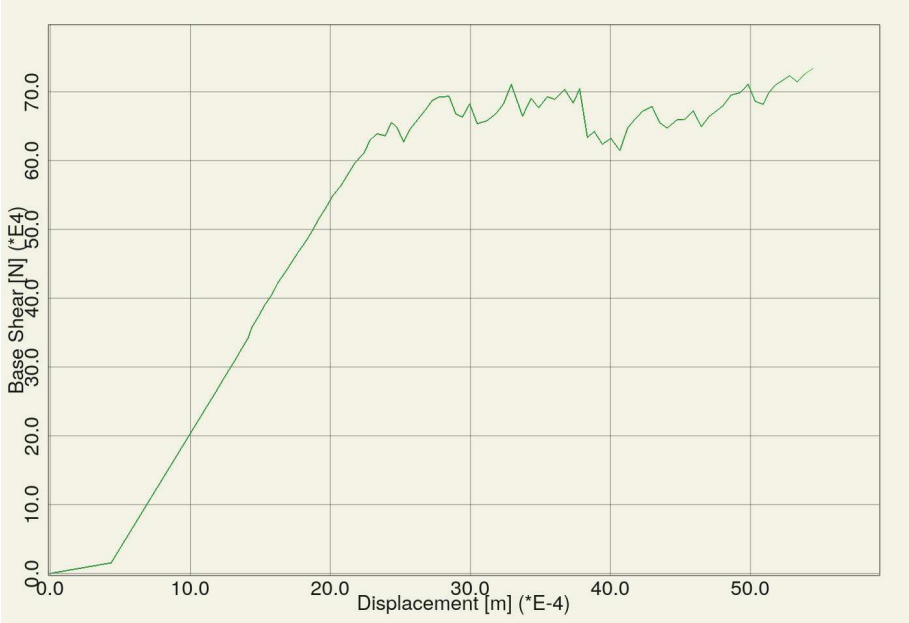

Abb. 6.23: Pushover-Analyse gemäß vierter Eigenform - Basisquerkraft in y-Richtung $Q_{b,4,y,R}$

Im Anschluss werden die Widerstände für beide Richtungen wie in Gleichung 6.8 beschrieben überlagert, um eine Resultierende zu berechnen. Dieser resultierende Basisquerkraftwiderstand $Q_{b,i,R}$ wird für jede Eigenform der einwirkenden Gesamterdbebenkraft $F_{b,i}$ gegenübergestellt. Hierin steht der Index i für die jeweils untersuchte Eigenform.

$$Q_{b,i,R} = \sqrt{Q_{b,i,x,R}^2 + Q_{b,i,y,R}^2} \tag{6.8}$$

$$\begin{aligned}
\boldsymbol{Eigenform1:}\quad & \boldsymbol{Q_{b,1,R} = \sqrt{Q_{b,1,x,R}^2 + Q_{b,1,y,R}^2} = \sqrt{1610180^2 + 145903^2} = 1616777 = 1617kN} \\
\boldsymbol{Eigenform2:}\quad & \boldsymbol{Q_{b,2,R} = \sqrt{Q_{b,2,x,R}^2 + Q_{b,2,y,R}^2} = \sqrt{427880^2 + 1496439^2} = 1556410 = 1556kN} \\
\boldsymbol{Eigenform3:}\quad & \boldsymbol{Q_{b,3,R} = \sqrt{Q_{b,3,x,R}^2 + Q_{b,3,y,R}^2} = \sqrt{776465^2 + 799924^2} = 1114799 = 1115kN} \\
\boldsymbol{Eigenform4:}\quad & \boldsymbol{Q_{b,4,R} = \sqrt{Q_{b,4,x,R}^2 + Q_{b,4,y,R}^2} = \sqrt{1199201^2 + 733909^2} = 1405954 = 1406kN}
\end{aligned} \tag{6.9}$$

Da nun die Widerstandskräfte des Gebäudemodells ermittelt wurde, sollen weiters die Gesamterdbebenkräfte $\boldsymbol{F_{b,i}}$ bestimmt werden. Dazu ist die Bestimmung der Perioden notwendig. In diesem Beispiel wird sich auf die Untersuchung der ersten vier Eigenformen beschränkt, da höhere Eigenformen derartig niedrige Perioden aufweisen, sodass die zugeordnete Gesamterdbebenkraft keine maßgebende Größe erreicht.

$$\begin{aligned}
\boldsymbol{T_1} &= \boldsymbol{0{,}197s} \\
\boldsymbol{T_2} &= \boldsymbol{0{,}176s} \\
\boldsymbol{T_3} &= \boldsymbol{0{,}139s} \\
\boldsymbol{T_4} &= \boldsymbol{0{,}090s}
\end{aligned} \tag{6.10}$$

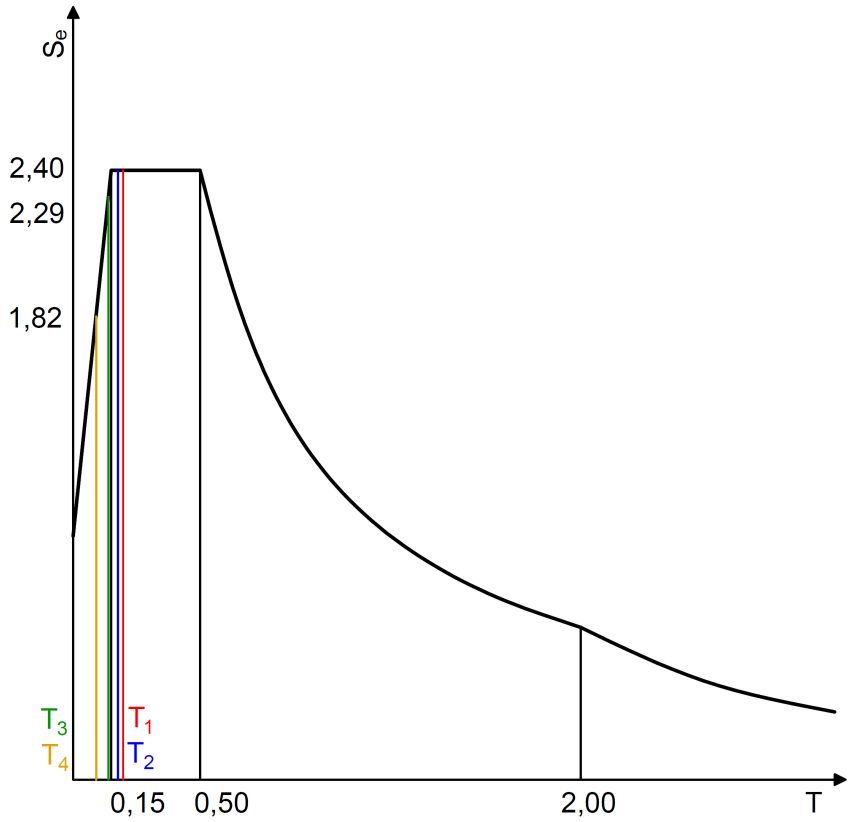

Abb. 6.24: Perioden des unregelmäßigen Gebäudemodells im linear elastischen Antwortspektrum

Die Gesamtmasse des unregelmäßigen L-förmigigen Gemädemodells wurde wie bereits zuvor numerisch ermittelt.

$$\boldsymbol{m = 385881kg} \tag{6.11}$$

Die einwirkenden Gesamterdbebenkräfte können nun für jede untersuchte Eigenform bestimmt werden.

$$\begin{aligned} \boldsymbol{F_{b,1} = S_d(T_1) \cdot m = 2{,}4 \cdot 385881 = 926114 = 926kN} \\ \boldsymbol{F_{b,2} = S_d(T_2) \cdot m = 2{,}4 \cdot 385881 = 926114 = 926kN} \\ \boldsymbol{F_{b,3} = S_d(T_3) \cdot m = 2{,}29 \cdot 385881 = 883667 = 884kN} \\ \boldsymbol{F_{b,4} = S_d(T_4) \cdot m = 1{,}82 \cdot 385881 = 702303 = 702kN} \end{aligned} \tag{6.12}$$

$$\boldsymbol{\eta = max(\frac{F_{b,i}}{Q_{b,R,i}}) = max(\frac{926}{1617}, \frac{926}{1556}, \frac{884}{1115}, \frac{702}{1406}) = 0{,}79 \leq 1{,}00} \tag{6.13}$$

Wie aus dem oben angeführten Nachweis ersichtlich ist, war für das unregelmäßige Beispiel die dritte Eigenform maßgebend und besitzt somit den höchsten Ausnutzungsgrad. Hätte man lediglich eine Pushover-Analyse in beide Hauptsymmetrieachsen des Gebäudes durchgeführt, wäre die Tragsicherheit um über 30% überschätzt worden. Durch die Analyse des unregelmäßigen Gebäudemodells mit einem L-förmigen Grundriss konnte somit gezeigt werden, dass für derartige Bauwerke auch Rotationseigenformen maßgebend werden können. Diese zu vernachlässigen, wie es oftmals vereinfacht vorgeschlagen wird[14], kann zu Bemessungsergebnissen auf der unsicheren Seite führen.

Abschließend muss hier nochmals erwähnt werden, dass das oben angewendete Nachweiskonzept eine vereinfachte und nicht normkonforme Methodik umfasst, die zur Veranschaulichung des Einflusses der unterschiedlichen Eigenformen auf die Tragsicherheit der untersuchten Strukturen dient.

Kapitel 7

Abschluss

In diesem abschließenden Kapitel soll ein Rückblick auf die in dieser Arbeit untersuchten Inhalte gegeben werden. Anfangs wurde ein kurzer Überblick über die Berechnungsverfahren nach Eurocode 8[14] und das statische Tragsystem von Wiener Gründerzeithäusern gegeben. Es wurde beschlossen, derartige Gebäude näher durch das Verfahren der Pushover-Analyse zu untersuchen, da dieses der Bemessung aus den genannten Gründen höhere Tragreserven als die vereinfachten Nachweisverfahren zugänglich macht. Dazu sollten ein regelmäßiges und ein unregelmäßiges L-förmiges Gebäudemodell untersucht werden, um insbesondere den Einfluss von Rotationseigenformen auf die Tragsicherheit von Bauwerken zu ermitteln, da diese oftmals im Rahmen von Pushover-Analysen vernachlässigt werden.

Um die Berechnung zu bewerkstelligen, wurde auf die an der TU Wien entwickelte Skriptsprache slangTNG[17] zurückgegriffen. Diese sollte um das rechteckige Finite Element **RQuad** erweitert werden, um die Wandscheiben der Gebäudemodelle zu modellieren. Bei diesem Element handelt es sich um ein rechteckiges, ebenes Schalenelement, dessen Knoten jeweils die vollen sechs Freiheitsgrade zur Verfügung stehen. Dieses Element wurde nach dessen Vorstellung in Kapitel 2 durch zahlreiche numerische Beispiele in Kapitel 3 auf seine Zuverlässigkeit und Genauigkeit getestet.

Weiters wurde in Kapitel 4 das Materialmodell **ElasticFailure** definiert, welches ein Versagen der Elemente nach Spannungsüberschreitungen in den einzelnen Integrationspunkten abzubilden vermag. Dadurch wurde sichergestellt, dass die im Zuge der Pushover-Analyse monoton gesteigerten Horizontallasten ab einem gewissen Punkt der Berechnung zur Verminderung der Steifigkeit und folglich zum Ausfall einzelner Elemente führen, wodurch es zu einem Abflachen der Kurve der Kraft-Verformunsbeziehungen kommt. Erst dadurch wird eine Pushover-Analyse möglich. Schließlich wurde auch die Funktionsweise des Materialmodells in zwei Anwendungsbeispielen näher beschrieben. Dabei wurde die Pushover-Analyse in dieser Arbeit jeweils bis zum Versagen der gesamten untersuchten Struktur durchgeführt.

Im darauf folgenden Kapitel 5 wurden einige theoretische Grundlagen erläutert, die zum Verständis der nachfolgenden Beispiele zwingend erforderlich sind. Außerdem wurde die Vorgehensweise bei der Berechnung dieser Beispiele zusammenfassend beschrieben.

Im anschließenden Kapitel 6 konnten schließlich die zuvor neu in slangTNG implementierten Funktionalitäten eingesetzt werden, um die Untersuchungen der beiden Gebäudemodelle vorzunehmen.

7.1 Diskussion der Ergebnisse

Das regelmäßige Gebäudemodell wurde durch eine Pushover-Analyse einmal in Gebäudelängsrichtung und einmal in Querrichtung untersucht. Dabei wurde das in dieser Arbeit eingeführte Nachweiskonzept angewendet.

Für das unregelmäßige Gebäudemodell mit L-förmigem Grundriss wurde dieses Konzept entsprechend erweitert, um auch Pushover-Ananlysen gemäß höheren Eigenformen durchführen zu

können. Es konnte gezeigt werden, dass bei derartigen Gebäuden auch eine Rotationseigenform maßgebend für die Bemessung werden kann. Für das vereinfachte hier vorgestellte Nachweiskonzept ergab sich eine Überschätzung der Tragsicherheit von circa 30%. Auf Grund dieser Tatsache wird empfohlen, bei Vernachlässigung höherer Eigenformen in der baupraktischen Berechnung ausreichende Tragreserven vorzusehen. Bei Sicherstellung von Tragreserven könnte man die Pushover-Analyse weiterhin nur für Translationseigenformen durchführen. Empfehlenswert ist jedoch die Berücksichtigung aller maßgebenden Eigenformen, um Abschätzungen eventuell erforderlicher Tragreserven zu vermeiden.

7.2 Ausblick

Das in dieser Arbeit angewendete vereinfachte Nachweiskonzept dient dazu, die gemäß der Pushover-Analysen ermittelten Widerstandskräfte den einwirkenden Erbebenbelastungen gemäß Eurocode 8 gegenüberzustellen. Es hätte jedoch den Rahmen dieser Arbeit gesprengt, ein detailliertes normkonformes Nachweisformat inklusive Teilsicherheitsbeiwerten zu entwerfen. Für weitere Untersuchungen wäre es daher auf jeden Fall von Interesse, die Nachweise auf eine normgerechte Basis zu stellen. Dazu wäre in weiterer Folge eine Erweiterung des zugrunde liegenden Materialmodells **ElasticFailure** erforderlich, welches die verschiedensten Versagensmechanismen einer Mauerwerkswand detaillierter abzubilden vermag. Weiters könnte man den Modellierungsgrad der Decken steigern, um die orthotrope Wirkung mit ihren unterschiedlichen Steifigkeiten in beiden Tragrichtungen zu berücksichtigen.

Einen weiteren interessanten Punkt würde die Untersuchung von Modellen mit unterschiedlichen Deckensteifigkeiten darstellen. Im Zuge von Umbauten und vor allem Dachgeschoßausbauten kommt es oft zu einer Verstärkung der obersten Geschoßdecke, welche meist als Dippelbaumdecke ausgeführt wurde. Durch Einbringen von Verbundmitteln (Schrauben, Dübeln) in die bestehende Holzdecke und anschließende Herstellung einer Aufbetonschicht erhält man eine Holzbetonverbunddecke, welche die Vorteile der Druckfestigkeit des Betons und der Zugfestigkeit des Holzes über die schubfeste Verbindung der Verbundmittel vereint. Diese Verbunddecke besitzt eine stark gesteigerte Wirkung als schubsteife Decke im Vergleich zur bisherigen Dippelbaumdecke. Dadurch werden die darunterliegenden Wandscheiben aus Mauerwerk gekoppelt, das bedeutet, sie beeinflussen sich gegenseitig in ihrer Tragwirkung. Dies sollte in einer wesentlichen Erhöhung der aufnehmbaren Erdbebenbelastung des gesamten Gebäudes resultieren. Eine detaillierte Beschreibung der Veränderung des Tragverhaltens von Bauwerken durch die beschriebenen Maßnahmen wäre auf jeden Fall von Interesse für weitere Untersuchungen.

Schließlich wäre der Vergleich der hier berechneten Pushover-Analysen mit nichtlinearen Zeitverlaufsberechnungen eine sinnvolle Forschungsfrage für weiterführende Arbeiten. Die Möglichkeit einer solchen Berechnung mit den hier zur Verfügung gestellten Mitteln wird im anschließenden Unterkapitel kurz angerissen.

7.3 Demonstration der Möglichkeit einer nichtlinearen Zeitverlaufsberechnung

Hier soll ein kurzer Überblick über die Anwendung der nichtlinearen Zeitverlaufsberechnung in Form eines Beispieles geboten werden, um die Tauglichkeit des Elements **RQuad** zusammen mit dem Materialmodell **ElasticFailure** für derartige Berechnungen zu demonstrieren.

Das im vorigen Kapitel 6 definierte regelmäßige Gebäudemodell wird nun einem Beschleunigungsverlauf ausgesetzt. Die Erdbebenwirkung verläuft dabei in Richtung der Gebäudequerachse. Weitere Details können dem file dynam_RQuad.tng in Anhang E entnommen werden.

Bei dem der Berechnung zugrunde gelegten Erdbeben handelt es sich um das El Centro Beben von 1940, wobei die Beschleunigungsverläufe dazu frei zugänglich sind. Die Abbildung 7.1 zeigt die auf das Bauwerk in y-Achse einwirkenden Bodenbeschleunigungen über die Zeitachse. Die Beschleunigungswerte wurden mit dem Faktor $\frac{1}{2}$ skaliert, um kein sofortiges Versagen der Struktur auszulösen. Weiters wurde die Zeitreihe auf ein Intervall von knapp drei Sekunden gekürzt, um nur einen maßgebenden Ausschnitt zu untersuchen und dadurch die Berechnungszeit erheblich zu verkürzen. Das Intervall zwischen zwei gemessenen Beschleunigungswerten beträgt **0,02** Sekunden. Die Daten sind in einer Matrix mit zwei Spalten gespeichert, wobei in der ersten Spalte die Zeitschritte und in der zweiten Spalte der dazugehörige Beschleunigungswert eingetragen sind.

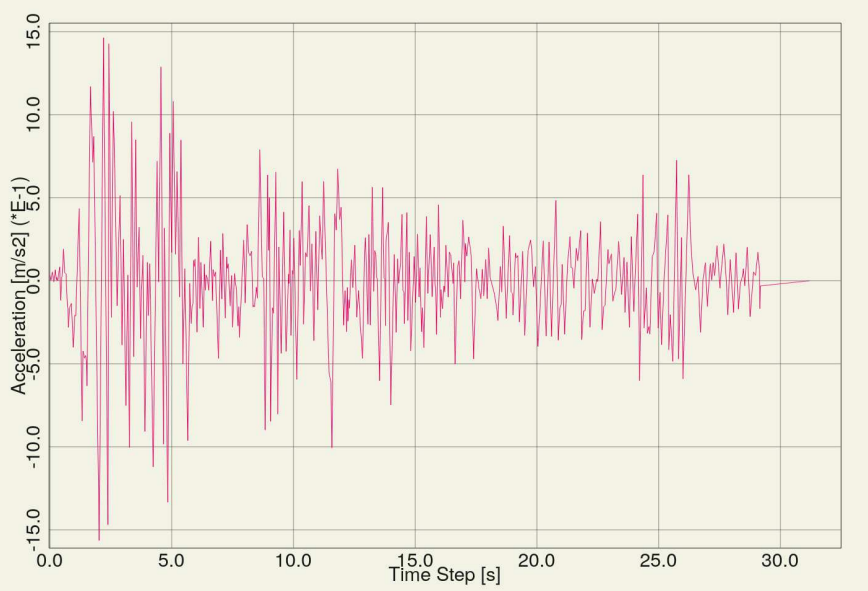

Abb. 7.1: Bodenbeschleuigungsverlauf des der Berechnung zugrunde gelegten Erdbebens

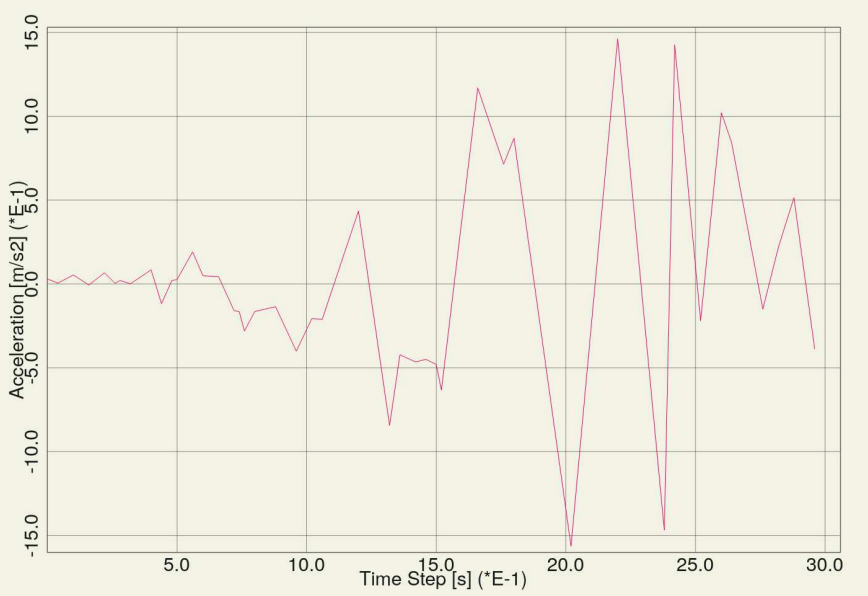

Abb. 7.2: Ausschnitt des untersuchten Intervalls des Bodenbeschleuigungsverlauf

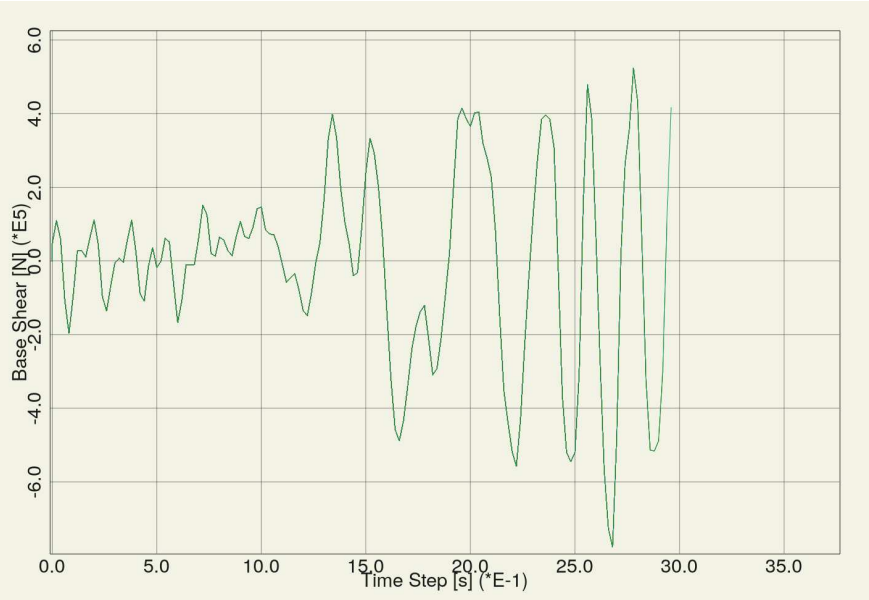

Abb. 7.3: Zeitverlauf der resultierenden Kraft in Richtung der untersuchten Erdbebeneinwirkung

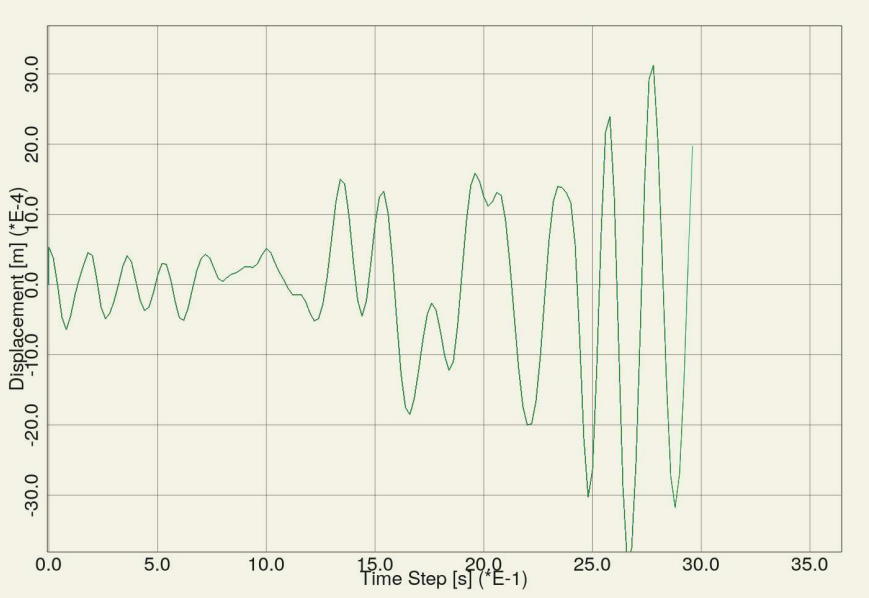

Abb. 7.4: Zeitverlauf der Verschiebungen des Kontrollknotens in Richtung der untersuchten Erdbebeneinwirkung

Literaturverzeichnis

[1] Allman. „A compatible triangular element including vertex rotations for plane elasticity analysis“. In: *Computers & Structures* 19 (1984), S. 1–8.

[2] Autodesk. *Autocad 2018 Studentenversion*. URL: `https://www.autodesk.de/education/edu-software/overview` (Zugriff am 07. 08. 2020).

[3] Klaus-Jürgen Bathe. *Finite Element Procedures*. 1. Auflage. New Jersey: Prentice-Hall Inc., 1996. ISBN: 0-13-301458-4.

[4] Cook. „Four-node 'flat' shell element: drilling degrees of freedom, membrane-bending coupling, warped geometry, and behavior“. In: *Computers & Structures* 50 (1994), S. 549–555.

[5] Cook. „On the allman triangle and a related quadrilateral element“. In: *Computers & Structures* 22 (1986), S. 1065–1067.

[6] Djamel Boutagouga; Kamel Djeghaba. „Nonlinear dynamic co-rotational formulation for membrane elements with in-plane drilling rotational degree of freedom“. In: *Engineering Computations* 33 (2016), S. 667–697.

[7] L. Jin. „Analysis and Evaluation of a Shell Finite Element with Drilling Degree of Freedom“. Thesis Report Master's Degree. University of Maryland, 1994.

[8] Bauer; Kern. „Anhang B zur Erläuterung 03/2013 - Berechnungsbeispiele anhand des Wiener Gründerzeit-Mustergebäudes“. In: *Fachgruppe Bauwesen der LK W/Nö/Bgld* 03/2013 (2014), S. 1–38.

[9] Andreas Kolbitsch. *Altbaukonstruktionen*. Wien: Springer Verlag-Wien, 1989.

[10] Andreas Kolbitsch. *E+E - Erhaltung und Erneuerung von Hochbauten*. LVA Skriptum. TU Verlag, 2018. 184 S.

[11] Michael Mistler. „Verformungsbasiertes seismisches Bemessungskonzept für Mauerwerksbauten“. Dissertation. Technischen Hochschule Aachen, 2006.

[12] Pifko; Winter; Ogilvie. *DYCAST - A finite Element Program for the Crash Analysis of Structures*. Forschungsber. Grumman Corporation Research Center, 1987. 347 S.

[13] *ÖNORM B 1998-1:2011 06 15: Auslegung von Bauwerken gegen Erdbeben – Teil 1: Grundlagen, Erdbebeneinwirkungen und Regeln für Hochbauten - Nationale Festlegungen*. Wien: Austrian Standards, Juni 2011.

[14] *ÖNORM EN 1998-1:2011 06 15: Auslegung von Bauwerken gegen Erdbeben – Teil 1: Grundlagen, Erdbebeneinwirkungen und Regeln für Hochbauten*. Wien: Austrian Standards, Juni 2011.

[15] *ÖNORM EN 1998-3:2013 02 15: Auslegung von Bauwerken gegen Erdbeben – Teil 1: Beurteilung und Ertüchtigung von Gebäuden*. Wien: Austrian Standards, Feb. 2013.

[16] Kefal; Oterkus; Tessler; Spangler. „A quadrilateral inverse-shell element with drilling degrees of freedom for shape sensing and structural health monitoring“. In: *Engineering Science and Technology, an International Journal* 19 (2016), S. 1299–1313.

[17] Christian Bucher; Sebastian Wolff. *slangTNG (structural language)*. URL: `http://info.tuwien.ac.at/bucher/Private/slangTNG.html` (Zugriff am 30. 06. 2020).

Alle selbsterstellten Zeichnungen und Abbildungen wurden mit Hilfe der kostenfreien Studentenversion Autocad 2018[2] erstellt.

Anhang A

Dokumentation implementierter Funktionen

Der Anhang enthält neben einer kurzen Dokumention der implementierten Funktionen den wesentlichen im Rahmen dieser Arbeit erstellten Quellcode.

Bei der folgenden Übersicht über die implementierten Funkionen erfolgt in den jeweils auskommentierten Zeilen eine Erklärung der Eingabeparamter, darunter ist schließlich die tatsächliche Eingabe beispielhaft angegeben.

Definieren einer Struktur

```
-- structure=fem.Structure(Bezeichnung der Struktur)
structure=fem.Structure("building")
```

Konstruktor ElasticFailure

```
-- material = structure:AddMaterial(Materialnummer,
   Materialtyp)
-- material:SetData(tmath.Matrix({{Elastizitätsmodul,
   Querdehnungszahl, Wichte, zulässige Druckspannung (neg.
   Vorzeichen), zulässige Zugspannung (pos. Vorzeichen) }}))
material = structure:AddMaterial(001, 'ELASTIC_FAILURE')
material:SetData(tmath.Matrix({{1.6e+9, 0.2 , 1700, -5e+6, 4e
   +5 }}))
```

Konstruktor RQuad

```
-- structure:AddElements(Elementtyp, Material, Querschnitt,
   tmath.Matrix({{ Elementnummer, Knoten 1, Knoten 2, Knoten
   3, Knoten 4}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
   0001, 001, 002, 003, 004}}))
```

Ausgabe der Versagensmatrizen für die gesamte Struktur

```
print("Failure Tension", structure:GetFailureTension())
print("Failure Compression", structure:GetFailureCompression
   ())
```

Ausgabe der Versagensmatrizen für ein bestimmtes Element

```
1 -- print("Failure Tension", structure:GetFailureTension(
     Elementnummer))
2 print("Failure Tension", structure:GetFailureTension(0001))
3 -- print("Failure Compression", structure:
     GetFailureCompression(Elementnummer))
4 print("Failure Compression", structure:GetFailureCompression
     (0001))
```

Anhang B

Quellcode RQuad

Der Autor dieser Masterarbeit hat die implementierten Finiten Elemente und das Materialmodell nach bestem Wissen und Gewissen erstellt. Es kann jedoch keine Haftung für die Richtigkeit gewährleistet werden und der Autor haftet nicht für dadurch entstehende eventuelle Schäden. Das folgende file RQuad.cpp enthält den Quellcode des implementierten Elements **RQuad**.

RQuad.cpp

```
1  //Copyright: Matthias Grobauer, Master Thesis 2020, TU Wien
2
3  #include "Structure.h"
4  #include "Element.h"
5  #include "graphics/Draw.h"
6  #include "graphics/Canvas.h"
7  #include "fem/common/fem_mat_vec.h"
8  #include "tmath/functions/functions.hpp"
9  #include "fem/common/fem_throw.h"
10
11 #include <iostream>
12
13
14 namespace fem {
15
16
17 void ShapeFunc(const double & x, const double & y, const
       double & a, const double & b, double *Ni) {
18 // shape functions
19
20 Ni[0] = (a-x)*(b-y)/(2*a*2*b);
21 Ni[1] = (a+x)*(b-y)/(2*a*2*b);
22 Ni[2] = (a+x)*(b+y)/(2*a*2*b);
23 Ni[3] = (a-x)*(b+y)/(2*a*2*b);
24 Ni[4] = (a*a-x*x)*(b-y)/(2*a*a*b);
25 Ni[5] = (a+x)*(b*b-y*y)/(2*a*b*b);
26 Ni[6] = (a*a-x*x)*(b+y)/(2*a*a*b);
27 Ni[7] = (a-x)*(b*b-y*y)/(2*a*b*b);
28
29 }
30
```

```
31 void ShapeFuncDer(const double & x, const double & y, const
      double & a, const double & b, double *Ni_x, double* Ni_y)
      {
32 // derivatives of shape functions
33
34 Ni_x[0] = (y-b)/(2*a*2*b);
35 Ni_x[1] = -(y-b)/(2*a*2*b);
36 Ni_x[2] = (y+b)/(2*a*2*b);
37 Ni_x[3] = -(y+b)/(2*a*2*b);
38 Ni_x[4] = x*(y-b)/(a*a*b);
39 Ni_x[5] = (b*b-y*y)/(2*a*b*b);
40 Ni_x[6] = -x*(b+y)/(a*a*b);
41 Ni_x[7] = (y*y-b*b)/(2*a*b*b);
42
43 Ni_y[0] = (x-a)/(2*a*2*b);
44 Ni_y[1] = -(x+a)/(2*a*2*b);
45 Ni_y[2] = (x+a)/(2*a*2*b);
46 Ni_y[3] = -(x-a)/(2*a*2*b);
47 Ni_y[4] = (x*x-a*a)/(2*a*a*b);
48 Ni_y[5] = -y*(a+x)/(a*b*b);
49 Ni_y[6] = (a*a-x*x)/(2*a*a*b);
50 Ni_y[7] = y*(x-a)/(a*b*b);
51
52 }
53
54 int RQuadIntPoints(Structure *s, Element* el) {
55 //       std::cout << std::endl << "RQuadIntPoints BEGIN " <<
       std::endl;
56 int nIntPoints = el->elementType->numIntPoints;
57 int nStressPoints = el->elementType->numStressPoints;
58 bool isRect = 0;
59
60 for (int i=0; i<nIntPoints+nStressPoints; i++) {
61 el->intPoints[i]->shape = tmath::Matrix(3,24);
62 el->intPoints[i]->shapeDer = tmath::Matrix(9,24);
63 //    el->intPoints[i]->shapeDerGeo = tmath::Matrix(4,24);
64 }
65
66 double thickness = el->section->physicalData.data()[0]*el->
      physicalFactors.data()[0];
67
68 //    printf("thickness %g\n", thickness);
69
70 double c0[3], c1[3], c2[3], c3[3];
71 double lc0[3], lc1[3], lc2[3], lc3[3];
72 memcpy(c0, s->nodes[el->nodes[0]]->coordinates, 3*sizeof(
      double));
73 memcpy(c1, s->nodes[el->nodes[1]]->coordinates, 3*sizeof(
      double));
```

```
74  memcpy(c2, s->nodes[el->nodes[2]]->coordinates, 3*sizeof(
        double));
75  memcpy(c3, s->nodes[el->nodes[3]]->coordinates, 3*sizeof(
        double));
76
77  //    matrix_print_real(c0, 1, 3, "%g ", "c0");
78  //    matrix_print_real(c1, 1, 3, "%g ", "c1");
79  //    matrix_print_real(c2, 1, 3, "%g ", "c2");
80  //    matrix_print_real(c3, 1, 3, "%g ", "c3");
81  //    matrix_print_real(el->trafoMat, 3, 3, "%g ", "trafoMat
        ");
82
83  matrix_mult(el->trafoMat, 3, 3, c0, 1, lc0);
84  matrix_mult(el->trafoMat, 3, 3, c1, 1, lc1);
85  matrix_mult(el->trafoMat, 3, 3, c2, 1, lc2);
86  matrix_mult(el->trafoMat, 3, 3, c3, 1, lc3); //transformed in
        xy-plane
87
88  //    matrix_print_real(lc0, 1, 3, "%g ", "lc0");
89  //    matrix_print_real(lc1, 1, 3, "%g ", "lc1");
90  //    matrix_print_real(lc2, 1, 3, "%g ", "lc2");
91  //    matrix_print_real(lc3, 1, 3, "%g ", "lc3");
92
93  double l01[3], l12[3], l23[3], l30[3];
94
95  memcpy(l01, lc1, 3*sizeof(double));    //lenghts of all four
        edges
96  memcpy(l12, lc2, 3*sizeof(double));
97  memcpy(l23, lc3, 3*sizeof(double));
98  memcpy(l30, lc3, 3*sizeof(double));
99  matrix_minus(l01, lc0, 3, 1);
100 matrix_minus(l12, lc1, 3, 1);
101 matrix_minus(l23, lc2, 3, 1);
102 matrix_minus(l30, lc0, 3, 1);
103
104 //    matrix_print_real(l01, 1, 3, "%g ", "l01");
105 //    matrix_print_real(l12, 1, 3, "%g ", "l12");
106 //    matrix_print_real(l23, 1, 3, "%g ", "l23");
107 //    matrix_print_real(l30, 1, 3, "%g ", "l30");
108
109 double x[4] = {lc0[0], lc1[0], lc2[0], lc3[0]};
110 double y[4] = {lc0[1], lc1[1], lc2[1], lc3[1]};
111
112 //    matrix_print_real(x, 1, 4, "%g ", "x");
113 //    matrix_print_real(y, 1, 4, "%g ", "y");
114
115 double e1[3], e2[3];
116 vector_cross(l01, l12, e1);
117 vector_cross(l23, l30, e2);
```

```
118 double vol1, vol2;
119 vector_length(e1, 3, &vol1);
120 vector_length(e2, 3, &vol2);
121 el->volume = thickness*fabs(vol1+vol2)/2.;
122
123 //    printf("volume = %g\n", el->volume);
124
125 //for rectangular quad only
126 // width of element = 2*a, height of element = 2*b
127
128 double a_width = fabs(l01[0]/2);
129 double b_height = fabs(l12[1]/2);
130
131 //    printf("a = %g\n", a_width);
132 //    printf("b = %g\n", b_height);
133
134 double vol_check = a_width*2*b_height*2*thickness;
135 //    cout << endl << "volume_check= " << vol_check << endl;
136
137 if (el->volume==vol_check)
138 isRect=1;
139 if (!isRect)
140 fem_throw("Element of Type RQuad must be rectangular");
141
142
143 // integration order 3x3 (9 Integrationspunkte) Bathe page
    462
144 double gm = 0.; //Int Coordinates
145 double gc = 0.774596669241483;
146
147 double wm = 0.888888888888889;
148 double wc = 0.555555555555556;
149
150 double intPoints_xi[] = {-gc, gc, gc, -gc, gm, gc, gm, -gc,
    gm};
151 double intPoints_eta[] = {-gc, -gc, gc, gc, -gc, gm, gc, gm,
    gm};
152
153 //    matrix_print_real(intPoints_xi, 1, 9, "%g ", "
    intPoints_xi");
154 //    matrix_print_real(intPoints_eta, 1, 9, "%g ", "
    intPoints_eta");
155
156 //Coordinates of 9*2 IntPoints
157 for (int j=0; j<2; j++) {
158 //int points
159 el->intPoints[0+j*9]->coor.data()[0] = intPoints_xi[0]*
    a_width;
```

```
160 el->intPoints[0+j*9]->coor.data()[1] = intPoints_eta[0]*
      b_height;
161 el->intPoints[1+j*9]->coor.data()[0] = intPoints_xi[1]*
      a_width;
162 el->intPoints[1+j*9]->coor.data()[1] = intPoints_eta[1]*
      b_height;
163 el->intPoints[2+j*9]->coor.data()[0] = intPoints_xi[2]*
      a_width;
164 el->intPoints[2+j*9]->coor.data()[1] = intPoints_eta[2]*
      b_height;
165 el->intPoints[3+j*9]->coor.data()[0] = intPoints_xi[3]*
      a_width;
166 el->intPoints[3+j*9]->coor.data()[1] = intPoints_eta[3]*
      b_height;
167 el->intPoints[4+j*9]->coor.data()[0] = intPoints_xi[4]*
      a_width;
168 el->intPoints[4+j*9]->coor.data()[1] = intPoints_eta[4]*
      b_height;
169 el->intPoints[5+j*9]->coor.data()[0] = intPoints_xi[5]*
      a_width;
170 el->intPoints[5+j*9]->coor.data()[1] = intPoints_eta[5]*
      b_height;
171 el->intPoints[6+j*9]->coor.data()[0] = intPoints_xi[6]*
      a_width;
172 el->intPoints[6+j*9]->coor.data()[1] = intPoints_eta[6]*
      b_height;
173 el->intPoints[7+j*9]->coor.data()[0] = intPoints_xi[7]*
      a_width;
174 el->intPoints[7+j*9]->coor.data()[1] = intPoints_eta[7]*
      b_height;
175 el->intPoints[8+j*9]->coor.data()[0] = intPoints_xi[8]*
      a_width;
176 el->intPoints[8+j*9]->coor.data()[1] = intPoints_eta[8]*
      b_height;
177
178 //stress points
179 el->intPoints[18+j*4]->coor.data()[0] = intPoints_xi[4]*
      a_width;
180 el->intPoints[18+j*4]->coor.data()[1] = intPoints_eta[4]*
      b_height;
181 el->intPoints[19+j*4]->coor.data()[0] = intPoints_xi[5]*
      a_width;
182 el->intPoints[19+j*4]->coor.data()[1] = intPoints_eta[5]*
      b_height;
183 el->intPoints[20+j*4]->coor.data()[0] = intPoints_xi[6]*
      a_width;
184 el->intPoints[20+j*4]->coor.data()[1] = intPoints_eta[6]*
      b_height;
```

```
185 el->intPoints[21+j*4]->coor.data()[0] = intPoints_xi[7]*
       a_width;
186 el->intPoints[21+j*4]->coor.data()[1] = intPoints_eta[7]*
       b_height;
187 }
188
189 double sq3 = .5/sqrt(3.);
190
191 for (int j=0; j<9; j++) {
192 el->intPoints[j]->coor.data()[2] = thickness*sq3;
193 el->intPoints[j+9]->coor.data()[2] = -thickness*sq3;
194 }
195
196 for (int j=0; j<4; j++) {
197 el->intPoints[18+j]->coor.data()[2] = thickness*0.5;
198 el->intPoints[22+j]->coor.data()[2] = -thickness*0.5;
199 }
200
201 //Applying weights to IntPoints
202 for (int j=0; j<4; j++) {
203 el->intPoints[j]->weight = wc*a_width*wc*b_height;
204 el->intPoints[j+4]->weight = wm*a_width*wc*b_height;
205 el->intPoints[j+9]->weight = wc*a_width*wc*b_height;
206 el->intPoints[j+13]->weight = wm*a_width*wc*b_height;
207 }
208 el->intPoints[8]->weight = wm*a_width*wm*b_height;
209 el->intPoints[17]->weight = wm*a_width*wm*b_height;
210
211 double sum_weights = 0.;
212 for (int j=0; j<18; j++) {
213 sum_weights += el->intPoints[j]->weight;
214 //              std::cout << std::endl << "intpoint " << j+1
       << "  - weight = " << el->intPoints[j]->weight << std::
       endl;
215 }
216 //           cout << endl << "sum weights = " << sum_weights
       << endl;
217
218 double xij[4], yij[4], lij[4], bk[4], ck[4], dk[4], ek[4];
219
220 xij[0] = x[1]-x[0];
221 xij[1] = x[2]-x[1];
222 xij[2] = x[3]-x[2];
223 xij[3] = x[0]-x[3];
224
225 yij[0] = y[1]-y[0];
226 yij[1] = y[2]-y[1];
227 yij[2] = y[3]-y[2];
228 yij[3] = y[0]-y[3];
```

```
229
230 for (int k=0; k<4; k++) {
231 lij[k] = sqrt(xij[k]*xij[k]+yij[k]*yij[k]);
232 bk[k] = (0.5*yij[k]*yij[k]-0.25*xij[k]*xij[k])/(lij[k]*lij[k
       ]);
233 ck[k] = -xij[k]/(lij[k]*lij[k]);
234 dk[k] = (0.5*xij[k]*xij[k]-0.25*yij[k]*yij[k])/(lij[k]*lij[k
       ]);
235 ek[k] = -yij[k]/(lij[k]*lij[k]);
236 }
237
238 //Setup variables
239 double Ni[8], Li[4], Mi[4], Ni_x[8], Ni_y[8], Li_x[4], Li_y
       [4], Mi_x[4], Mi_y[4];
240 double Nib[8], Nib_x[8], Nib_y[8], Hi_x[12], Hi_y[12], Hi_xx
       [12], Hi_yy[12], Hi_xy[12], Hi_yx[12];
241 double Nibend[12], Nibend_xx[12], Nibend_yy[12], Nibend_xy
       [12];
242
243 //Loop over all intPoints
244 for (int i=0; i<nIntPoints+nStressPoints; i++) {
245 double* h = el->intPoints[i]->shape.data();
246 double* b = el->intPoints[i]->shapeDer.data();
247 matrix_init(h, 3, 24);
248 matrix_init(b, 9, 24);
249
250 double x_intPoint = el->intPoints[i]->coor.data()[0];
251 double y_intPoint = el->intPoints[i]->coor.data()[1];
252 double z_intPoint = el->intPoints[i]->coor.data()[2]*(-1);
253
254 //calculation of h-matrix and b-matrix
255
256 //shapes for membrane part
257 ShapeFunc(x_intPoint, y_intPoint, a_width, b_height, Ni);
258 ShapeFuncDer(x_intPoint, y_intPoint, a_width, b_height, Ni_x,
        Ni_y);
259
260 //shapes for bending part
261 ShapeFunc(x_intPoint, y_intPoint, a_width, b_height, Nib);
262 ShapeFuncDer(x_intPoint, y_intPoint, a_width, b_height, Nib_x
       , Nib_y);
263 Nib[0] -= (0.5*Ni[7]+0.5*Ni[4]);
264 Nib[1] -= (0.5*Ni[4]+0.5*Ni[5]);
265 Nib[2] -= (0.5*Ni[5]+0.5*Ni[6]);
266 Nib[3] -= (0.5*Ni[6]+0.5*Ni[7]);
267 Nib_x[0] -= (0.5*Ni_x[7]+0.5*Ni_x[4]);
268 Nib_x[1] -= (0.5*Ni_x[4]+0.5*Ni_x[5]);
269 Nib_x[2] -= (0.5*Ni_x[5]+0.5*Ni_x[6]);
270 Nib_x[3] -= (0.5*Ni_x[6]+0.5*Ni_x[7]);
```

```
271 Nib_y[0] -= (0.5*Ni_y[7]+0.5*Ni_y[4]);
272 Nib_y[1] -= (0.5*Ni_y[4]+0.5*Ni_y[5]);
273 Nib_y[2] -= (0.5*Ni_y[5]+0.5*Ni_y[6]);
274 Nib_y[3] -= (0.5*Ni_y[6]+0.5*Ni_y[7]);
275
276 //Terms for membrane part
277 Li[0] = ((y[3]-y[0])*Ni[7]-(y[1]-y[0])*Ni[4])/8.;
278 Li[1] = ((y[0]-y[1])*Ni[4]-(y[2]-y[1])*Ni[5])/8.;
279 Li[2] = ((y[1]-y[2])*Ni[5]-(y[3]-y[2])*Ni[6])/8.;
280 Li[3] = ((y[2]-y[3])*Ni[6]-(y[0]-y[3])*Ni[7])/8.;
281
282 Li_x[0] = ((y[3]-y[0])*Ni_x[7]-(y[1]-y[0])*Ni_x[4])/8.;
283 Li_x[1] = ((y[0]-y[1])*Ni_x[4]-(y[2]-y[1])*Ni_x[5])/8.;
284 Li_x[2] = ((y[1]-y[2])*Ni_x[5]-(y[3]-y[2])*Ni_x[6])/8.;
285 Li_x[3] = ((y[2]-y[3])*Ni_x[6]-(y[0]-y[3])*Ni_x[7])/8.;
286
287 Li_y[0] = ((y[3]-y[0])*Ni_y[7]-(y[1]-y[0])*Ni_y[4])/8.;
288 Li_y[1] = ((y[0]-y[1])*Ni_y[4]-(y[2]-y[1])*Ni_y[5])/8.;
289 Li_y[2] = ((y[1]-y[2])*Ni_y[5]-(y[3]-y[2])*Ni_y[6])/8.;
290 Li_y[3] = ((y[2]-y[3])*Ni_y[6]-(y[0]-y[3])*Ni_y[7])/8.;
291
292 Mi[0] = (-(x[3]-x[0])*Ni[7]+(x[1]-x[0])*Ni[4])/8.;
293 Mi[1] = (-(x[0]-x[1])*Ni[4]+(x[2]-x[1])*Ni[5])/8.;
294 Mi[2] = (-(x[1]-x[2])*Ni[5]+(x[3]-x[2])*Ni[6])/8.;
295 Mi[3] = (-(x[2]-x[3])*Ni[6]+(x[0]-x[3])*Ni[7])/8.;
296
297 Mi_x[0] = (-(x[3]-x[0])*Ni_x[7]+(x[1]-x[0])*Ni_x[4])/8.;
298 Mi_x[1] = (-(x[0]-x[1])*Ni_x[4]+(x[2]-x[1])*Ni_x[5])/8.;
299 Mi_x[2] = (-(x[1]-x[2])*Ni_x[5]+(x[3]-x[2])*Ni_x[6])/8.;
300 Mi_x[3] = (-(x[2]-x[3])*Ni_x[6]+(x[0]-x[3])*Ni_x[7])/8.;
301
302 Mi_y[0] = (-(x[3]-x[0])*Ni_y[7]+(x[1]-x[0])*Ni_y[4])/8.;
303 Mi_y[1] = (-(x[0]-x[1])*Ni_y[4]+(x[2]-x[1])*Ni_y[5])/8.;
304 Mi_y[2] = (-(x[1]-x[2])*Ni_y[5]+(x[3]-x[2])*Ni_y[6])/8.;
305 Mi_y[3] = (-(x[2]-x[3])*Ni_y[6]+(x[0]-x[3])*Ni_y[7])/8.;
306
307 //Terms for bending
308 Hi_x[0] = 3/2*(ck[0]*Nib[4]-ck[3]*Nib[7]);
309 Hi_x[1] = 0.;
310 Hi_x[2] = -Nib[0]-bk[0]*Nib[4]-bk[3]*Nib[7];
311 Hi_x[3] = 3/2*(ck[1]*Nib[5]-ck[0]*Nib[4]);
312 Hi_x[4] = 0.;
313 Hi_x[5] = -Nib[1]-bk[1]*Nib[5]-bk[0]*Nib[4];
314 Hi_x[6] = 3/2*(ck[2]*Nib[6]-ck[1]*Nib[5]);
315 Hi_x[7] = 0.;
316 Hi_x[8] = -Nib[2]-bk[2]*Nib[6]-bk[1]*Nib[5];
317 Hi_x[9] = 3/2*(ck[3]*Nib[7]-ck[2]*Nib[6]);
318 Hi_x[10] = 0.;
319 Hi_x[11] = -Nib[3]-bk[3]*Nib[7]-bk[2]*Nib[6];
```

```
320
321 Hi_y[0] = 3/2*(ek[0]*Nib[4]-ek[3]*Nib[7]);
322 Hi_y[1] = Nib[0]+dk[0]*Nib[4]+dk[3]*Nib[7];
323 Hi_y[2] = 0.;
324 Hi_y[3] = 3/2*(ek[1]*Nib[5]-ek[0]*Nib[4]);
325 Hi_y[4] = Nib[1]+dk[1]*Nib[5]+dk[0]*Nib[4];
326 Hi_y[5] = 0.;
327 Hi_y[6] = 3/2*(ek[2]*Nib[6]-ek[1]*Nib[5]);
328 Hi_y[7] = Nib[2]+dk[2]*Nib[6]+dk[1]*Nib[5];
329 Hi_y[8] = 0.;
330 Hi_y[9] = 3/2*(ek[3]*Nib[7]-ek[2]*Nib[6]);
331 Hi_y[10] = Nib[3]+dk[3]*Nib[7]+dk[2]*Nib[6];
332 Hi_y[11] = 0.;
333
334 Hi_xx[0] = 3/2*(ck[0]*Nib_x[4]-ck[3]*Nib_x[7]);
335 Hi_xx[1] = 0.;
336 Hi_xx[2] = -Nib_x[0]-bk[0]*Nib_x[4]-bk[3]*Nib_x[7];
337 Hi_xx[3] = 3/2*(ck[1]*Nib_x[5]-ck[0]*Nib_x[4]);
338 Hi_xx[4] = 0.;
339 Hi_xx[5] = -Nib_x[1]-bk[1]*Nib_x[5]-bk[0]*Nib_x[4];
340 Hi_xx[6] = 3/2*(ck[2]*Nib_x[6]-ck[1]*Nib_x[5]);
341 Hi_xx[7] = 0.;
342 Hi_xx[8] = -Nib_x[2]-bk[2]*Nib_x[6]-bk[1]*Nib_x[5];
343 Hi_xx[9] = 3/2*(ck[3]*Nib_x[7]-ck[2]*Nib_x[6]);
344 Hi_xx[10] = 0.;
345 Hi_xx[11] = -Nib_x[3]-bk[3]*Nib_x[7]-bk[2]*Nib_x[6];
346
347 Hi_yy[0] = 3/2*(ek[0]*Nib_y[4]-ek[3]*Nib_y[7]);
348 Hi_yy[1] = Nib_y[0]+dk[0]*Nib_y[4]+dk[3]*Nib_y[7];
349 Hi_yy[2] = 0.;
350 Hi_yy[3] = 3/2*(ek[1]*Nib_y[5]-ek[0]*Nib_y[4]);
351 Hi_yy[4] = Nib_y[1]+dk[1]*Nib_y[5]+dk[0]*Nib_y[4];
352 Hi_yy[5] = 0.;
353 Hi_yy[6] = 3/2*(ek[2]*Nib_y[6]-ek[1]*Nib_y[5]);
354 Hi_yy[7] = Nib_y[2]+dk[2]*Nib_y[6]+dk[1]*Nib_y[5];
355 Hi_yy[8] = 0.;
356 Hi_yy[9] = 3/2*(ek[3]*Nib_y[7]-ek[2]*Nib_y[6]);
357 Hi_yy[10] = Nib_y[3]+dk[3]*Nib_y[7]+dk[2]*Nib_y[6];
358 Hi_yy[11] = 0.;
359
360 Hi_xy[0] = 3/2*(ck[0]*Nib_y[4]-ck[3]*Nib_y[7]);
361 Hi_xy[1] = 0.;
362 Hi_xy[2] = -Nib_y[0]-bk[0]*Nib_y[4]-bk[3]*Nib_y[7];
363 Hi_xy[3] = 3/2*(ck[1]*Nib_y[5]-ck[0]*Nib_y[4]);
364 Hi_xy[4] = 0.;
365 Hi_xy[5] = -Nib_y[1]-bk[1]*Nib_y[5]-bk[0]*Nib_y[4];
366 Hi_xy[6] = 3/2*(ck[2]*Nib_y[6]-ck[1]*Nib_y[5]);
367 Hi_xy[7] = 0.;
368 Hi_xy[8] = -Nib_y[2]-bk[2]*Nib_y[6]-bk[1]*Nib_y[5];
```

```
369 Hi_xy[9] = 3/2*(ck[3]*Nib_y[7]-ck[2]*Nib_y[6]);
370 Hi_xy[10] = 0.;
371 Hi_xy[11] = -Nib_y[3]-bk[3]*Nib_y[7]-bk[2]*Nib_y[6];
372
373 Hi_yx[0] = 3/2*(ek[0]*Nib_x[4]-ek[3]*Nib_x[7]);
374 Hi_yx[1] = Nib_x[0]+dk[0]*Nib_x[4]+dk[3]*Nib_x[7];
375 Hi_yx[2] = 0.;
376 Hi_yx[3] = 3/2*(ek[1]*Nib_x[5]-ek[0]*Nib_x[4]);
377 Hi_yx[4] = Nib_x[1]+dk[1]*Nib_x[5]+dk[0]*Nib_x[4];
378 Hi_yx[5] = 0.;
379 Hi_yx[6] = 3/2*(ek[2]*Nib_x[6]-ek[1]*Nib_x[5]);
380 Hi_yx[7] = Nib_x[2]+dk[2]*Nib_x[6]+dk[1]*Nib_x[5];
381 Hi_yx[8] = 0.;
382 Hi_yx[9] = 3/2*(ek[3]*Nib_x[7]-ek[2]*Nib_x[6]);
383 Hi_yx[10] = Nib_x[3]+dk[3]*Nib_x[7]+dk[2]*Nib_x[6];
384 Hi_yx[11] = 0.;
385
386 for (int k=0; k<4; k++) {
387
388 h[0] = Ni[k];
389 h[4] = Ni[k];
390 h[6] = -Hi_x[0+3*k]*z_intPoint; h[7] = -Hi_y[0+3*k]*
       z_intPoint; h[8] = Ni[k];
391 h[9] = -Hi_x[1+3*k]*z_intPoint; h[10] = -Hi_y[1+3*k]*
       z_intPoint; h[11] = -Li[k];
392 h[12] = -Hi_x[2+3*k]*z_intPoint; h[13] = -Hi_y[2+3*k]*
       z_intPoint; h[14] = -Mi[k];
393 h[15] = Li[k]; h[16] = Mi[k];
394
395 b[0] = Ni_x[k]; b[1] = Ni_y[k];
396 b[12] = Ni_x[k]; b[13] = Ni_y[k];
397 b[18] = -Hi_xx[0+3*k]*z_intPoint; b[19] = -Hi_xy[0+3*k]*
       z_intPoint; b[21] = -Hi_yx[0+3*k]*z_intPoint; b[22] = -
       Hi_yy[0+3*k]*z_intPoint; b[24] = Ni_x[k]; b[25] = Ni_y[k];
398 b[27] = -Hi_xx[1+3*k]*z_intPoint; b[28] = -Hi_xy[1+3*k]*
       z_intPoint; b[30] = -Hi_yx[1+3*k]*z_intPoint; b[31] = -
       Hi_yy[1+3*k]*z_intPoint; b[33] = -Li_x[k]; b[34] = -Li_y[k
       ];
399 b[36] = -Hi_xx[2+3*k]*z_intPoint; b[37] = -Hi_xy[2+3*k]*
       z_intPoint; b[39] = -Hi_yx[2+3*k]*z_intPoint; b[40] = -
       Hi_yy[2+3*k]*z_intPoint; b[42] = -Mi_x[k]; b[43] = -Mi_y[k
       ];
400 b[45] = Li_x[k]; b[46] = Li_y[k]; b[48] = Mi_x[k]; b[49] =
       Mi_y[k];
401
402 h += 18;
403 b += 54;
404 }
```

```
405 //     matrix_print_real(el->intPoints[i]->shape.data(), 3,
        24, "%g ", "h");
406 //     matrix_print_real(el->intPoints[i]->shapeDer.data(), 9,
         24, "%g ", "b");
407 }
408 return 0;
409 }
410
411 int RQuadDef(Structure *s, Element* el, int pt, double*def) {
412 //     cout << endl << "RQuadDef BEGIN" << endl;
413 double u[24], ul[24];
414 memcpy(u, s->nodes[el->nodes[0]]->displacements, 6*sizeof(
        double));
415 memcpy(u+6, s->nodes[el->nodes[1]]->displacements, 6*sizeof(
        double));
416 memcpy(u+12, s->nodes[el->nodes[2]]->displacements, 6*sizeof(
        double));
417 memcpy(u+18, s->nodes[el->nodes[3]]->displacements, 6*sizeof(
        double));
418
419 double* trf = new double[24*24];
420 matrix_block_fill(el->trafoMat, 3, trf, 8);
421 matrix_mult(trf, 24, 24, u, 1, ul);
422
423 matrix_mult(el->intPoints[pt]->shape.data(), 3, 24, ul, 1,
        def);
424 return 0;
425 }
426
427
428 int RQuadDefGrad(Structure *s, Element* el, int pt, double*
        defgrad) {
429 //     cout << endl << "RQuadDefGrad BEGIN" << endl;
430 double u[24], ul[24];
431 memcpy(u, s->nodes[el->nodes[0]]->displacements, 6*sizeof(
        double));
432 memcpy(u+6, s->nodes[el->nodes[1]]->displacements, 6*sizeof(
        double));
433 memcpy(u+12, s->nodes[el->nodes[2]]->displacements, 6*sizeof(
        double));
434 memcpy(u+18, s->nodes[el->nodes[3]]->displacements, 6*sizeof(
        double));
435
436 double* trf = new double[24*24];
437 matrix_block_fill(el->trafoMat, 3, trf, 8);
438 matrix_mult(trf, 24, 24, u, 1, ul);
439 matrix_mult(el->intPoints[pt]->shapeDer.data(), 9, 24, ul, 1,
         defgrad);
440 return 0;
```

```
441 }
442
443 int RQuadStiffness(Structure* s, Element* el, int thePoint) {
444 //        std::cout << std::endl << "RQuadStiffness BEGIN " <<
        std::endl;
445 int nIntPoints = el->elementType->numIntPoints;
446 if (thePoint<0) thePoint=0;
447 if (thePoint>nIntPoints) thePoint = nIntPoints;
448 int nd = el->elementType->numDofsPerNode;
449 int mm = el->elementType->numDofNodes;
450 int nnn = mm*nd;
451 if (!el->stiffness) {
452 el->stiffness = new double[nnn*nnn];
453 }
454 matrix_init(el->stiffness, nnn, nnn);
455 double thickness = el->section->physicalData.data()[0]*el->
      physicalFactors.data()[0];
456 double fc = el->material->materialData.data()[3];
457 double ft = el->material->materialData.data()[4];
458
459 double* trf = new double[nnn*nnn];
460
461 double *matstiff = new double[9];
462
463 if (el->material->materialType->ident == "LINEAR_ELASTIC" ||
      (el->material->materialType->ident == "ELASTIC_FAILURE" &&
       !el->isStiffness)) {
464 el->material->ElasticFailure2DStiff(matstiff);
465 }
466 for (int i=0; i<9; i++) matstiff[i] *=el->density;
467 for (int i=0; i<9; i++) matstiff[i] *=el->materialFactors.
      data()[0];
468
469 double *bmatMem = new double[3*24];
470 double *bmatBen = new double[3*24];
471 double *tmpMem = new double[3*24];
472 double *tmpBen = new double[3*24];
473 double *sum = new double[24*24];
474
475 for (int i=0; i<3*24; i++) {
476 bmatMem[i] = 0;
477 bmatBen[i] = 0;
478 }
479 int kBegin = 0;
480 int kEnd = nIntPoints;
481 if (thePoint>0) {
482 kBegin = thePoint-1;
483 kEnd = thePoint;
484 }
```

```
485
486 for (int k=kBegin; k<kEnd; k++) {
487 double *b = el->intPoints[k]->shapeDer.data();
488 double *BMem = bmatMem;
489 double *BBen = bmatBen;
490 double *elmatMem = new double[24*24];
491 double *elmatBen = new double[24*24];
492
493 if (el->material->materialType->ident == "ELASTIC_FAILURE" &&
        el->isStiffness) {
494 double defgrad[9], strain[6], eps[3], sig[3];
495 RQuadDefGrad(s, el, k, defgrad);
496 el->section->Strain(defgrad, strain);
497 eps[0] = strain[0]; eps[1] = strain[1]; eps[2] = strain[3];
498 el->material->ElasticFailure2DStress(eps, sig);
499
500 //compute principle stresses
501 tmath::Matrix sigma(2, 2);
502 sigma(0,0)=sig[0]; sigma(0,1)=sig[2];
503 sigma(1,0)=sig[2]; sigma(1,1)=sig[1];
504 tmath::MatrixEigenSym eig = tmath::MatrixEigenSym (sigma);
505 tmath::Matrix psig = eig.Eigenvalues();
506
507 if (psig.data()[0] > ft || psig.data()[1] > ft) {
508 el->intPoints[k]->FailureTension = true;
509 }
510 if (psig.data()[0] < fc || psig.data()[1] < fc) {
511 el->intPoints[k]->FailureCompression = true;
512 }
513
514 el->material->ElasticFailure2DStiff(matstiff, el->intPoints[k
      ]->FailureCompression, el->intPoints[k]->FailureTension);
515 }
516
517 for (int i=0; i<4; i++) {
518
519 BMem[0] = b[0];
520 BMem[2] = b[1];
521 BMem[4] = b[13];
522 BMem[5] = b[12];
523 BMem[15] = b[45];
524 BMem[16] = b[49];
525 BMem[17] = b[46]+b[48];
526
527 BBen[6] = b[18];
528 BBen[7] = b[22];
529 BBen[8] = b[19]+b[21];
530 BBen[9] = b[27];
531 BBen[10] = b[31];
```

```
532 BBen[11] = b[28]+b[30];
533 BBen[12] = b[36];
534 BBen[13] = b[40];
535 BBen[14] = b[37]+b[39];
536
537 BMem += 18;
538 BBen += 18;
539 b += 54;
540
541 }
542
543 matrix_mult(matstiff, 3, 3, bmatMem, 24, tmpMem);
544 matrix_trans_mult(bmatMem, 3, 24, tmpMem, 24, elmatMem);
545 matrix_scal(elmatMem, nnn, nnn, thickness/2., elmatMem);
546
547 matrix_mult(matstiff, 3, 3, bmatBen, 24, tmpBen);
548 matrix_trans_mult(bmatBen, 3, 24, tmpBen, 24, elmatBen);
549 matrix_scal(elmatBen, nnn, nnn, thickness, elmatBen);
550
551 matrix_plus(elmatMem, elmatBen, 24, 24);
552 matrix_scal(elmatMem, nnn, nnn, el->intPoints[k]->weight,
      elmatMem);
553 matrix_plus(el->stiffness, elmatMem, 24, 24);
554 delete elmatMem; delete elmatBen;
555 }
556
557 matrix_block_fill(el->trafoMat, 3, trf, 8);
558 matrix_mult(el->stiffness, nnn, nnn, trf, nnn, sum);
559 matrix_trans_mult(trf, nnn, nnn, sum, nnn, el->stiffness);
560 //    matrix_print_real(el->stiffness, 24, 24, "%g ", "
      Element Stiffness");
561 el->isStiffness = true;
562 delete matstiff; delete bmatMem; delete bmatBen; delete
      tmpMem; delete tmpBen;
563 delete sum;
564 delete trf;
565 return 0;
566
567 }
568
569 int RQuadGeoStiffness(Structure* s, Element* el) {
570 //    std::cout << std::endl << "RQuadGeoStiffness BEGIN " <<
      std::endl;
571 int nIntPoints = el->elementType->numIntPoints;
572 int nd = el->elementType->numDofsPerNode;
573 int mm = el->elementType->numDofNodes;
574 int nnn = mm*nd;
575
```

```
576 double thickness = el->section->physicalData.data()[0]*el->
       physicalFactors.data()[0];
577
578 if (!el->geostiffness) {
579 el->geostiffness = new double[nnn*nnn];
580 }
581 matrix_init(el->geostiffness, nnn, nnn);
582 double* trf = new double[nnn*nnn];
583 double *bmat = new double[2*24];
584 double *tmp = new double[2*24];
585 double *elmat = new double[24*24];
586 double *sum = new double[24*24];
587 double *C = new double[2*2];
588
589 for (int i=0; i<2*2; i++) C[i] = 0;
590 for (int i=0; i<2*24; i++) bmat[i] = 0;
591 for (int k=0; k<nIntPoints; k++) {
592 double *b = el->intPoints[k]->shapeDer.data();
593 double *B = bmat;
594
595 double defgrad[9], strain[6], eps[3], sig[3];
596 RQuadDefGrad(s, el, k, defgrad);
597 el->section->Strain(defgrad, strain);
598 eps[0] = strain[0]; eps[1] = strain[1]; eps[2] = strain[3];
599 el->material->ElasticFailure2DStress(eps, sig);
600 C[0] = sig[0];
601 C[1] = sig[2];
602 C[2] = sig[2];
603 C[3] = sig[1];
604 for (int i=0; i<4; i++) {
605
606 B[0] = b[0]; B[1] = b[1];
607 B[2] = b[12]; B[3] = b[13];
608 B[4] = b[24]+b[18]+b[21]; B[5] = b[25]+b[19]+b[22];
609 B[6] = b[33]+b[27]+b[30]; B[7] = b[34]+b[28]+b[31];
610 B[8] = b[42]+b[36]+b[39]; B[9] = b[43]+b[37]+b[40];
611 B[10] = b[45]+b[48]; B[11] = b[46]+b[49];
612 B+=12;
613 b+=54;
614 }
615
616 matrix_mult(C, 2, 2, bmat, 24, tmp);
617 matrix_trans_mult(bmat, 2, 24, tmp, 24, elmat);
618
619 matrix_scal(elmat, nnn, nnn, el->intPoints[k]->weight, elmat)
       ;
620 matrix_plus(el->geostiffness, elmat, 24, 24);
621 }
622
```

```
623 matrix_scal(el->geostiffness, nnn, nnn, thickness/2., el->
      geostiffness); // divided by 2 for 3x3 Gauss x2
624 matrix_block_fill(el->trafoMat, 3, trf, 8);
625 matrix_mult(el->geostiffness, nnn, nnn, trf, nnn, sum);
626 matrix_trans_mult(trf, nnn, nnn, sum, nnn, el->geostiffness);
627 //    matrix_print_real(el->geostiffness, 24, 24, "%g ", "
      Geometrical Element Stiffness");
628 delete bmat; delete tmp;
629 delete elmat; delete sum;
630 delete trf; delete C;
631 return 0;
632 }
633
634 int RQuadMass(Element* el) {
635 //    std::cout << std::endl << "RQuadMass BEGIN " << std::
      endl;
636 if (!el->mass) {
637 int mm = el->elementType->numDofNodes;
638 int nd = el->elementType->numDofsPerNode;
639 int nnn = mm*nd;
640 el->mass = new double[nnn*nnn];
641 }
642 int nNodes = el->elementType->numDofNodes;
643 int nDN = el->elementType->numDofsPerNode;
644 int nDof = nDN*nNodes;
645 int nIntPoints = el->elementType->numIntPoints;
646 double rho = el->material->materialData.data()[2];
647 double w;
648 double thickness = el->section->physicalData.data()[0]*el->
      physicalFactors.data()[0];
649
650 for (int j=0; j<nDof*nDof; j++) el->mass[j] = 0.;
651 for (int i=0; i<nIntPoints; i++) {
652 tmath::Matrix sum = el->intPoints[i]->shape.Transpose()
653 *el->intPoints[i]->shape;
654 w = rho*el->intPoints[i]->weight*thickness/2;
655 matrix_add_scal(el->mass, sum.data(), nDof, nDof, 1., w, el->
      mass);
656 }
657 //    matrix_print_real(el->mass, 24, 24, "%g ", "elmass");
658 return 0;
659 }
660
661 int RQuadResForceStiffness(Structure *s, Element* el) {
662 //       std::cout << std::endl << "RQuadResForceStiffness
      BEGIN " << std::endl;
663 if (el->resforce) delete el->resforce;
664 el->resforce=new double[24];
665 double disp[24], *d;
```

```
666 for (int i=0; i<4; i++) {
667 d = s->nodes[el->nodes[i]]->displacements;
668 memcpy(disp+6*i, d, 6*sizeof(double));
669 }
670 // if (el->material->materialType->ident == "ELASTIC_FAILURE
    ") {
671 //    for (int i=0; i<el->elementType->numIntPoints; i++) {
672 //       RQuadStiffness(s, el, i);
673 //    }
674 // }
675 matrix_mult(el->stiffness, 24, 24, disp, 1, el->resforce);
676 return 0;
677 }
678
679 int RQuadStress(Structure *s, Element* el) {
680 //       std::cout << std::endl << "RQuadStress BEGIN " <<
    std::endl;
681 double defgrad[9], strain[6], stress[9], tmp[9], eps[3], sig
    [3];
682 int npt = el->elementType->numIntPoints + el->elementType->
    numStressPoints;
683 //    cout << endl << "numIntPoints = " << el->elementType->
    numIntPoints << endl << "numStressPoints = " << el->
    elementType->numStressPoints << endl;
684 for (int i=0; i<npt; i++) {
685 //       cout << endl << "Point No: " << i << endl;
686 RQuadDefGrad(s, el, i, defgrad);
687 //       matrix_print_real(defgrad, 3, 3, "%g ", "defgrad");
688 el->section->Strain(defgrad, strain);
689 eps[0] = strain[0]; eps[1] = strain[1]; eps[2] = strain[3];
690 //       matrix_print_real(strain, 6, 1, "%g ", "strain");
691 //el->material->LinearElastic2DStress(eps, sig);
692 if (el->material->materialType->ident == "LINEAR_ELASTIC") {
    el->material->LinearElastic2DStress(eps, sig);}
693 if (el->material->materialType->ident == "ELASTIC_FAILURE") {
    el->material->ElasticFailure2DStress(eps, sig);}
694 for (int i=0; i<3; i++) sig[i] *=el->materialFactors.data()
    [0];
695 matrix_init(stress, 3, 3);
696 stress[0] = sig[0];
697 stress[1] = sig[2];
698 stress[3] = sig[2];
699 stress[4] = sig[1];
700 //       matrix_print_real(stress, 3, 3, "%g ", "Stress in
    RQuadStress");
701 matrix_trans_mult(el->trafoMat, 3, 3, stress, 3, tmp);
702 matrix_mult(tmp, 3, 3, el->trafoMat, 3, stress);
703 matrix_init(el->intPoints[i]->stress.data(), 6, 1);
704 el->intPoints[i]->stress.data()[0] = stress[0];
```

```
705 el->intPoints[i]->stress.data()[1] = stress[4];
706 el->intPoints[i]->stress.data()[2] = stress[8];
707 el->intPoints[i]->stress.data()[3] = stress[1];
708 el->intPoints[i]->stress.data()[4] = stress[2];
709 el->intPoints[i]->stress.data()[5] = stress[5];
710 }
711 return 0;
712 }
713
714 int RQuadRoutine(Structure *s, Element* el, elemAction action
       , void* View, int thePoint) {
715 int error=-1;
716 switch (action) {
717 case elemIntPoints:
718 error = RQuadIntPoints(s, el);
719 break;
720 case elemStiffness:
721 error = RQuadStiffness(s, el, thePoint);
722 break;
723 case elemGeoStiffness:
724 error = RQuadGeoStiffness(s, el);
725 break;
726 case elemMass:
727 error = RQuadMass(el);
728 break;
729 case elemForce:
730 //        error = RQuadForce(s, el);
731 break;
732 case elemResForce:
733 error = RQuadStiffness(s, el, 0);
734 error = RQuadResForceStiffness(s, el);
735 //        error = RQuadResForce(s, el);
736 break;
737 case elemResForceStiffness:
738 error = RQuadResForceStiffness(s, el);
739 break;
740 case elemStress:
741 error = RQuadStress(s, el);
742 break;
743 case elemEnergy:
744 //        error = RQuadEnergy(s, el);
745 break;
746 case elemKinetic:
747 //        error = RQuadKinetic(s, el);
748 break;
749 case elemMomentum:
750 //        error = RQuadMomentum(s, el);
751 break;
752 default:
```

```
753 break;
754 }
755 return error;
756 }
757
758 tmath::Matrix RQuadDraw3D(Structure *s, Element* el, double
       def) {
759 return RQuad3D(s, el, def, 0, 1, 2, 3);
760 }
761
762 tmath::Matrix RQuad3D(Structure *s, Element* el, double def,
       int n1, int n2, int n3, int n4) {
763 double d1[3], d2[3], d3[3], d4[3], dd1[3], dd2[3], dd3[3],
       dd4[3];
764 memcpy(d1, s->nodes[el->nodes[n1]]->coordinates, 3*sizeof(
       double));
765 memcpy(d2, s->nodes[el->nodes[n2]]->coordinates, 3*sizeof(
       double));
766 memcpy(d3, s->nodes[el->nodes[n3]]->coordinates, 3*sizeof(
       double));
767 memcpy(d4, s->nodes[el->nodes[n4]]->coordinates, 3*sizeof(
       double));
768 double e1[3], e2[3], e3[3], e4[3], e5[3], e6[3], e7[3];
769
770 if (def) {
771 matrix_add_scal(d1, s->nodes[el->nodes[0]]->displacements, 3,
        1, 1, def, d1);
772 matrix_add_scal(d2, s->nodes[el->nodes[1]]->displacements, 3,
        1, 1, def, d2);
773 matrix_add_scal(d3, s->nodes[el->nodes[2]]->displacements, 3,
        1, 1, def, d3);
774 matrix_add_scal(d4, s->nodes[el->nodes[3]]->displacements, 3,
        1, 1, def, d4);
775 }
776
777 double thick = el->section->physicalData.data()[0]/2.*el->
       physicalFactors.data()[0];
778
779 matrix_sub(d1, d2, 3, 1, e1);
780 matrix_sub(d1, d3, 3, 1, e2);
781 vector_cross(e1, e2, e3);
782 double l;
783 vector_length(e3, 3, &l);
784 matrix_scal(e3, 3, 1, 1./l, e3);
785
786 matrix_add_scal(d1, e3, 3, 1, 1, thick, d1);
787 matrix_add_scal(d2, e3, 3, 1, 1, thick, d2);
788 matrix_add_scal(d3, e3, 3, 1, 1, thick, d3);
789 matrix_add_scal(d4, e3, 3, 1, 1, thick, d4);
```

```
790 matrix_add_scal(d1, e3, 3, 1, 1, -2*thick, dd1);
791 matrix_add_scal(d2, e3, 3, 1, 1, -2*thick, dd2);
792 matrix_add_scal(d3, e3, 3, 1, 1, -2*thick, dd3);
793 matrix_add_scal(d4, e3, 3, 1, 1, -2*thick, dd4);
794
795 matrix_sub(d1, dd1, 3, 1, e1);
796 matrix_sub(d1, d2, 3, 1, e2);
797 vector_cross(e1, e2, e4);
798 vector_length(e4, 3, &l);
799 matrix_scal(e4, 3, 1, 1./l, e4);
800
801 matrix_sub(d2, dd2, 3, 1, e1);
802 matrix_sub(d2, d3, 3, 1, e2);
803 vector_cross(e1, e2, e5);
804 vector_length(e5, 3, &l);
805 matrix_scal(e5, 3, 1, 1./l, e5);
806
807 matrix_sub(d3, dd3, 3, 1, e1);
808 matrix_sub(d3, d4, 3, 1, e2);
809 vector_cross(e1, e2, e6);
810 vector_length(e6, 3, &l);
811 matrix_scal(e6, 3, 1, 1./l, e6);
812
813 matrix_sub(d4, dd4, 3, 1, e1);
814 matrix_sub(d4, d1, 3, 1, e2);
815 vector_cross(e1, e2, e7);
816 vector_length(e7, 3, &l);
817 matrix_scal(e7, 3, 1, 1./l, e7);
818
819 tmath::Matrix m(11,36);
820 tmath::Matrix poly(11,1);
821 unsigned int i=0;
822 unsigned int r[8], g[8], b[8], a[8];
823 for (unsigned int k=0; k<8; k++) {
824 r[k] = el->red[0], g[k] = el->green[0], b[k] = el->blue[0], a
      [k] = el->alpha[0];
825 }
826 if (el->red.size()>1) {
827 for (unsigned int k=1; k<8; k++) {
828 r[k] = el->red[k], g[k] = el->green[k], b[k] = el->blue[k], a
      [k] = el->alpha[k];
829 }
830 }
831 // Top
832 poly<<d1[0], d1[1], d1[2], e3[0], e3[1], e3[2], r[0], g[0], b
      [0], a[0], el->key; m.SetCols(poly, i); i++;
833 poly<<d2[0], d2[1], d2[2], e3[0], e3[1], e3[2], r[1], g[1], b
      [1], a[1], el->key; m.SetCols(poly, i); i++;
```

```
834 poly<<d3[0], d3[1], d3[2], e3[0], e3[1], e3[2], r[2], g[2], b
        [2], a[2], el->key; m.SetCols(poly, i); i++;
835
836 poly<<d1[0], d1[1], d1[2], e3[0], e3[1], e3[2], r[0], g[0], b
        [0], a[0], el->key; m.SetCols(poly, i); i++;
837 poly<<d3[0], d3[1], d3[2], e3[0], e3[1], e3[2], r[2], g[2], b
        [2], a[2], el->key; m.SetCols(poly, i); i++;
838 poly<<d4[0], d4[1], d4[2], e3[0], e3[1], e3[2], r[2], g[3], b
        [3], a[3], el->key; m.SetCols(poly, i); i++;
839
840 // Bottom
841 poly<<dd1[0], dd1[1], dd1[2], -e3[0], -e3[1], -e3[2], r[4], g
        [4], b[4], a[4], el->key; m.SetCols(poly, i); i++;
842 poly<<dd3[0], dd3[1], dd3[2], -e3[0], -e3[1], -e3[2], r[6], g
        [6], b[6], a[6], el->key; m.SetCols(poly, i); i++;
843 poly<<dd2[0], dd2[1], dd2[2], -e3[0], -e3[1], -e3[2], r[5], g
        [5], b[5], a[5], el->key; m.SetCols(poly, i); i++;
844
845 poly<<dd4[0], dd4[1], dd4[2], -e3[0], -e3[1], -e3[2], r[7], g
        [7], b[7], a[7], el->key; m.SetCols(poly, i); i++;
846 poly<<dd3[0], dd3[1], dd3[2], -e3[0], -e3[1], -e3[2], r[6], g
        [6], b[6], a[6], el->key; m.SetCols(poly, i); i++;
847 poly<<dd1[0], dd1[1], dd1[2], -e3[0], -e3[1], -e3[2], r[4], g
        [4], b[4], a[4], el->key; m.SetCols(poly, i); i++;
848
849 // Side 1
850 poly<<d1[0], d1[1], d1[2], e4[0], e4[1], e4[2], r[0], g[0], b
        [0], a[0], el->key; m.SetCols(poly, i); i++;
851 poly<<dd1[0], dd1[1], dd1[2], e4[0], e4[1], e4[2], r[4], g
        [4], b[4], a[4], el->key; m.SetCols(poly, i); i++;
852 poly<<d2[0], d2[1], d2[2], e4[0], e4[1], e4[2], r[1], g[1], b
        [1], a[1], el->key; m.SetCols(poly, i); i++;
853
854 poly<<dd1[0], dd1[1], dd1[2], e4[0], e4[1], e4[2], r[4], g
        [4], b[4], a[4], el->key; m.SetCols(poly, i); i++;
855 poly<<dd2[0], dd2[1], dd2[2], e4[0], e4[1], e4[2], r[5], g
        [5], b[5], a[5], el->key; m.SetCols(poly, i); i++;
856 poly<<d2[0], d2[1], d2[2], e4[0], e4[1], e4[2], r[1], g[1], b
        [1], a[1], el->key; m.SetCols(poly, i); i++;
857
858 // Side 2
859 poly<<d2[0], d2[1], d2[2], e5[0], e5[1], e5[2], r[1], g[1], b
        [1], a[1], el->key; m.SetCols(poly, i); i++;
860 poly<<dd2[0], dd2[1], dd2[2], e5[0], e5[1], e5[2], r[5], g
        [5], b[5], a[5], el->key; m.SetCols(poly, i); i++;
861 poly<<d3[0], d3[1], d3[2], e5[0], e5[1], e5[2], r[2], g[2], b
        [2], a[2], el->key; m.SetCols(poly, i); i++;
862
```

```
863 poly<<dd2[0], dd2[1], dd2[2], e5[0], e5[1], e5[2], r[5], g
        [5], b[5], a[5], el->key; m.SetCols(poly, i); i++;
864 poly<<dd3[0], dd3[1], dd3[2], e5[0], e5[1], e5[2], r[6], g
        [6], b[6], a[6], el->key; m.SetCols(poly, i); i++;
865 poly<<d3[0], d3[1], d3[2], e5[0], e5[1], e5[2], r[2], g[2], b
        [2], a[2], el->key; m.SetCols(poly, i); i++;
866
867 // Side 3
868 poly<<d3[0], d3[1], d3[2], e6[0], e6[1], e6[2], r[2], g[2], b
        [2], a[2], el->key; m.SetCols(poly, i); i++;
869 poly<<dd3[0], dd3[1], dd3[2], e6[0], e6[1], e6[2], r[6], g
        [6], b[6], a[6], el->key; m.SetCols(poly, i); i++;
870 poly<<d4[0], d4[1], d4[2], e6[0], e6[1], e6[2], r[3], g[3], b
        [3], a[3], el->key; m.SetCols(poly, i); i++;
871
872 poly<<dd3[0], dd3[1], dd3[2], e6[0], e6[1], e6[2], r[6], g
        [6], b[6], a[6], el->key; m.SetCols(poly, i); i++;
873 poly<<dd4[0], dd4[1], dd4[2], e6[0], e6[1], e6[2], r[7], g
        [7], b[7], a[7], el->key; m.SetCols(poly, i); i++;
874 poly<<d4[0], d4[1], d4[2], e6[0], e6[1], e6[2], r[3], g[3], b
        [3], a[3], el->key; m.SetCols(poly, i); i++;
875
876 // Side 4
877 poly<<d4[0], d4[1], d4[2], e7[0], e7[1], e7[2], r[3], g[3], b
        [3], a[3], el->key; m.SetCols(poly, i); i++;
878 poly<<dd4[0], dd4[1], dd4[2], e7[0], e7[1], e7[2], r[7], g
        [7], b[7], a[7], el->key; m.SetCols(poly, i); i++;
879 poly<<d1[0], d1[1], d1[2], e6[0], e6[1], e6[2], r[0], g[0], b
        [0], a[0], el->key; m.SetCols(poly, i); i++;
880
881 poly<<dd4[0], dd4[1], dd4[2], e6[0], e6[1], e6[2], r[7], g
        [7], b[7], a[7], el->key; m.SetCols(poly, i); i++;
882 poly<<dd1[0], dd1[1], dd1[2], e6[0], e6[1], e6[2], r[4], g
        [4], b[4], a[4], el->key; m.SetCols(poly, i); i++;
883 poly<<d1[0], d1[1], d1[2], e6[0], e6[1], e6[2], r[0], g[0], b
        [0], a[0], el->key; m.SetCols(poly, i); i++;
884
885 return m;
886 }
887
888 }
```

Anhang C

Quellcode Elastic Failure

Folgender Code stellt einen Ausschnitt des files Material.cpp dar, welcher relevant für das implementierte Materialmodell **ElasticFailure** ist.

Material.cpp

```
1  void Material::ElasticFailure2DStiff(double *stiff, const
       bool & FailureCompression, const bool & FailureTension) {
2  double E = materialData.data()[0];
3  if (FailureCompression == true && materialType->ident == "
       ELASTIC_FAILURE") {E = 0.0*E;}
4  if (FailureTension == true && materialType->ident == "
       ELASTIC_FAILURE") {E = 0.0*E;}
5  bool planeStrain = materialData.data()[1] < 0;
6  double nu = fabs(materialData.data()[1]);
7  double lam = E/(1-nu*nu);
8  double G = E/2/(1+nu);
9  double EL = E/(1+nu)/(1-2*nu);
10 for (int i=0; i<9; i++) stiff[i] = 0;
11 if (planeStrain) {
12 stiff[0] = EL*(1-nu); stiff[1] = nu*EL;
13 stiff[3] = nu*EL; stiff[4] = EL*(1-nu);
14 }
15 else {
16 stiff[0] = lam; stiff[3] = nu*lam;
17 stiff[1] = nu*lam; stiff[4] = lam;
18 }
19 stiff[8] = G;
20 }
```

Anhang D

Anwendungsbeispiele Kapitel 3 und Kapitel 4

Abschließend werden einige .tng files mit Anwendungs- und Berechnungsbeispielen zur Verfügung gestellt.

auch alle Testfiles aus Kapitel 4 (Wandscheibe) einfügen

D.1 Beispiel 3.1 Membrantragwirkung, 3.2 Plattentragwirkung, 3.3 Eigenfrequenzen und Eigenschwingungsformen

cantilever_rquad.tng

```
1
2  -- Create and new FE structure
3  structure=fem.Structure("Cantilever Beam")
4
5  --Element dImensions
6  a = 12
7  b = 12
8
9  load_dir = 1
10 --0 = x
11 --1 = y
12 --2 = z
13
14 -- Define node IDs and coordinates
15 nodes = tmath.Matrix({
16 {10, 0, 0, 1}, -- reference node
17 {11, 0, 0, 0},
18 {12, a, 0, 0},
19 {13, a, b, 0},
20 {14, 0, b, 0},
21 {15, 2*a, 0, 0},
22 {16, 2*a, b, 0},
23 {17, 3*a, 0, 0},
24 {18, 3*a, b, 0},
25 {19, 4*a, 0, 0},
26 {20, 4*a, b, 0},
27
28 })
```

```
structure:AddNodes(nodes)

-- Define support conditions and fix reference node
structure:GetNode(10):SetAvailDof(tmath.Matrix(
   tmath.ZeroMatrix(1,6)))
structure:GetNode(11):SetAvailDof(tmath.Matrix({{0, 0, 0, 0,
   0, 0}}))
structure:GetNode(14):SetAvailDof(tmath.Matrix({{0, 0, 0, 0,
   0, 0}}))

-- Define cross sections
t = 1
cyan = tmath.Matrix({{100, 255, 255, 255}})
blue = tmath.Matrix({{100, 200, 200, 200}})
s1 = structure:AddSection(101, "SHELL", 0)
s1:SetColor(cyan)
s1:SetData(tmath.Matrix({{t}}))

s2 = structure:AddSection(102, "SHELL", 0)
s2:SetColor(blue)
s2:SetData(tmath.Matrix({{t}}))

-- Define material
m=structure:AddMaterial(101, "LINEAR_ELASTIC")
m:SetData(tmath.Matrix({{30000, 0., 1}}))

-- Define elements

elements1 = tmath.Matrix({
{11, 11, 12, 13, 14},
--    {12, 12, 15, 16, 13},
{13, 15, 17, 18, 16},
--    {14, 17, 19, 20, 18},
})

elements2 = tmath.Matrix({
--    {11, 11, 12, 13, 14},
{12, 12, 15, 16, 13},
--    {13, 15, 17, 18, 16},
{14, 17, 19, 20, 18},
})


structure:AddElements("RQUAD", 101, 101, elements1)
structure:AddElements("RQUAD", 101, 102, elements2)

print("Elements", structure:GetNumberOfElements())

g = graph3d.Graph3D("Shell")
```

```
g:Clear()
tris = structure:Draw()
g:Triangles(tris)
g:Autoscale()
g:Zoom(0.7)

-- load
F = -20

-- Find global DOFs and assemble stiffness
nd = structure:GlobalDof()
print("nd",nd)
structure:Print()
K = structure:SparseStiffness()
print("K",K)
k=K:Expand()
print("k",k)

-- Construct a load vector
Loads=structure:GetAllDisplacements()
print("Loads", Loads)

-- Applying vertical loads
Loads[{9, load_dir}] = F
Loads[{10, load_dir}] = F
print("Loads2", Loads)

P = structure:ToDofDisplacements(Loads)
print("P", P)

-- Solve for displacements and assign to structure
U=K:Solve(P)
print("K", K)
print("U", U)

U1=structure:ToAllDisplacements(U)
print("U1", U1)
structure:SetAllDisplacements(U1)
print("displacements set")
--
-- Compute section forces/moments
FS = structure:ElementForce()
print("FS",FS)
str = structure:ElementStress(1)

print("k",k)
print("Loads2", Loads)
tipnode = structure:GetNodeIndex(20)
disp = structure:GetAllDisplacements()
```

```
124
125 u = disp[{tipnode,0}]
126 v = disp[{tipnode,1}]
127 w = disp[{tipnode,2}]
128
129 print("u", u)
130 print("v", v)
131 print("w", w)
132 print("U", U)
133
134 d = graph3d.Graph3D("Displacements")
135 trim = structure:Draw(10)
136 d:Triangles(trim)
137 d:Autoscale()
138 d:Zoom(0.7)
139 disp = structure:DrawNodes(0.03,10)
140 d:Triangles(disp)
141 d:Render()
142
143 --Test Mass Matrix
144 M = structure:SparseMass()
145 --Compute first xxx eigenvalues and mode shapes
146 val,vec = K:Eigen(M, 20)
147 freq = val:CW():Sqrt()/2/math.pi
148 print("freq", freq)
149
150 -- Show first N mode shape
151
152 N_shapes = 0
153 g={}
154 for i=0,N_shapes do
155 shape = vec:GetCols(i)
156 structure:SetDofDisplacements(shape)
157 g[i] = graph3d.Graph3D("Mode "..i, 255, 255, 255, 255)
158 g[i]:Rotate(-60, 1, 0, 1)
159 plot = structure:Draw(200)
160 g[i]:Triangles(plot)
161 g[i]:Autoscale()
162 g[i]:Render()
163 end
```

D.2 Beispiel 3.4 Stabilität - Beulen

buckling_rquad.tng

```
1
```

```
b = 1
h = 12
L = 48
E = 30000

structure = fem.Structure("Buckling Beam")
nodes = tmath.Matrix({
{0, 0, 0, 0},
{1, 1*L/10, 0, 0},
{2, 2*L/10, 0, 0},
{3, 3*L/10, 0, 0},
{4, 4*L/10, 0, 0},
{5, 5*L/10, 0, 0},
{6, 6*L/10, 0, 0},
{7, 7*L/10, 0, 0},
{8, 8*L/10, 0, 0},
{9, 9*L/10, 0, 0},
{10, L, 0, 0},
{11, 0, h, 0},
{12, 1*L/10, h, 0},
{13, 2*L/10, h, 0},
{14, 3*L/10, h, 0},
{15, 4*L/10, h, 0},
{16, 5*L/10, h, 0},
{17, 6*L/10, h, 0},
{18, 7*L/10, h, 0},
{19, 8*L/10, h, 0},
{20, 9*L/10, h, 0},
{21, L, h, 0},
})

structure:AddNodes(nodes)
structure:SetAvailDof(tmath.Matrix({{1, 1, 1, 1, 1, 1}}))
structure:GetNode(0):SetAvailDof(tmath.Matrix({{0, 0, 0, 0,
   0, 0}}))
structure:GetNode(11):SetAvailDof(tmath.Matrix({{0, 0, 0, 0,
   0, 0}}))

cyan = tmath.Matrix({{100, 255, 255, 255}})
blue = tmath.Matrix({{100, 200, 200, 200}})

s1 = structure:AddSection(101, "SHELL", 1)
s1:SetData(tmath.Matrix({{b}}))
s1:SetColor(cyan)

s2 = structure:AddSection(102, "SHELL", 1)
s2:SetData(tmath.Matrix({{b}}))
s2:SetColor(blue)

```

```
49 m=structure:AddMaterial(201, "LINEAR_ELASTIC")
50 m:SetData(tmath.Matrix({{E, 0.0, 1}}))
51
52 elements1 =  tmath.Matrix({
53 {1, 0, 1, 12, 11},
54 --    {2, 1, 2, 13, 12},
55 {3, 2, 3, 14, 13},
56 --    {4, 3, 4, 15, 14},
57 {5, 4, 5, 16, 15},
58 --    {6, 5, 6, 17, 16},
59 {7, 6, 7, 18, 17},
60 --    {8, 7, 8, 19, 18},
61 {9, 8, 9, 20, 19},
62 --    {10, 9, 10, 21, 20},
63 })
64
65 elements2 =  tmath.Matrix({
66 --    {1, 0, 1, 12, 11},
67 {2, 1, 2, 13, 12},
68 --    {3, 2, 3, 14, 13},
69 {4, 3, 4, 15, 14},
70 --    {5, 4, 5, 16, 15},
71 {6, 5, 6, 17, 16},
72 --    {7, 6, 7, 18, 17},
73 {8, 7, 8, 19, 18},
74 --    {9, 8, 9, 20, 19},
75 {10, 9, 10, 21, 20},
76 })
77
78 structure:AddElements("RQUAD", 201, 101, elements1)
79 structure:AddElements("RQUAD", 201, 102, elements2)
80
81 --structure:SetSection(101)
82
83 nd = structure:GlobalDof()
84 -- Compute stiffness matrix in stress free state
85 K = structure:SparseStiffness()
86 k = K:Expand()
87
88 -- Check static load in y-direction
89 F1 = structure:GetAllDisplacements()
90
91 F1:SetZero()
92 F1[{10, 2}] = 0.5
93 F1[{21, 2}] = 0.5
94 F = structure:ToDofDisplacements(F1)
95 U = K:Solve(F)
96 U1 = structure:ToAllDisplacements(U)
97 structure:SetDofDisplacements(U)
```

```
98
99  print("Static deflection:", U1[{10,2}])
100 print("Static deflection:", U1[{21,2}])
101
102 -- Compute theoretical static deflection
103 EI = b^3*h/12*E
104 print("theoretical static deflection:", L^3/3/EI)
105
106
107 -- Compute stiffness matrix in stressed state (unit load in x
    -direction)
108 F1:SetZero()
109 F1[{10, 0}] = -0.5
110 F1[{21, 0}] = -0.5
111 F = structure:ToDofDisplacements(F1)
112 U2 = K:Solve(F)
113 structure:SetDofDisplacements(U2)
114 KG = structure:SparseGeoStiffness()
115 kg = KG:Expand()
116
117 eigen = tmath.MatrixEigenSym(kg, k)
118 val = eigen:Eigenvalues()
119 print("val", val)
120 print("val1", -val:CW():Pow(-1))
121 -- Solve eigenvalue problem for buckling load
122 ev, vec = K:EigenBuckling(KG, 1)
123 print("FE Buckling load:", ev[0])
124
125 -- Compute theoretical Euler buckling load
126 print("Theoretical Buckling load:", math.pi^2*EI/(2*L)^2)
127
128 -- Visualize first buckling shape
129 structure:SetDofDisplacements(vec:GetCols(0))
130 v = graph3d.Graph3D("Buckling Shape")
131 tri = structure:Draw(20)
132 v:Triangles(tri)
133 v:Autoscale()
134 v:Render()
```

D.3 Beispiel 4.2 Mauerwerkswand unter reiner Scheibenbeanspruchung

push_wandscheibe.tng

```
1 --push over analysis of a shear wall
2
```

```
F = 2e+4
DF = 0.5e+3

-- Number of load steps
N = 120
ki = 5 --iterations per loadstep

--Dimensions of wall
a = 6
b = 3.6
t = 0.3

--Direction of Load 0 = x
load_dir = 0

--Graph number of data rows
PrintN = 590

-- Create and new FE structure
structure=fem.Structure("shell")

-- Define node IDs and coordinates
nodes = tmath.Matrix({
{10, 0, 0, 1}, -- reference node
{11, -2*a/4, -2*b/4, 0},
{12, -a/4, -2*b/4, 0},
{13, 0, -2*b/4, 0},
{14, a/4, -2*b/4, 0},
{15, 2*a/4, -2*b/4, 0},
{16, -2*a/4, -b/4, 0},
{17, -a/4, -b/4, 0},
{18, 0, -b/4, 0},
{19, a/4, -b/4, 0},
{20, 2*a/4, -b/4, 0},
{21, -2*a/4, 0, 0},
{22, -a/4, 0, 0},
{23, 0, 0, 0},
{24, a/4, 0, 0},
{25, 2*a/4, 0, 0},
{26, -2*a/4, b/4, 0},
{27, -a/4, b/4, 0},
{28, 0, b/4, 0},
{29, a/4, b/4, 0},
{30, 2*a/4, b/4, 0},
{31, -2*a/4, 2*b/4, 0},
{32, -a/4, 2*b/4, 0},
{33, 0, 2*b/4, 0},
{34, a/4, 2*b/4, 0},
{35, 2*a/4, 2*b/4, 0},
```

```
52 })
53 structure:AddNodes(nodes)
54
55 tipnode1 = structure:GetNodeIndex(31)
56 tipnode2 = structure:GetNodeIndex(32)
57 tipnode3 = structure:GetNodeIndex(33)
58 tipnode4 = structure:GetNodeIndex(34)
59 tipnode5 = structure:GetNodeIndex(35)
60 centernode = structure:GetNodeIndex(23)
61 basenode = structure:GetNodeIndex(13)
62
63 -- Define support conditions and fix reference node
64 structure:GetNode(10):SetAvailDof(tmath.Matrix(
      tmath.ZeroMatrix(1,6)))
65 structure:GetNode(11):SetAvailDof(tmath.Matrix(
      tmath.ZeroMatrix(1,6)))
66 structure:GetNode(12):SetAvailDof(tmath.Matrix(
      tmath.ZeroMatrix(1,6)))
67 structure:GetNode(13):SetAvailDof(tmath.Matrix(
      tmath.ZeroMatrix(1,6)))
68 structure:GetNode(14):SetAvailDof(tmath.Matrix(
      tmath.ZeroMatrix(1,6)))
69 structure:GetNode(15):SetAvailDof(tmath.Matrix(
      tmath.ZeroMatrix(1,6)))
70
71 -- Define cross sections
72 cyan = tmath.Matrix({{100, 255, 255, 255}})
73 blue = tmath.Matrix({{100, 200, 200, 200}})
74
75 s1 = structure:AddSection(101, "SHELL", 0)
76 s1:SetColor(cyan)
77 s1:SetData(tmath.Matrix({{t}}))
78
79 s2 = structure:AddSection(102, "SHELL", 0)
80 s2:SetColor(blue)
81 s2:SetData(tmath.Matrix({{t}}))
82
83 -- Define material
84 m=structure:AddMaterial(101, "ELASTIC_FAILURE")
85 m:SetData(tmath.Matrix({{1.6e+9, 0.2 , 1, -5e+6, 2e+5 }}))
86
87 -- linear elastic
88 -- m=structure:AddMaterial(101, "LINEAR_ELASTIC")
89 -- m:SetData(tmath.Matrix({{1.6e+9, 0.2 , 1700}}))
90
91
92 -- Define elements
93
94 elements1 = tmath.Matrix({
```

```
{11, 11, 12, 17, 16},
--    {12, 12, 13, 18, 17},
{13, 13, 14, 19, 18},
--    {14, 14, 15, 20, 19},
--    {15, 16, 17, 22, 21},
{16, 17, 18, 23, 22},
--    {17, 18, 19, 24, 23},
{18, 19, 20, 25, 24},
{19, 21, 22, 27, 26},
--    {20, 22, 23, 28, 27},
{21, 23, 24, 29, 28},
--    {22, 24, 25, 30, 29},
--    {23, 26, 27, 32, 31},
{24, 27, 28, 33, 32},
--    {25, 28, 29, 34, 33},
{26, 29, 30, 35, 34},
})

elements2 = tmath.Matrix({
--    {11, 11, 12, 17, 16},
{12, 12, 13, 18, 17},
--    {13, 13, 14, 19, 18},
{14, 14, 15, 20, 19},
{15, 16, 17, 22, 21},
--    {16, 17, 18, 23, 22},
{17, 18, 19, 24, 23},
--    {18, 19, 20, 25, 24},
--    {19, 21, 22, 27, 26},
{20, 22, 23, 28, 27},
--    {21, 23, 24, 29, 28},
{22, 24, 25, 30, 29},
{23, 26, 27, 32, 31},
--    {24, 27, 28, 33, 32},
{25, 28, 29, 34, 33},
--    {26, 29, 30, 35, 34},
})

structure:AddElements("RQUAD", 101, 101, elements1)

structure:AddElements("RQUAD", 101, 102, elements2)

ndof = structure:GlobalDof()
topnode = structure:GetNodeIndex(tipnode3)
basepoint = structure:GetNodeIndex(basenode)

disp_load = tmath.Matrix(1, 2)
disp_load_local = tmath.Matrix(1, 2)
disp_load_print = tmath.Matrix(1, 2)

```

```
144 -- Prepare graphics
145 g = graph3d.Graph3D("Push", 220, 220, 255)
146 g:SetCulling(false)
147 plot = structure:Draw(1000)
148 g:Triangles(plot)
149 g:Autoscale()
150 g:Render()
151
152 disp_load[{0,0}] = 0
153 disp_load[{0,1}] = 0
154 disp_load_local[{0,0}] = 0
155 disp_load_local[{0,1}] = 0
156
157 --disp-force Graph
158 v2=graph.Graph("Response", "Bright")
159 v2:AxisLabels("Displacement [m]", "Base Shear [N]")
160
161 for i=1,N-1 do
162
163 -- Applying loads
164 Loads=structure:GetAllDisplacements()
165 Loads[{tipnode1, load_dir}] = F/5
166 Loads[{tipnode2, load_dir}] = F/5
167 Loads[{tipnode3, load_dir}] = F/5
168 Loads[{tipnode4, load_dir}] = F/5
169 Loads[{tipnode5, load_dir}] = F/5
170
171 --print("Loads", Loads)
172 P = structure:ToDofDisplacements(Loads)
173
174 if (i>1) then
175 for k=1, ki do
176
177 -- Assemble stiffness matrix
178 K = structure:SparseStiffness()
179
180 -- Solve for displacements and assign to structure
181
182 U=K:Solve(P)
183 --    print("K", K)
184 --    print("U", U)
185
186 U1=structure:ToAllDisplacements(U)
187 --       print("U1", U1)
188 structure:SetAllDisplacements(U1)
189 disp = structure:GetAllDisplacements()
190 disp_load_local[{0,0}] = disp[{topnode,0}]
191
192 -- compute internal (elastic and plastic) forces
```

```
193 RF = structure:GlobalResForce()
194
195 -- Convert to matrix representation and add up all x-
       components
196 F1 = structure:ToAllDisplacements(RF)
197 sum = tmath.Sum(F1:GetCols(0))
198 disp_load_local[{0,1}] = sum
199 disp_load = disp_load:AppendRows(disp_load_local)
200
201 g:Clear()
202 plot = structure:Draw(1000) -- Deformation with scale 1000
203 g:Triangles(plot)
204 g:Render()
205
206 v2:Plot(disp_load:GetCols(0), disp_load:GetCols(1), 1)
207 v2:Render()
208 -- write vector disp_load in txt file
209 tmath.Output(disp_load ,"disp_load.txt")
210
211 end --end for k
212 end --end if
213
214 print("Failure Tension", structure:GetFailureTension())
215 -- print("Failure Compression", structure:
       GetFailureCompression())
216
217 F = F+DF
218 end --for
219
220 for j=1, PrintN do
221 disp_load_print = disp_load_print:AppendRows(disp_load:
       GetRows(j))
222 end
223
224 v3=graph.Graph("Response", "Bright")
225 v3:AxisLabels("Displacement [m]", "Base Shear [N]")
226 v3:Plot(disp_load_print:GetCols(0), disp_load_print:GetCols
       (1), 1)
227 v3:Render()
228
229 print("GlobalResForce", structure:GlobalResForce())
230
231 --print("Failure Tension", structure:GetFailureTension())
232 --print("Failure Compression", structure:
       GetFailureCompression())
233
234 disp = structure:GetAllDisplacements()
235 rel_disp_x = disp[{23,0}] - disp[{3,0}]
236 rel_disp_y = disp[{23,1}] - disp[{3,1}]
```

```
print("rel_disp_x", rel_disp_x)
print("rel_disp_y", rel_disp_y)
```

D.4 Beispiel 4.3 Gebäudemodell unter kraftgesteuerter Pushover-Analyse

modell_test.tng

```
--[[
Modelling of the building
--]]

-- Create and new FE structure
structure=fem.Structure("building")

--Define node coordinates of structure

nodes = tmath.Matrix({

{001, 0.0, 0.0, 0.0},
{002, 4.0, 0.0, 0.0},
{003, 8.0, 0.0, 0.0},
{004, 0.0, 6.0, 0.0},
{005, 4.0, 6.0, 0.0},
{006, 8.0, 6.0, 0.0},

{101, 0.0, 0.0, 3.0},
{102, 4.0, 0.0, 3.0},
{103, 8.0, 0.0, 3.0},
{104, 0.0, 6.0, 3.0},
{105, 4.0, 6.0, 3.0},
{106, 8.0, 6.0, 3.0},

{201, 0.0, 0.0, 6.0},
{202, 4.0, 0.0, 6.0},
{203, 8.0, 0.0, 6.0},
{204, 0.0, 6.0, 6.0},
{205, 4.0, 6.0, 6.0},
{206, 8.0, 6.0, 6.0},

{301, 0.0, 0.0, 9.0},
{302, 4.0, 0.0, 9.0},
{303, 8.0, 0.0, 9.0},
{304, 0.0, 6.0, 9.0},
{305, 4.0, 6.0, 9.0},
{306, 8.0, 6.0, 9.0}
```

```
39
40 })
41
42 structure:AddNodes(nodes)
43
44 -- Define support conditions
45
46 -- structure:GetNode(001):SetAvailDof(tmath.Matrix({{0, 0, 0,
       0, 0, 0}}))
47 -- structure:GetNode(002):SetAvailDof(tmath.Matrix({{0, 0, 0,
       0, 0, 0}}))
48 -- structure:GetNode(003):SetAvailDof(tmath.Matrix({{0, 0, 0,
       0, 0, 0}}))
49 -- structure:GetNode(004):SetAvailDof(tmath.Matrix({{0, 0, 0,
       0, 0, 0}}))
50 -- structure:GetNode(005):SetAvailDof(tmath.Matrix({{0, 0, 0,
       0, 0, 0}}))
51 -- structure:GetNode(006):SetAvailDof(tmath.Matrix({{0, 0, 0,
       0, 0, 0}}))
52 structure:GetNode(001):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
       1, 1}}))
53 structure:GetNode(002):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
       1, 1}}))
54 structure:GetNode(003):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
       1, 1}}))
55 structure:GetNode(004):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
       1, 1}}))
56 structure:GetNode(005):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
       1, 1}}))
57 structure:GetNode(006):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
       1, 1}}))
58
59
60 -- Define material --- e+6 = N/mm2
61 mas = structure:AddMaterial(001, 'ELASTIC_FAILURE')
62 mas:SetData(tmath.Matrix({{1.6e+9, 0.2 , 1700, -5e+6, 2e+5
     }}))
63
64 --    mas = structure:AddMaterial(001, 'LINEAR_ELASTIC')
65 --    mas:SetData(tmath.Matrix({{3e+9, 0.2 , 1700}}))
66
67 --    timber = structure:AddMaterial(002, 'LINEAR_ELASTIC')
     --slab linear elastic
68 --    timber:SetData(tmath.Matrix({{3.7e+8, 0.3 , 500 }}))
     --3.7e+7 (in Querrichtung), 1.1e+9 (in Faserrichtung)
69
70 timber = structure:AddMaterial(002, 'ELASTIC_FAILURE') --slab
       elastic failure
```

```
71 timber:SetData(tmath.Matrix({{3.7e+8, 0.3 , 500, -3e+6, 5.5e
      +6 }})) --3.7e+8 (in Querrichtung), 1.1e+10 (in
      Faserrichtung)
72
73 concrete = structure:AddMaterial(003, 'LINEAR_ELASTIC')
74 concrete:SetData(tmath.Matrix({{3e+10, 0.2 , 2500 }}))
75
76 -- Define cross sections
77 sec = structure:AddSection(001, 'SHELL') -- wall
78 sec:SetData(tmath.Matrix({{0.3}}))
79 sec:SetColor(tmath.Matrix({{128, 50, 50, 100}}))
80
81 sec = structure:AddSection(010, 'SHELL') -- shear wall
82 sec:SetData(tmath.Matrix({{0.15}}))
83 sec:SetColor(tmath.Matrix({{128, 50, 50, 100}}))
84
85 sec = structure:AddSection(100, 'SHELL') -- wooden slab
86 sec:SetData(tmath.Matrix({{0.16}}))
87 sec:SetColor(tmath.Matrix({{128, 128, 128, 100}}))
88
89 sec = structure:AddSection(101, 'SHELL') -- concrete-wooden
      slab
90 sec:SetData(tmath.Matrix({{0.22}}))
91 sec:SetColor(tmath.Matrix({{128, 128, 128, 100}}))
92
93 -- Add elements to structure
94
95 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0001, 001, 002, 102, 101}}))
96 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0002, 002, 003, 103, 102}}))
97 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0003, 004, 005, 105, 104}}))
98 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0004, 005, 006, 106, 105}}))
99 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0005, 001, 004, 104, 101}}))
100 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0006, 002, 005, 105, 102}}))
101 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0007, 003, 006, 106, 103}}))
102
103 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      0101, 101, 102, 105, 104}}))
104 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      0102, 102, 103, 106, 105}}))
105
106 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1001, 101, 102, 202, 201}}))
```

```
107 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1002, 102, 103, 203, 202}}))
108 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1003, 104, 105, 205, 204}}))
109 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1004, 105, 106, 206, 205}}))
110 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1005, 101, 104, 204, 201}}))
111 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1006, 102, 105, 205, 202}}))
112 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      1007, 103, 106, 206, 203}}))
113
114 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      1101, 201, 202, 205, 204}}))
115 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      1102, 202, 203, 206, 205}}))
116
117 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2001, 201, 202, 302, 301}}))
118 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2002, 202, 203, 303, 302}}))
119 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2003, 204, 205, 305, 304}}))
120 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2004, 205, 206, 306, 305}}))
121 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2005, 201, 204, 304, 301}}))
122 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2006, 202, 205, 305, 302}}))
123 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      2007, 203, 206, 306, 303}}))
124
125 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      2101, 301, 302, 305, 304}}))
126 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      2102, 302, 303, 306, 305}}))
127
128 -- Assemble all dofs
129 ndof = structure:GlobalDof()
130 --print(ndof)
131
132 topnode1 = structure:GetNodeIndex(301)
133 topnode2 = structure:GetNodeIndex(302)
134 topnode3 = structure:GetNodeIndex(303)
135 topnode4 = structure:GetNodeIndex(304)
136 topnode5 = structure:GetNodeIndex(305)
137 topnode6 = structure:GetNodeIndex(306)
138
```

```
level2node1 = structure:GetNodeIndex(201)
level2node2 = structure:GetNodeIndex(202)
level2node3 = structure:GetNodeIndex(203)
level2node4 = structure:GetNodeIndex(204)
level2node5 = structure:GetNodeIndex(205)
level2node6 = structure:GetNodeIndex(206)

level1node1 = structure:GetNodeIndex(101)
level1node2 = structure:GetNodeIndex(102)
level1node3 = structure:GetNodeIndex(103)
level1node4 = structure:GetNodeIndex(104)
level1node5 = structure:GetNodeIndex(105)
level1node6 = structure:GetNodeIndex(106)

basepoint = structure:GetNodeIndex(001)

--[[
-- plotting structure
g = graph3d.Graph3D("structure")
g:Clear()
tris = structure:Draw()
g:Triangles(tris)
g:Autoscale()
g:Zoom(0.7)
]]--
```

push_modell.tng

```

--push over analysis

--load structure modell
dofile("modell_test.tng")

F = 1e+3
DF = 1e+3

-- Number of load steps
N = 30
ki = 12 --iterations per loadstep

--Graph number of data rows
PrintN = 335

dir = 0
--0 = x
--1 = y
```

```
--2 = z

-- Assemble stiffness matrix
K = structure:SparseStiffness()


disp_load = tmath.Matrix(1, 2)
disp_load_local = tmath.Matrix(1, 2)
disp_load_print = tmath.Matrix(1, 2)

-- Prepare graphics
g = graph3d.Graph3D("Push", 220, 220, 255)
g:SetCulling(false)
g:Rotate(-90, 1, 0, 0)
g:Rotate(0, 0, 1, 0)
plot = structure:Draw(1)
g:Triangles(plot)
g:Autoscale()
g:Render()

disp_load[{0,0}] = 0
disp_load[{0,1}] = 0
disp_load_local[{0,0}] = 0
disp_load_local[{0,1}] = 0

-- setup Force-Displacement Graph
v2=graph.Graph("Response", "Bright")
v2:AxisLabels("Displacement [m]", "Base Shear [N]")

for i=1,N-1 do

-- Applying loads
Loads=structure:GetAllDisplacements()

Loads[{topnode1, dir}] = F
Loads[{topnode2, dir}] = F
Loads[{topnode3, dir}] = F
Loads[{topnode4, dir}] = F
Loads[{topnode5, dir}] = F
Loads[{topnode6, dir}] = F

Loads[{level2node1, dir}] = F*2/3
Loads[{level2node2, dir}] = F*2/3
Loads[{level2node3, dir}] = F*2/3
Loads[{level2node4, dir}] = F*2/3
Loads[{level2node5, dir}] = F*2/3
Loads[{level2node6, dir}] = F*2/3

Loads[{level1node1, dir}] = F*1/3
```

```
Loads[{level1node2, dir}] = F*1/3
Loads[{level1node3, dir}] = F*1/3
Loads[{level1node4, dir}] = F*1/3
Loads[{level1node5, dir}] = F*1/3
Loads[{level1node6, dir}] = F*1/3

P = structure:ToDofDisplacements(Loads)

if (i>1) then
for k=1, ki do

K = structure:SparseStiffness()
--print("K Stiff", K)

-- Solve for displacements and assign to structure

U=K:Solve(P)
--          print("K", K)
--          print("U", U)

U1=structure:ToAllDisplacements(U)
--        print("U1", U1)
structure:SetAllDisplacements(U1)
--     print("displacements set")
disp = structure:GetAllDisplacements()
disp_load_local[{0,0}] = disp[{topnode1,dir}]

-- compute internal (elastic and plastic) forces
RF = structure:GlobalResForce()

-- Convert to matrix representation and add up all x-
   components
F1 = structure:ToAllDisplacements(RF)
sum = tmath.Sum(F1:GetCols(dir))
disp_load_local[{0,1}] = sum
disp_load = disp_load:AppendRows(disp_load_local)
--        print("disp_load_local", disp_load_local)
--     print("disp_load", disp_load)

g:Clear()
plot = structure:Draw(100) -- Deformation with scale 100
g:Triangles(plot)
g:Render()


-- str = structure:ElementStress(0)
-- stress = structure:ElementResults(str, 0)
-- print("Stress im Loadstep", str)

```

```
117 v2:Plot(disp_load:GetCols(0), disp_load:GetCols(1), 1)
118 v2:Render()
119
120 -- write vector disp_load in txt file
121 tmath.Output(disp_load ,"disp_load.txt")
122
123 -- print("Failure Tension", structure:GetFailureTension())
124 -- print("Failure Compression", structure:
       GetFailureCompression())
125
126 end --end while k
127 end --end if
128
129 F = F+DF
130
131 end --for i
132
133 X = structure:GetCoordinates()
134 z_topnode1 = X[{topnode1, 2}]
135 print("z_topnode", z_topnode1)
136 u_max = X[{topnode1, 2}]*0.004
137 print("u_max", u_max)
138
139 for j=1, PrintN do
140 disp_load_print = disp_load_print:AppendRows(disp_load:
       GetRows(j))
141 end
142
143 v3=graph.Graph("Response", "Bright")
144 v3:AxisLabels("Displacement [m]", "Base Shear [N]")
145 v3:Plot(disp_load_print:GetCols(0), disp_load_print:GetCols
       (1), 1)
146 v3:Render()
147
148 print("GlobalResForce", structure:GlobalResForce())
149
150 print("Failure Compression", structure:GetFailureCompression
       ())
151 print("Failure Tension", structure:GetFailureTension())
152
153 disp = structure:GetAllDisplacements()
154 rel_disp_x = disp[{topnode1,0}] - disp[{basepoint,0}]
155 rel_disp_y = disp[{topnode1,1}] - disp[{basepoint,1}]
156 print("rel_disp_x", rel_disp_x)
157 print("rel_disp_y", rel_disp_y)
158
159 print("disp topnode 1", disp[{topnode1,dir}])
160 print("disp topnode 2", disp[{topnode2,dir}])
161 print("disp topnode 3", disp[{topnode3,dir}])
```

```
162 print("disp topnode 4", disp[{topnode4,dir}])
163 print("disp topnode 5", disp[{topnode5,dir}])
164 print("disp topnode 6", disp[{topnode6,dir}])
```

Anhang E

Anwendungsbeispiele Kapitel 6 und Kapitel 7

E.1 Regelmäßiges Gebäudemodell

modell_sym.tng

```
--[[
Modelling of the symmetric building
--]]

-- Create and new FE structure
structure=fem.Structure("building")

--Dimensions of Building
a = 10
b = 6
h = 3.9

--Define node coordinates of corner nodes

nodes = tmath.Matrix({

{001, 0.0, 0.0, 0.0},
{002, a/5, 0.0, 0.0},
{003, 2*a/5, 0.0, 0.0},
{004, 3*a/5, 0.0, 0.0},
{005, 4*a/5, 0.0, 0.0},
{006, a, 0.0, 0.0},
{007, 0.0, b/2, 0.0},
{008, a/5, b/2, 0.0},
{009, 2*a/5, b/2, 0.0},
{010, 3*a/5, b/2, 0.0},
{011, 4*a/5, b/2, 0.0},
{012, a, b/2, 0.0},
{013, 0.0, b, 0.0},
{014, a/5, b, 0.0},
{015, 2*a/5, b, 0.0},
{016, 3*a/5, b, 0.0},
{017, 4*a/5, b, 0.0},
{018, a, b, 0.0},
```

```
35
36  {101, 0.0, 0.0, h/3},
37  {102, a/5, 0.0, h/3},
38  {103, 2*a/5, 0.0, h/3},
39  {104, 3*a/5, 0.0, h/3},
40  {105, 4*a/5, 0.0, h/3},
41  {106, a, 0.0, h/3},
42  {107, 0.0, b/2, h/3},
43  {108, a/5, b/2, h/3},
44  {109, 2*a/5, b/2, h/3},
45  {110, 3*a/5, b/2, h/3},
46  {111, 4*a/5, b/2, h/3},
47  {112, a, b/2, h/3},
48  {113, 0.0, b, h/3},
49  {114, a/5, b, h/3},
50  {115, 2*a/5, b, h/3},
51  {116, 3*a/5, b, h/3},
52  {117, 4*a/5, b, h/3},
53  {118, a, b, h/3},
54
55  {201, 0.0, 0.0, 2/3*h},
56  {202, a/5, 0.0, 2/3*h},
57  {203, 2*a/5, 0.0, 2/3*h},
58  {204, 3*a/5, 0.0, 2/3*h},
59  {205, 4*a/5, 0.0, 2/3*h},
60  {206, a, 0.0, 2/3*h},
61  {207, 0.0, b/2, 2/3*h},
62  {208, a/5, b/2, 2/3*h},
63  {209, 2*a/5, b/2, 2/3*h},
64  {210, 3*a/5, b/2, 2/3*h},
65  {211, 4*a/5, b/2, 2/3*h},
66  {212, a, b/2, 2/3*h},
67  {213, 0.0, b, 2/3*h},
68  {214, a/5, b, 2/3*h},
69  {215, 2*a/5, b, 2/3*h},
70  {216, 3*a/5, b, 2/3*h},
71  {217, 4*a/5, b, 2/3*h},
72  {218, a, b, 2/3*h},
73
74  {301, 0.0, 0.0, h},
75  {302, a/5, 0.0, h},
76  {303, 2*a/5, 0.0, h},
77  {304, 3*a/5, 0.0, h},
78  {305, 4*a/5, 0.0, h},
79  {306, a, 0.0, h},
80  {307, 0.0, b/2, h},
81  {308, a/5, b/2, h},
82  {309, 2*a/5, b/2, h},
83  {310, 3*a/5, b/2, h},
```

```
84  {311, 4*a/5, b/2, h},
85  {312, a, b/2, h},
86  {313, 0.0, b, h},
87  {314, a/5, b, h},
88  {315, 2*a/5, b, h},
89  {316, 3*a/5, b, h},
90  {317, 4*a/5, b, h},
91  {318, a, b, h},
92
93  {401, 0.0, 0.0, 4/3*h},
94  {402, a/5, 0.0, 4/3*h},
95  {403, 2*a/5, 0.0, 4/3*h},
96  {404, 3*a/5, 0.0, 4/3*h},
97  {405, 4*a/5, 0.0, 4/3*h},
98  {406, a, 0.0, 4/3*h},
99  {407, 0.0, b/2, 4/3*h},
100 {408, a/5, b/2, 4/3*h},
101 {409, 2*a/5, b/2, 4/3*h},
102 {410, 3*a/5, b/2, 4/3*h},
103 {411, 4*a/5, b/2, 4/3*h},
104 {412, a, b/2, 4/3*h},
105 {413, 0.0, b, 4/3*h},
106 {414, a/5, b, 4/3*h},
107 {415, 2*a/5, b, 4/3*h},
108 {416, 3*a/5, b, 4/3*h},
109 {417, 4*a/5, b, 4/3*h},
110 {418, a, b, 4/3*h},
111
112 {501, 0.0, 0.0, 5/3*h},
113 {502, a/5, 0.0, 5/3*h},
114 {503, 2*a/5, 0.0, 5/3*h},
115 {504, 3*a/5, 0.0, 5/3*h},
116 {505, 4*a/5, 0.0, 5/3*h},
117 {506, a, 0.0, 5/3*h},
118 {507, 0.0, b/2, 5/3*h},
119 {508, a/5, b/2, 5/3*h},
120 {509, 2*a/5, b/2, 5/3*h},
121 {510, 3*a/5, b/2, 5/3*h},
122 {511, 4*a/5, b/2, 5/3*h},
123 {512, a, b/2, 5/3*h},
124 {513, 0.0, b, 5/3*h},
125 {514, a/5, b, 5/3*h},
126 {515, 2*a/5, b, 5/3*h},
127 {516, 3*a/5, b, 5/3*h},
128 {517, 4*a/5, b, 5/3*h},
129 {518, a, b, 5/3*h},
130
131 {601, 0.0, 0.0, 2*h},
132 {602, a/5, 0.0, 2*h},
```

```
133 {603, 2*a/5, 0.0, 2*h},
134 {604, 3*a/5, 0.0, 2*h},
135 {605, 4*a/5, 0.0, 2*h},
136 {606, a, 0.0, 2*h},
137 {607, 0.0, b/2, 2*h},
138 {608, a/5, b/2, 2*h},
139 {609, 2*a/5, b/2, 2*h},
140 {610, 3*a/5, b/2, 2*h},
141 {611, 4*a/5, b/2, 2*h},
142 {612, a, b/2, 2*h},
143 {613, 0.0, b, 2*h},
144 {614, a/5, b, 2*h},
145 {615, 2*a/5, b, 2*h},
146 {616, 3*a/5, b, 2*h},
147 {617, 4*a/5, b, 2*h},
148 {618, a, b, 2*h},
149
150 {701, 0.0, 0.0, 7/3*h},
151 {702, a/5, 0.0, 7/3*h},
152 {703, 2*a/5, 0.0, 7/3*h},
153 {704, 3*a/5, 0.0, 7/3*h},
154 {705, 4*a/5, 0.0, 7/3*h},
155 {706, a, 0.0, 7/3*h},
156 {707, 0.0, b/2, 7/3*h},
157 {708, a/5, b/2, 7/3*h},
158 {709, 2*a/5, b/2, 7/3*h},
159 {710, 3*a/5, b/2, 7/3*h},
160 {711, 4*a/5, b/2, 7/3*h},
161 {712, a, b/2, 7/3*h},
162 {713, 0.0, b, 7/3*h},
163 {714, a/5, b, 7/3*h},
164 {715, 2*a/5, b, 7/3*h},
165 {716, 3*a/5, b, 7/3*h},
166 {717, 4*a/5, b, 7/3*h},
167 {718, a, b, 7/3*h},
168
169 {801, 0.0, 0.0, 8/3*h},
170 {802, a/5, 0.0, 8/3*h},
171 {803, 2*a/5, 0.0, 8/3*h},
172 {804, 3*a/5, 0.0, 8/3*h},
173 {805, 4*a/5, 0.0, 8/3*h},
174 {806, a, 0.0, 8/3*h},
175 {807, 0.0, b/2, 8/3*h},
176 {808, a/5, b/2, 8/3*h},
177 {809, 2*a/5, b/2, 8/3*h},
178 {810, 3*a/5, b/2, 8/3*h},
179 {811, 4*a/5, b/2, 8/3*h},
180 {812, a, b/2, 8/3*h},
181 {813, 0.0, b, 8/3*h},
```

```
182 {814, a/5, b, 8/3*h},
183 {815, 2*a/5, b, 8/3*h},
184 {816, 3*a/5, b, 8/3*h},
185 {817, 4*a/5, b, 8/3*h},
186 {818, a, b, 8/3*h},
187
188 {901, 0.0, 0.0, 3*h},
189 {902, a/5, 0.0, 3*h},
190 {903, 2*a/5, 0.0, 3*h},
191 {904, 3*a/5, 0.0, 3*h},
192 {905, 4*a/5, 0.0, 3*h},
193 {906, a, 0.0, 3*h},
194 {907, 0.0, b/2, 3*h},
195 {908, a/5, b/2, 3*h},
196 {909, 2*a/5, b/2, 3*h},
197 {910, 3*a/5, b/2, 3*h},
198 {911, 4*a/5, b/2, 3*h},
199 {912, a, b/2, 3*h},
200 {913, 0.0, b, 3*h},
201 {914, a/5, b, 3*h},
202 {915, 2*a/5, b, 3*h},
203 {916, 3*a/5, b, 3*h},
204 {917, 4*a/5, b, 3*h},
205 {918, a, b, 3*h},
206 })
207
208 structure:AddNodes(nodes)
209
210
211 -- Define support conditions
212
213 structure:GetNode(001):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
214 structure:GetNode(002):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
215 structure:GetNode(003):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
216 structure:GetNode(004):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
217 structure:GetNode(005):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
218 structure:GetNode(006):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
219 structure:GetNode(007):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
220 structure:GetNode(008):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
221 structure:GetNode(009):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
```

```
222 structure:GetNode(010):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
223 structure:GetNode(011):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
224 structure:GetNode(012):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
225 structure:GetNode(013):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
226 structure:GetNode(014):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
227 structure:GetNode(015):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
228 structure:GetNode(016):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
229 structure:GetNode(017):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
230 structure:GetNode(018):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
231
232 -- Define material --- e+6 = N/mm2
233 mas = structure:AddMaterial(001, 'ELASTIC_FAILURE')
234 mas:SetData(tmath.Matrix({{1.6e+9, 0.2 , 1700, -5e+6, 4e+5
       }}))
235
236 timber = structure:AddMaterial(002, 'ELASTIC_FAILURE') --slab
        elastic failure
237 timber:SetData(tmath.Matrix({{3.7e+8, 0.3 , 500, -3e+6, 5.5e
       +6 }})) --E-Modul 3.7e+8 (in Querrichtung), 1.1e+10 (in
       Faserrichtung)
238
239 -- Define cross sections
240 sec = structure:AddSection(001, 'SHELL') -- wall
241 sec:SetData(tmath.Matrix({{0.45}}))
242 sec:SetColor(tmath.Matrix({{128, 50, 50, 100}}))
243
244 sec = structure:AddSection(010, 'SHELL') -- shear wall
245 sec:SetData(tmath.Matrix({{0.30}}))
246 sec:SetColor(tmath.Matrix({{128, 50, 50, 100}}))
247
248 sec = structure:AddSection(100, 'SHELL') -- wooden slab
249 sec:SetData(tmath.Matrix({{0.16}}))
250 sec:SetColor(tmath.Matrix({{128, 128, 128, 100}}))
251
252 sec = structure:AddSection(101, 'SHELL') -- concrete slab
253 sec:SetData(tmath.Matrix({{0.2}}))
254 sec:SetColor(tmath.Matrix({{128, 128, 128, 100}}))
255
256 -- Add elements to structure
257 --storey 0
```

```
258 --walls a
259 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0001, 001, 002, 102, 101}}))
260 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0002, 003, 004, 104, 103}}))
261 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0003, 005, 006, 106, 105}}))
262 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0004, 007, 008, 108, 107}}))
263 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0005, 008, 009, 109, 108}}))
264 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 0006, 009, 010, 110, 109}}))
265 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0007, 010, 011, 111, 110}}))
266 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0008, 011, 012, 112, 111}}))
267 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0009, 013, 014, 114, 113}}))
268 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0010, 015, 016, 116, 115}}))
269 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       0011, 017, 018, 118, 117}}))
270 --walls b
271 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0012, 001, 007, 107, 101}}))
272 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0013, 007, 013, 113, 107}}))
273 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0014, 003, 009, 109, 103}}))
274 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0015, 009, 015, 115, 109}}))
275 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0016, 004, 010, 110, 104}}))
276 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0017, 010, 016, 116, 110}}))
277 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0018, 006, 012, 112, 106}}))
278 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0019, 012, 018, 118, 112}}))
279
280 --walls a
281 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1001, 101, 102, 202, 201}}))
282 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1002, 103, 104, 204, 203}}))
283 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1003, 105, 106, 206, 205}}))
```

```
284 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1004, 107, 108, 208, 207}}))
285 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1005, 108, 109, 209, 208}}))
286 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 1006, 109, 110, 210, 209}}))
287 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1007, 110, 111, 211, 210}}))
288 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1008, 111, 112, 212, 211}}))
289 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1009, 113, 114, 214, 213}}))
290 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1010, 115, 116, 216, 215}}))
291 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1011, 117, 118, 218, 217}}))
292 --walls b
293 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1012, 101, 107, 207, 201}}))
294 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1013, 107, 113, 213, 207}}))
295 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1014, 103, 109, 209, 203}}))
296 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1015, 109, 115, 215, 209}}))
297 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1016, 104, 110, 210, 204}}))
298 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1017, 110, 116, 216, 210}}))
299 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1018, 106, 112, 212, 206}}))
300 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1019, 112, 118, 218, 212}}))
301
302 --walls a
303 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2001, 201, 202, 302, 301}}))
304 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2002, 203, 204, 304, 303}}))
305 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2003, 205, 206, 306, 305}}))
306 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2004, 207, 208, 308, 307}}))
307 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2005, 208, 209, 309, 308}}))
308 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 2006, 209, 210, 310, 309}}))
309 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2007, 210, 211, 311, 310}}))
```

```
310 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2008, 211, 212, 312, 311}}))
311 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2009, 213, 214, 314, 313}}))
312 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2010, 215, 216, 316, 315}}))
313 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2011, 217, 218, 318, 317}}))
314 --walls b
315 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2012, 201, 207, 307, 301}}))
316 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2013, 207, 213, 313, 307}}))
317 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2014, 203, 209, 309, 303}}))
318 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2015, 209, 215, 315, 309}}))
319 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2016, 204, 210, 310, 304}}))
320 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2017, 210, 216, 316, 310}}))
321 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2018, 206, 212, 312, 306}}))
322 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2019, 212, 218, 318, 312}}))
323
324 --slab
325 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2101, 301, 302, 308, 307}}))
326 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2102, 302, 303, 309, 308}}))
327 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2103, 303, 304, 310, 309}}))
328 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2104, 304, 305, 311, 310}}))
329 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2105, 305, 306, 312, 311}}))
330 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2106, 307, 308, 314, 313}}))
331 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2107, 308, 309, 315, 314}}))
332 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2108, 309, 310, 316, 315}}))
333 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2109, 310, 311, 317, 316}}))
334 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2110, 311, 312, 318, 317}}))
335
336 --storey 1
```

```
337
338 --walls a
339 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3001, 301, 302, 402, 401}}))
340 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3002, 303, 304, 404, 403}}))
341 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3003, 305, 306, 406, 405}}))
342 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3004, 307, 308, 408, 407}}))
343 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3005, 308, 309, 409, 408}}))
344 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 3006, 309, 310, 410, 409}}))
345 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3007, 310, 311, 411, 410}}))
346 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3008, 311, 312, 412, 411}}))
347 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3009, 313, 314, 414, 413}}))
348 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3010, 315, 316, 416, 415}}))
349 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       3011, 317, 318, 418, 417}}))
350 --walls b
351 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3012, 301, 307, 407, 401}}))
352 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3013, 307, 313, 413, 407}}))
353 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3014, 303, 309, 409, 403}}))
354 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3015, 309, 315, 415, 409}}))
355 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3016, 304, 310, 410, 404}}))
356 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3017, 310, 316, 416, 410}}))
357 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3018, 306, 312, 412, 406}}))
358 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3019, 312, 318, 418, 412}}))
359
360 --walls a
361 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4001, 401, 402, 502, 501}}))
362 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4002, 403, 404, 504, 503}}))
363 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4003, 405, 406, 506, 505}}))
```

```
364 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4004, 407, 408, 508, 507}}))
365 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4005, 408, 409, 509, 508}}))
366 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 4006, 409, 410, 510, 509}}))
367 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4007, 410, 411, 511, 510}}))
368 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4008, 411, 412, 512, 511}}))
369 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4009, 413, 414, 514, 513}}))
370 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4010, 415, 416, 516, 515}}))
371 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4011, 417, 418, 518, 517}}))
372 --walls b
373 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4012, 401, 407, 507, 501}}))
374 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4013, 407, 413, 513, 507}}))
375 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4014, 403, 409, 509, 503}}))
376 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4015, 409, 415, 515, 509}}))
377 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4016, 404, 410, 510, 504}}))
378 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4017, 410, 416, 516, 510}}))
379 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4018, 406, 412, 512, 506}}))
380 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4019, 412, 418, 518, 512}}))
381
382 --walls a
383 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5001, 501, 502, 602, 601}}))
384 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5002, 503, 504, 604, 603}}))
385 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5003, 505, 506, 606, 605}}))
386 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5004, 507, 508, 608, 607}}))
387 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5005, 508, 509, 609, 608}}))
388 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 5006, 509, 510, 610, 609}}))
389 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5007, 510, 511, 611, 610}}))
```

```
390 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5008, 511, 512, 612, 611}}))
391 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5009, 513, 514, 614, 613}}))
392 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5010, 515, 516, 616, 615}}))
393 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       5011, 517, 518, 618, 617}}))
394 --walls b
395 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5012, 501, 507, 607, 601}}))
396 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5013, 507, 513, 613, 607}}))
397 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5014, 503, 509, 609, 603}}))
398 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5015, 509, 515, 615, 609}}))
399 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5016, 504, 510, 610, 604}}))
400 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5017, 510, 516, 616, 610}}))
401 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5018, 506, 512, 612, 606}}))
402 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5019, 512, 518, 618, 612}}))
403
404 --slab
405 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5101, 601, 602, 608, 607}}))
406 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5102, 602, 603, 609, 608}}))
407 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5103, 603, 604, 610, 609}}))
408 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5104, 604, 605, 611, 610}}))
409 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5105, 605, 606, 612, 611}}))
410 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5106, 607, 608, 614, 613}}))
411 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5107, 608, 609, 615, 614}}))
412 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5108, 609, 610, 616, 615}}))
413 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5109, 610, 611, 617, 616}}))
414 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5110, 611, 612, 618, 617}}))
415
416 --storey 2
```

```
417
418 --walls a
419 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6001, 601, 602, 702, 701}}))
420 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6002, 603, 604, 704, 703}}))
421 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6003, 605, 606, 706, 705}}))
422 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6004, 607, 608, 708, 707}}))
423 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6005, 608, 609, 709, 708}}))
424 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
      ({{ 6006, 609, 610, 710, 709}}))
425 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6007, 610, 611, 711, 710}}))
426 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6008, 611, 612, 712, 711}}))
427 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6009, 613, 614, 714, 713}}))
428 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6010, 615, 616, 716, 715}}))
429 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      6011, 617, 618, 718, 717}}))
430 --walls b
431 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6012, 601, 607, 707, 701}}))
432 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6013, 607, 613, 713, 707}}))
433 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6014, 603, 609, 709, 703}}))
434 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6015, 609, 615, 715, 709}}))
435 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6016, 604, 610, 710, 704}}))
436 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6017, 610, 616, 716, 710}}))
437 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6018, 606, 612, 712, 706}}))
438 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      6019, 612, 618, 718, 712}}))
439
440 --walls a
441 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      7001, 701, 702, 802, 801}}))
442 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      7002, 703, 704, 804, 803}}))
443 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      7003, 705, 706, 806, 805}}))
```

```
444 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7004, 707, 708, 808, 807}}))
445 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7005, 708, 709, 809, 808}}))
446 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 7006, 709, 710, 810, 809}}))
447 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7007, 710, 711, 811, 810}}))
448 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7008, 711, 712, 812, 811}}))
449 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7009, 713, 714, 814, 813}}))
450 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7010, 715, 716, 816, 815}}))
451 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7011, 717, 718, 818, 817}}))
452 --walls b
453 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7012, 701, 707, 807, 801}}))
454 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7013, 707, 713, 813, 807}}))
455 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7014, 703, 709, 809, 803}}))
456 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7015, 709, 715, 815, 809}}))
457 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7016, 704, 710, 810, 804}}))
458 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7017, 710, 716, 816, 810}}))
459 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7018, 706, 712, 812, 806}}))
460 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7019, 712, 718, 818, 812}}))
461
462 --walls a
463 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8001, 801, 802, 902, 901}}))
464 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8002, 803, 804, 904, 903}}))
465 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8003, 805, 806, 906, 905}}))
466 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8004, 807, 808, 908, 907}}))
467 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8005, 808, 809, 909, 908}}))
468 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 8006, 809, 810, 910, 909}}))
469 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8007, 810, 811, 911, 910}}))
```

```
470 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8008, 811, 812, 912, 911}}))
471 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8009, 813, 814, 914, 913}}))
472 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8010, 815, 816, 916, 915}}))
473 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8011, 817, 818, 918, 917}}))
474 --walls b
475 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8012, 801, 807, 907, 901}}))
476 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8013, 807, 813, 913, 907}}))
477 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8014, 803, 809, 909, 903}}))
478 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8015, 809, 815, 915, 909}}))
479 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8016, 804, 810, 910, 904}}))
480 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8017, 810, 816, 916, 910}}))
481 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8018, 806, 812, 912, 906}}))
482 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8019, 812, 818, 918, 912}}))
483
484 --slab
485 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8101, 901, 902, 908, 907}}))
486 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8102, 902, 903, 909, 908}}))
487 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8103, 903, 904, 910, 909}}))
488 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8104, 904, 905, 911, 910}}))
489 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8105, 905, 906, 912, 911}}))
490 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8106, 907, 908, 914, 913}}))
491 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8107, 908, 909, 915, 914}}))
492 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8108, 909, 910, 916, 915}}))
493 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8109, 910, 911, 917, 916}}))
494 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8110, 911, 912, 918, 917}}))
495
496
```

```
497
498 -- Assemble all dofs
499 ndof = structure:GlobalDof()
500 --print(ndof)
501
502 basepoint1 = structure:GetNodeIndex(001)
503 basepoint2 = structure:GetNodeIndex(002)
504 basepoint3 = structure:GetNodeIndex(003)
505 basepoint4 = structure:GetNodeIndex(004)
506 basepoint5 = structure:GetNodeIndex(005)
507 basepoint6 = structure:GetNodeIndex(006)
508 basepoint7 = structure:GetNodeIndex(007)
509 basepoint8 = structure:GetNodeIndex(008)
510 basepoint9 = structure:GetNodeIndex(009)
511 basepoint10 = structure:GetNodeIndex(010)
512 basepoint11 = structure:GetNodeIndex(011)
513 basepoint12 = structure:GetNodeIndex(012)
514 basepoint13 = structure:GetNodeIndex(013)
515 basepoint14 = structure:GetNodeIndex(014)
516 basepoint15 = structure:GetNodeIndex(015)
517 basepoint16 = structure:GetNodeIndex(016)
518 basepoint17 = structure:GetNodeIndex(017)
519 basepoint18 = structure:GetNodeIndex(018)
520
521 topnode1 = structure:GetNodeIndex(901)
522 topnode2 = structure:GetNodeIndex(902)
523 topnode3 = structure:GetNodeIndex(903)
524 topnode4 = structure:GetNodeIndex(904)
525 topnode5 = structure:GetNodeIndex(905)
526 topnode6 = structure:GetNodeIndex(906)
527 topnode7 = structure:GetNodeIndex(907)
528 topnode8 = structure:GetNodeIndex(908)
529 topnode9 = structure:GetNodeIndex(909)
530 topnode10 = structure:GetNodeIndex(910)
531 topnode11 = structure:GetNodeIndex(911)
532 topnode12 = structure:GetNodeIndex(912)
533 topnode13 = structure:GetNodeIndex(913)
534 topnode14 = structure:GetNodeIndex(914)
535 topnode15 = structure:GetNodeIndex(915)
536 topnode16 = structure:GetNodeIndex(916)
537 topnode17 = structure:GetNodeIndex(917)
538 topnode18 = structure:GetNodeIndex(918)
539
540 -- plotting structure
541 --[[    g = graph3d.Graph3D("structure", 255, 255, 255, 255)
542 g:Clear()
543 tris = structure:Draw()
544 g:Triangles(tris)
545 g:Autoscale()
```

```
546 g:Zoom(0.7)
547 ]]--
```

E.2 Unregelmäßiges L-förmiges Gebäudemodell

modell_asym.tng

```
1  --[[
2  Modelling of the asymmetric building
3  --]]
4
5  -- Create and new FE structure
6  structure=fem.Structure("building")
7
8  --Dimensions of Building
9  a = 10
10 b = 6
11 h = 3.9
12
13 --Define node coordinates of corner nodes
14
15 nodes = tmath.Matrix({
16
17 {001, 0.0, 0.0, 0.0},
18 {002, a/5, 0.0, 0.0},
19 {003, 2*a/5, 0.0, 0.0},
20 {004, 3*a/5, 0.0, 0.0},
21 {005, 4*a/5, 0.0, 0.0},
22 {006, a, 0.0, 0.0},
23 {007, 0.0, b/2, 0.0},
24 {008, a/5, b/2, 0.0},
25 {009, 2*a/5, b/2, 0.0},
26 {010, 3*a/5, b/2, 0.0},
27 {011, 4*a/5, b/2, 0.0},
28 {012, a, b/2, 0.0},
29 {013, 0.0, b, 0.0},
30 {014, a/5, b, 0.0},
31 {015, 2*a/5, b, 0.0},
32 {016, 3*a/5, b, 0.0},
33 {017, 4*a/5, b, 0.0},
34 {018, a, b, 0.0},
35 {019, 3*a/5, 4*b/3, 0.0},
36 {020, 4*a/5, 4*b/3, 0.0},
37 {021, a, 4*b/3, 0.0},
38 {022, 3*a/5, 5*b/3, 0.0},
39 {023, 4*a/5, 5*b/3, 0.0},
```

```
40 {024, a, 5*b/3, 0.0},
41
42 {101, 0.0, 0.0, h/3},
43 {102, a/5, 0.0, h/3},
44 {103, 2*a/5, 0.0, h/3},
45 {104, 3*a/5, 0.0, h/3},
46 {105, 4*a/5, 0.0, h/3},
47 {106, a, 0.0, h/3},
48 {107, 0.0, b/2, h/3},
49 {108, a/5, b/2, h/3},
50 {109, 2*a/5, b/2, h/3},
51 {110, 3*a/5, b/2, h/3},
52 {111, 4*a/5, b/2, h/3},
53 {112, a, b/2, h/3},
54 {113, 0.0, b, h/3},
55 {114, a/5, b, h/3},
56 {115, 2*a/5, b, h/3},
57 {116, 3*a/5, b, h/3},
58 {117, 4*a/5, b, h/3},
59 {118, a, b, h/3},
60 {119, 3*a/5, 4*b/3, h/3},
61 {120, 4*a/5, 4*b/3, h/3},
62 {121, a, 4*b/3, h/3},
63 {122, 3*a/5, 5*b/3, h/3},
64 {123, 4*a/5, 5*b/3, h/3},
65 {124, a, 5*b/3, h/3},
66
67 {201, 0.0, 0.0, 2/3*h},
68 {202, a/5, 0.0, 2/3*h},
69 {203, 2*a/5, 0.0, 2/3*h},
70 {204, 3*a/5, 0.0, 2/3*h},
71 {205, 4*a/5, 0.0, 2/3*h},
72 {206, a, 0.0, 2/3*h},
73 {207, 0.0, b/2, 2/3*h},
74 {208, a/5, b/2, 2/3*h},
75 {209, 2*a/5, b/2, 2/3*h},
76 {210, 3*a/5, b/2, 2/3*h},
77 {211, 4*a/5, b/2, 2/3*h},
78 {212, a, b/2, 2/3*h},
79 {213, 0.0, b, 2/3*h},
80 {214, a/5, b, 2/3*h},
81 {215, 2*a/5, b, 2/3*h},
82 {216, 3*a/5, b, 2/3*h},
83 {217, 4*a/5, b, 2/3*h},
84 {218, a, b, 2/3*h},
85 {219, 3*a/5, 4*b/3, 2/3*h},
86 {220, 4*a/5, 4*b/3, 2/3*h},
87 {221, a, 4*b/3, 2/3*h},
88 {222, 3*a/5, 5*b/3, 2/3*h},
```

```
89  {223, 4*a/5, 5*b/3, 2/3*h},
90  {224, a, 5*b/3, 2/3*h},
91
92  {301, 0.0, 0.0, h},
93  {302, a/5, 0.0, h},
94  {303, 2*a/5, 0.0, h},
95  {304, 3*a/5, 0.0, h},
96  {305, 4*a/5, 0.0, h},
97  {306, a, 0.0, h},
98  {307, 0.0, b/2, h},
99  {308, a/5, b/2, h},
100 {309, 2*a/5, b/2, h},
101 {310, 3*a/5, b/2, h},
102 {311, 4*a/5, b/2, h},
103 {312, a, b/2, h},
104 {313, 0.0, b, h},
105 {314, a/5, b, h},
106 {315, 2*a/5, b, h},
107 {316, 3*a/5, b, h},
108 {317, 4*a/5, b, h},
109 {318, a, b, h},
110 {319, 3*a/5, 4*b/3, h},
111 {320, 4*a/5, 4*b/3, h},
112 {321, a, 4*b/3, h},
113 {322, 3*a/5, 5*b/3, h},
114 {323, 4*a/5, 5*b/3, h},
115 {324, a, 5*b/3, h},
116
117 {401, 0.0, 0.0, 4/3*h},
118 {402, a/5, 0.0, 4/3*h},
119 {403, 2*a/5, 0.0, 4/3*h},
120 {404, 3*a/5, 0.0, 4/3*h},
121 {405, 4*a/5, 0.0, 4/3*h},
122 {406, a, 0.0, 4/3*h},
123 {407, 0.0, b/2, 4/3*h},
124 {408, a/5, b/2, 4/3*h},
125 {409, 2*a/5, b/2, 4/3*h},
126 {410, 3*a/5, b/2, 4/3*h},
127 {411, 4*a/5, b/2, 4/3*h},
128 {412, a, b/2, 4/3*h},
129 {413, 0.0, b, 4/3*h},
130 {414, a/5, b, 4/3*h},
131 {415, 2*a/5, b, 4/3*h},
132 {416, 3*a/5, b, 4/3*h},
133 {417, 4*a/5, b, 4/3*h},
134 {418, a, b, 4/3*h},
135 {419, 3*a/5, 4*b/3, 4/3*h},
136 {420, 4*a/5, 4*b/3, 4/3*h},
137 {421, a, 4*b/3, 4/3*h},
```

```
138 {422, 3*a/5, 5*b/3, 4/3*h},
139 {423, 4*a/5, 5*b/3, 4/3*h},
140 {424, a, 5*b/3, 4/3*h},
141
142 {501, 0.0, 0.0, 5/3*h},
143 {502, a/5, 0.0, 5/3*h},
144 {503, 2*a/5, 0.0, 5/3*h},
145 {504, 3*a/5, 0.0, 5/3*h},
146 {505, 4*a/5, 0.0, 5/3*h},
147 {506, a, 0.0, 5/3*h},
148 {507, 0.0, b/2, 5/3*h},
149 {508, a/5, b/2, 5/3*h},
150 {509, 2*a/5, b/2, 5/3*h},
151 {510, 3*a/5, b/2, 5/3*h},
152 {511, 4*a/5, b/2, 5/3*h},
153 {512, a, b/2, 5/3*h},
154 {513, 0.0, b, 5/3*h},
155 {514, a/5, b, 5/3*h},
156 {515, 2*a/5, b, 5/3*h},
157 {516, 3*a/5, b, 5/3*h},
158 {517, 4*a/5, b, 5/3*h},
159 {518, a, b, 5/3*h},
160 {519, 3*a/5, 4*b/3, 5/3*h},
161 {520, 4*a/5, 4*b/3, 5/3*h},
162 {521, a, 4*b/3, 5/3*h},
163 {522, 3*a/5, 5*b/3, 5/3*h},
164 {523, 4*a/5, 5*b/3, 5/3*h},
165 {524, a, 5*b/3, 5/3*h},
166
167 {601, 0.0, 0.0, 2*h},
168 {602, a/5, 0.0, 2*h},
169 {603, 2*a/5, 0.0, 2*h},
170 {604, 3*a/5, 0.0, 2*h},
171 {605, 4*a/5, 0.0, 2*h},
172 {606, a, 0.0, 2*h},
173 {607, 0.0, b/2, 2*h},
174 {608, a/5, b/2, 2*h},
175 {609, 2*a/5, b/2, 2*h},
176 {610, 3*a/5, b/2, 2*h},
177 {611, 4*a/5, b/2, 2*h},
178 {612, a, b/2, 2*h},
179 {613, 0.0, b, 2*h},
180 {614, a/5, b, 2*h},
181 {615, 2*a/5, b, 2*h},
182 {616, 3*a/5, b, 2*h},
183 {617, 4*a/5, b, 2*h},
184 {618, a, b, 2*h},
185 {619, 3*a/5, 4*b/3, 2*h},
186 {620, 4*a/5, 4*b/3, 2*h},
```

```
187 {621, a, 4*b/3, 2*h},
188 {622, 3*a/5, 5*b/3, 2*h},
189 {623, 4*a/5, 5*b/3, 2*h},
190 {624, a, 5*b/3, 2*h},
191
192 {701, 0.0, 0.0, 7/3*h},
193 {702, a/5, 0.0, 7/3*h},
194 {703, 2*a/5, 0.0, 7/3*h},
195 {704, 3*a/5, 0.0, 7/3*h},
196 {705, 4*a/5, 0.0, 7/3*h},
197 {706, a, 0.0, 7/3*h},
198 {707, 0.0, b/2, 7/3*h},
199 {708, a/5, b/2, 7/3*h},
200 {709, 2*a/5, b/2, 7/3*h},
201 {710, 3*a/5, b/2, 7/3*h},
202 {711, 4*a/5, b/2, 7/3*h},
203 {712, a, b/2, 7/3*h},
204 {713, 0.0, b, 7/3*h},
205 {714, a/5, b, 7/3*h},
206 {715, 2*a/5, b, 7/3*h},
207 {716, 3*a/5, b, 7/3*h},
208 {717, 4*a/5, b, 7/3*h},
209 {718, a, b, 7/3*h},
210 {719, 3*a/5, 4*b/3, 7/3*h},
211 {720, 4*a/5, 4*b/3, 7/3*h},
212 {721, a, 4*b/3, 7/3*h},
213 {722, 3*a/5, 5*b/3, 7/3*h},
214 {723, 4*a/5, 5*b/3, 7/3*h},
215 {724, a, 5*b/3, 7/3*h},
216
217 {801, 0.0, 0.0, 8/3*h},
218 {802, a/5, 0.0, 8/3*h},
219 {803, 2*a/5, 0.0, 8/3*h},
220 {804, 3*a/5, 0.0, 8/3*h},
221 {805, 4*a/5, 0.0, 8/3*h},
222 {806, a, 0.0, 8/3*h},
223 {807, 0.0, b/2, 8/3*h},
224 {808, a/5, b/2, 8/3*h},
225 {809, 2*a/5, b/2, 8/3*h},
226 {810, 3*a/5, b/2, 8/3*h},
227 {811, 4*a/5, b/2, 8/3*h},
228 {812, a, b/2, 8/3*h},
229 {813, 0.0, b, 8/3*h},
230 {814, a/5, b, 8/3*h},
231 {815, 2*a/5, b, 8/3*h},
232 {816, 3*a/5, b, 8/3*h},
233 {817, 4*a/5, b, 8/3*h},
234 {818, a, b, 8/3*h},
235 {819, 3*a/5, 4*b/3, 8/3*h},
```

```
{820, 4*a/5, 4*b/3, 8/3*h},
{821, a, 4*b/3, 8/3*h},
{822, 3*a/5, 5*b/3, 8/3*h},
{823, 4*a/5, 5*b/3, 8/3*h},
{824, a, 5*b/3, 8/3*h},

{901, 0.0, 0.0, 3*h},
{902, a/5, 0.0, 3*h},
{903, 2*a/5, 0.0, 3*h},
{904, 3*a/5, 0.0, 3*h},
{905, 4*a/5, 0.0, 3*h},
{906, a, 0.0, 3*h},
{907, 0.0, b/2, 3*h},
{908, a/5, b/2, 3*h},
{909, 2*a/5, b/2, 3*h},
{910, 3*a/5, b/2, 3*h},
{911, 4*a/5, b/2, 3*h},
{912, a, b/2, 3*h},
{913, 0.0, b, 3*h},
{914, a/5, b, 3*h},
{915, 2*a/5, b, 3*h},
{916, 3*a/5, b, 3*h},
{917, 4*a/5, b, 3*h},
{918, a, b, 3*h},
{919, 3*a/5, 4*b/3, 3*h},
{920, 4*a/5, 4*b/3, 3*h},
{921, a, 4*b/3, 3*h},
{922, 3*a/5, 5*b/3, 3*h},
{923, 4*a/5, 5*b/3, 3*h},
{924, a, 5*b/3, 3*h},
})

structure:AddNodes(nodes)


-- Define support conditions

structure:GetNode(001):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
    1, 1}}))
structure:GetNode(002):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
    1, 1}}))
structure:GetNode(003):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
    1, 1}}))
structure:GetNode(004):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
    1, 1}}))
structure:GetNode(005):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
    1, 1}}))
structure:GetNode(006):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
    1, 1}}))
```

```
279 structure:GetNode(007):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
280 structure:GetNode(008):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
281 structure:GetNode(009):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
282 structure:GetNode(010):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
283 structure:GetNode(011):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
284 structure:GetNode(012):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
285 structure:GetNode(013):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
286 structure:GetNode(014):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
287 structure:GetNode(015):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
288 structure:GetNode(016):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
289 structure:GetNode(017):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
290 structure:GetNode(018):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
291 structure:GetNode(019):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
292 structure:GetNode(020):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
293 structure:GetNode(021):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
294 structure:GetNode(022):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
295 structure:GetNode(023):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
296 structure:GetNode(024):SetAvailDof(tmath.Matrix({{0, 0, 0, 1,
        1, 1}}))
297
298 -- Define material --- e+6 = N/mm2
299 mas = structure:AddMaterial(001, 'ELASTIC_FAILURE')
300 mas:SetData(tmath.Matrix({{1.6e+9, 0.2 , 1700, -5e+6, 4e+5
      }}))
301
302 timber = structure:AddMaterial(002, 'ELASTIC_FAILURE') --slab
       elastic failure
303 timber:SetData(tmath.Matrix({{3.7e+8, 0.3 , 500, -3e+6, 5.5e
      +6 }})) --E-Modul 3.7e+8 (in Querrichtung), 1.1e+10 (in
      Faserrichtung)
304
305 -- Define cross sections
```

```
306 sec = structure:AddSection(001, 'SHELL') -- wall
307 sec:SetData(tmath.Matrix({{0.45}}))
308 sec:SetColor(tmath.Matrix({{128, 50, 50, 100}}))
309
310 sec = structure:AddSection(010, 'SHELL') -- shear wall
311 sec:SetData(tmath.Matrix({{0.30}}))
312 sec:SetColor(tmath.Matrix({{128, 50, 50, 100}}))
313
314 sec = structure:AddSection(100, 'SHELL') -- wooden slab
315 sec:SetData(tmath.Matrix({{0.16}}))
316 sec:SetColor(tmath.Matrix({{128, 128, 128, 100}}))
317
318 sec = structure:AddSection(101, 'SHELL') -- concrete slab
319 sec:SetData(tmath.Matrix({{0.2}}))
320 sec:SetColor(tmath.Matrix({{128, 128, 128, 100}}))
321
322 -- Add elements to structure
323 --storey 0
324 --walls a
325 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0001, 001, 002, 102, 101}}))
326 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0002, 003, 004, 104, 103}}))
327 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0003, 005, 006, 106, 105}}))
328 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0004, 007, 008, 108, 107}}))
329 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0005, 008, 009, 109, 108}}))
330 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
      ({{ 0006, 009, 010, 110, 109}}))
331 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0007, 010, 011, 111, 110}}))
332 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0008, 011, 012, 112, 111}}))
333 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0009, 013, 014, 114, 113}}))
334 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0010, 015, 016, 116, 115}}))
335 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0011, 017, 018, 118, 117}}))
336 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0020, 022, 023, 123, 122}}))
337 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      0021, 023, 024, 124, 123}}))
338 --walls b
339 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      0012, 001, 007, 107, 101}}))
```

```
340 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0013, 007, 013, 113, 107}}))
341 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0014, 003, 009, 109, 103}}))
342 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0015, 009, 015, 115, 109}}))
343 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0016, 004, 010, 110, 104}}))
344 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0017, 010, 016, 116, 110}}))
345 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0018, 006, 012, 112, 106}}))
346 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0019, 012, 018, 118, 112}}))
347 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0022, 016, 019, 119, 116}}))
348 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0023, 018, 021, 121, 118}}))
349 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       0024, 021, 024, 124, 121}}))
350
351 --walls a
352 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1001, 101, 102, 202, 201}}))
353 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1002, 103, 104, 204, 203}}))
354 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1003, 105, 106, 206, 205}}))
355 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1004, 107, 108, 208, 207}}))
356 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1005, 108, 109, 209, 208}}))
357 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 1006, 109, 110, 210, 209}}))
358 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1007, 110, 111, 211, 210}}))
359 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1008, 111, 112, 212, 211}}))
360 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1009, 113, 114, 214, 213}}))
361 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1010, 115, 116, 216, 215}}))
362 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1011, 117, 118, 218, 217}}))
363 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1020, 122, 123, 223, 222}}))
364 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       1021, 123, 124, 224, 223}}))
365 --walls b
```

```
366 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1012, 101, 107, 207, 201}}))
367 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1013, 107, 113, 213, 207}}))
368 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1014, 103, 109, 209, 203}}))
369 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1015, 109, 115, 215, 209}}))
370 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1016, 104, 110, 210, 204}}))
371 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1017, 110, 116, 216, 210}}))
372 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1018, 106, 112, 212, 206}}))
373 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1019, 112, 118, 218, 212}}))
374 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1022, 116, 119, 219, 216}}))
375 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1023, 118, 121, 221, 218}}))
376 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       1024, 121, 124, 224, 221}}))
377
378 --walls a
379 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2001, 201, 202, 302, 301}}))
380 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2002, 203, 204, 304, 303}}))
381 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2003, 205, 206, 306, 305}}))
382 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2004, 207, 208, 308, 307}}))
383 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2005, 208, 209, 309, 308}}))
384 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 2006, 209, 210, 310, 309}}))
385 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2007, 210, 211, 311, 310}}))
386 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2008, 211, 212, 312, 311}}))
387 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2009, 213, 214, 314, 313}}))
388 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2010, 215, 216, 316, 315}}))
389 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2011, 217, 218, 318, 317}}))
390 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2020, 222, 223, 323, 322}}))
```

```
391 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       2021, 223, 224, 324, 323}}))
392 --walls b
393 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2012, 201, 207, 307, 301}}))
394 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2013, 207, 213, 313, 307}}))
395 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2014, 203, 209, 309, 303}}))
396 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2015, 209, 215, 315, 309}}))
397 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2016, 204, 210, 310, 304}}))
398 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2017, 210, 216, 316, 310}}))
399 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2018, 206, 212, 312, 306}}))
400 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2019, 212, 218, 318, 312}}))
401 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2022, 216, 219, 319, 316}}))
402 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2023, 218, 221, 321, 318}}))
403 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       2024, 221, 224, 324, 321}}))
404
405 --slab
406 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2101, 301, 302, 308, 307}}))
407 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2102, 302, 303, 309, 308}}))
408 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2103, 303, 304, 310, 309}}))
409 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2104, 304, 305, 311, 310}}))
410 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2105, 305, 306, 312, 311}}))
411 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2106, 307, 308, 314, 313}}))
412 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2107, 308, 309, 315, 314}}))
413 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2108, 309, 310, 316, 315}}))
414 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2109, 310, 311, 317, 316}}))
415 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2110, 311, 312, 318, 317}}))
416 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       2111, 316, 317, 320, 319}}))
```

```
417 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      2112, 317, 318, 321, 320}}))
418 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      2113, 319, 320, 323, 322}}))
419 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
      2114, 320, 321, 324, 323}}))
420
421 --storey 1
422
423 --walls a
424 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3001, 301, 302, 402, 401}}))
425 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3002, 303, 304, 404, 403}}))
426 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3003, 305, 306, 406, 405}}))
427 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3004, 307, 308, 408, 407}}))
428 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3005, 308, 309, 409, 408}}))
429 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
      ({{ 3006, 309, 310, 410, 409}}))
430 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3007, 310, 311, 411, 410}}))
431 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3008, 311, 312, 412, 411}}))
432 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3009, 313, 314, 414, 413}}))
433 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3010, 315, 316, 416, 415}}))
434 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3011, 317, 318, 418, 417}}))
435 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3020, 322, 323, 423, 422}}))
436 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
      3021, 323, 324, 424, 423}}))
437 --walls b
438 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      3012, 301, 307, 407, 401}}))
439 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      3013, 307, 313, 413, 407}}))
440 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      3014, 303, 309, 409, 403}}))
441 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      3015, 309, 315, 415, 409}}))
442 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      3016, 304, 310, 410, 404}}))
443 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
      3017, 310, 316, 416, 410}}))
```

```
444 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3018, 306, 312, 412, 406}}))
445 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3019, 312, 318, 418, 412}}))
446 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3022, 316, 319, 419, 416}}))
447 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3023, 318, 321, 421, 418}}))
448 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       3024, 321, 324, 424, 421}}))
449
450 --walls a
451 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4001, 401, 402, 502, 501}}))
452 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4002, 403, 404, 504, 503}}))
453 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4003, 405, 406, 506, 505}}))
454 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4004, 407, 408, 508, 507}}))
455 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4005, 408, 409, 509, 508}}))
456 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 4006, 409, 410, 510, 509}}))
457 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4007, 410, 411, 511, 510}}))
458 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4008, 411, 412, 512, 511}}))
459 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4009, 413, 414, 514, 513}}))
460 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4010, 415, 416, 516, 515}}))
461 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4011, 417, 418, 518, 517}}))
462 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4020, 422, 423, 523, 522}}))
463 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       4021, 423, 424, 524, 523}}))
464 --walls b
465 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4012, 401, 407, 507, 501}}))
466 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4013, 407, 413, 513, 507}}))
467 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4014, 403, 409, 509, 503}}))
468 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4015, 409, 415, 515, 509}}))
469 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       4016, 404, 410, 510, 504}}))
```

```
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    4017, 410, 416, 516, 510}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    4018, 406, 412, 512, 506}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    4019, 412, 418, 518, 512}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    4022, 416, 419, 519, 516}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    4023, 418, 421, 521, 518}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    4024, 421, 424, 524, 521}}))

--walls a
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5001, 501, 502, 602, 601}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5002, 503, 504, 604, 603}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5003, 505, 506, 606, 605}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5004, 507, 508, 608, 607}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5005, 508, 509, 609, 608}}))
--    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
    ({{ 5006, 509, 510, 610, 609}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5007, 510, 511, 611, 610}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5008, 511, 512, 612, 611}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5009, 513, 514, 614, 613}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5010, 515, 516, 616, 615}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5011, 517, 518, 618, 617}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5020, 522, 523, 623, 622}}))
structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
    5021, 523, 524, 624, 623}}))
--walls b
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    5012, 501, 507, 607, 601}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    5013, 507, 513, 613, 607}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    5014, 503, 509, 609, 603}}))
structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
    5015, 509, 515, 615, 609}}))
```

```
496 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5016, 504, 510, 610, 604}}))
497 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5017, 510, 516, 616, 610}}))
498 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5018, 506, 512, 612, 606}}))
499 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5019, 512, 518, 618, 612}}))
500 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5022, 516, 519, 619, 616}}))
501 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5023, 518, 521, 621, 618}}))
502 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       5024, 521, 524, 624, 621}}))
503
504 --slab
505 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5101, 601, 602, 608, 607}}))
506 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5102, 602, 603, 609, 608}}))
507 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5103, 603, 604, 610, 609}}))
508 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5104, 604, 605, 611, 610}}))
509 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5105, 605, 606, 612, 611}}))
510 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5106, 607, 608, 614, 613}}))
511 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5107, 608, 609, 615, 614}}))
512 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5108, 609, 610, 616, 615}}))
513 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5109, 610, 611, 617, 616}}))
514 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5110, 611, 612, 618, 617}}))
515 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5111, 616, 617, 620, 619}}))
516 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5112, 617, 618, 621, 620}}))
517 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5113, 619, 620, 623, 622}}))
518 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       5114, 620, 621, 624, 623}}))
519
520 --storey 2
521
522 --walls a
```

```
523 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6001, 601, 602, 702, 701}}))
524 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6002, 603, 604, 704, 703}}))
525 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6003, 605, 606, 706, 705}}))
526 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6004, 607, 608, 708, 707}}))
527 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6005, 608, 609, 709, 708}}))
528 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 6006, 609, 610, 710, 709}}))
529 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6007, 610, 611, 711, 710}}))
530 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6008, 611, 612, 712, 711}}))
531 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6009, 613, 614, 714, 713}}))
532 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6010, 615, 616, 716, 715}}))
533 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6011, 617, 618, 718, 717}}))
534 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6020, 622, 623, 723, 722}}))
535 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       6021, 623, 624, 724, 723}}))
536 --walls b
537 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6012, 601, 607, 707, 701}}))
538 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6013, 607, 613, 713, 707}}))
539 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6014, 603, 609, 709, 703}}))
540 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6015, 609, 615, 715, 709}}))
541 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6016, 604, 610, 710, 704}}))
542 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6017, 610, 616, 716, 710}}))
543 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6018, 606, 612, 712, 706}}))
544 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6019, 612, 618, 718, 712}}))
545 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6022, 616, 619, 719, 716}}))
546 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6023, 618, 621, 721, 718}}))
547 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       6024, 621, 624, 724, 721}}))
```

```
548
549 --walls a
550 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7001, 701, 702, 802, 801}}))
551 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7002, 703, 704, 804, 803}}))
552 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7003, 705, 706, 806, 805}}))
553 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7004, 707, 708, 808, 807}}))
554 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7005, 708, 709, 809, 808}}))
555 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 7006, 709, 710, 810, 809}}))
556 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7007, 710, 711, 811, 810}}))
557 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7008, 711, 712, 812, 811}}))
558 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7009, 713, 714, 814, 813}}))
559 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7010, 715, 716, 816, 815}}))
560 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7011, 717, 718, 818, 817}}))
561 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7020, 722, 723, 823, 822}}))
562 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       7021, 723, 724, 824, 823}}))
563 --walls b
564 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7012, 701, 707, 807, 801}}))
565 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7013, 707, 713, 813, 807}}))
566 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7014, 703, 709, 809, 803}}))
567 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7015, 709, 715, 815, 809}}))
568 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7016, 704, 710, 810, 804}}))
569 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7017, 710, 716, 816, 810}}))
570 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7018, 706, 712, 812, 806}}))
571 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7019, 712, 718, 818, 812}}))
572 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7022, 716, 719, 819, 816}}))
573 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7023, 718, 721, 821, 818}}))
```

```
574 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       7024, 721, 724, 824, 821}}))
575
576 --walls a
577 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8001, 801, 802, 902, 901}}))
578 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8002, 803, 804, 904, 903}}))
579 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8003, 805, 806, 906, 905}}))
580 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8004, 807, 808, 908, 907}}))
581 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8005, 808, 809, 909, 908}}))
582 --    structure:AddElements('RQUAD', 001, 001, tmath.Matrix
       ({{ 8006, 809, 810, 910, 909}}))
583 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8007, 810, 811, 911, 910}}))
584 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8008, 811, 812, 912, 911}}))
585 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8009, 813, 814, 914, 913}}))
586 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8010, 815, 816, 916, 915}}))
587 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8011, 817, 818, 918, 917}}))
588 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8020, 822, 823, 923, 922}}))
589 structure:AddElements('RQUAD', 001, 001, tmath.Matrix({{
       8021, 823, 824, 924, 923}}))
590 --walls b
591 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8012, 801, 807, 907, 901}}))
592 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8013, 807, 813, 913, 907}}))
593 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8014, 803, 809, 909, 903}}))
594 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8015, 809, 815, 915, 909}}))
595 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8016, 804, 810, 910, 904}}))
596 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8017, 810, 816, 916, 910}}))
597 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8018, 806, 812, 912, 906}}))
598 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8019, 812, 818, 918, 912}}))
599 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8022, 816, 819, 919, 916}}))
```

```
600 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8023, 818, 821, 921, 918}}))
601 structure:AddElements('RQUAD', 001, 010, tmath.Matrix({{
       8024, 821, 824, 924, 921}}))
602
603 --slab
604 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8101, 901, 902, 908, 907}}))
605 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8102, 902, 903, 909, 908}}))
606 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8103, 903, 904, 910, 909}}))
607 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8104, 904, 905, 911, 910}}))
608 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8105, 905, 906, 912, 911}}))
609 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8106, 907, 908, 914, 913}}))
610 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8107, 908, 909, 915, 914}}))
611 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8108, 909, 910, 916, 915}}))
612 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8109, 910, 911, 917, 916}}))
613 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8110, 911, 912, 918, 917}}))
614 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8111, 916, 917, 920, 919}}))
615 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8112, 917, 918, 921, 920}}))
616 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8113, 919, 920, 923, 922}}))
617 structure:AddElements('RQUAD', 002, 100, tmath.Matrix({{
       8114, 920, 921, 924, 923}}))
618
619
620
621 -- Assemble all dofs
622 ndof = structure:GlobalDof()
623 --print(ndof)
624
625 basepoint1 = structure:GetNodeIndex(001)
626 basepoint2 = structure:GetNodeIndex(002)
627 basepoint3 = structure:GetNodeIndex(003)
628 basepoint4 = structure:GetNodeIndex(004)
629 basepoint5 = structure:GetNodeIndex(005)
630 basepoint6 = structure:GetNodeIndex(006)
631 basepoint7 = structure:GetNodeIndex(007)
632 basepoint8 = structure:GetNodeIndex(008)
```

```
633 basepoint9 = structure:GetNodeIndex(009)
634 basepoint10 = structure:GetNodeIndex(010)
635 basepoint11 = structure:GetNodeIndex(011)
636 basepoint12 = structure:GetNodeIndex(012)
637 basepoint13 = structure:GetNodeIndex(013)
638 basepoint14 = structure:GetNodeIndex(014)
639 basepoint15 = structure:GetNodeIndex(015)
640 basepoint16 = structure:GetNodeIndex(016)
641 basepoint17 = structure:GetNodeIndex(017)
642 basepoint18 = structure:GetNodeIndex(018)
643 basepoint19 = structure:GetNodeIndex(019)
644 basepoint20 = structure:GetNodeIndex(020)
645 basepoint21 = structure:GetNodeIndex(021)
646 basepoint22 = structure:GetNodeIndex(022)
647 basepoint23 = structure:GetNodeIndex(023)
648 basepoint24 = structure:GetNodeIndex(024)
649
650 topnode1 = structure:GetNodeIndex(901)
651 topnode2 = structure:GetNodeIndex(902)
652 topnode3 = structure:GetNodeIndex(903)
653 topnode4 = structure:GetNodeIndex(904)
654 topnode5 = structure:GetNodeIndex(905)
655 topnode6 = structure:GetNodeIndex(906)
656 topnode7 = structure:GetNodeIndex(907)
657 topnode8 = structure:GetNodeIndex(908)
658 topnode9 = structure:GetNodeIndex(909)
659 topnode10 = structure:GetNodeIndex(910)
660 topnode11 = structure:GetNodeIndex(911)
661 topnode12 = structure:GetNodeIndex(912)
662 topnode13 = structure:GetNodeIndex(913)
663 topnode14 = structure:GetNodeIndex(914)
664 topnode15 = structure:GetNodeIndex(915)
665 topnode16 = structure:GetNodeIndex(916)
666 topnode17 = structure:GetNodeIndex(917)
667 topnode18 = structure:GetNodeIndex(918)
668 topnode19 = structure:GetNodeIndex(919)
669 topnode20 = structure:GetNodeIndex(920)
670 topnode21 = structure:GetNodeIndex(921)
671 topnode22 = structure:GetNodeIndex(922)
672 topnode23 = structure:GetNodeIndex(923)
673 topnode24 = structure:GetNodeIndex(924)
674
675 -- plotting structure
676 --[[    g = graph3d.Graph3D("structure", 255, 255, 255, 255)
677 g:Clear()
678 tris = structure:Draw()
679 g:Triangles(tris)
680 g:Autoscale()
681 g:Zoom(0.7)
```

```
682 ]]--
```

E.3 Pushover-Analyse

push_weg.tng

```
1  --push over analysis (adaptiert nach Bucher)
2
3  --load structure modell_sym
4  dofile("modell_asym.tng")
5
6  -- Number of load steps
7  N = 1000
8
9  -- perform pushover analysis for shape 0, 1, 2, ...
10 mode_shape = 0
11
12 --axis direction in pushover graph
13 -- 0=x, 1=y
14 push_dir = 0
15
16 --defination of topnode
17 topnode = topnode10
18
19 X = structure:GetCoordinates()
20 z_topnode = X[{topnode, 2}]
21 print("z_topnode", z_topnode)
22 u_max = X[{topnode, 2}]*0.004
23 print("u_max", u_max)
24 du = u_max/(N-1)
25
26 -- Assemble stiffness and mass matrices (sparse)
27 KK = structure:SparseStiffness()
28 MM = structure:SparseMass()
29
30 -- Influence vector gravity load
31 F1=structure:GetAllDisplacements()
32 col = F1:GetCols(2)
33 F1:SetZero()
34 col:SetConstant(1)
35 F1:SetCols(col, 2)
36
37 -- Bring load case into vector form and multiply unit
      acceleration with mass matrix
38 F=structure:ToDofDisplacements(F1)
39
```

```
g = 9.81
TotalMass = MM*F
print("G", TotalMass)
print("Total Mass",  tmath.Sum(TotalMass))

G = MM*F*(-g)
UG = KK:Solve(G)
structure:SetDofDisplacements(UG)
KG = structure:SparseGeoStiffness()
--print("KG: ", KG)
N_elem = structure:GetNumberOfElements()
N_nodes = structure:GetNumberOfNodes()
m = MM:Expand()

--Compute first xxx eigenvalues and mode shapes
val,vec = KK:Eigen(MM, N_elem)
freq = val:CW():Sqrt()/2/math.pi

T=tmath.Matrix (freq:Rows())

for i=0, freq:Rows()-1 do
T[i] = 1/freq[i]
end

print("freq", freq) --frequency in Hz
print("T", T) --period in s

-- Show mode_shape

shape = vec:GetCols(mode_shape)
structure:SetDofDisplacements(shape)
gm = graph3d.Graph3D("Mode "..mode_shape, 255, 255, 255, 255)
gm:Rotate(-60, 1, 0, 1)
--       g[i]:Rotate(-90, 1, 0, 0)
plot = structure:Draw(1000)
gm:Triangles(plot)
gm:Autoscale()
gm:Render()

--load vector normalized to push direction

--print("shape", shape, shape:MinCoeff(), shape:MaxCoeff())
print("shape Min", shape:MinCoeff())
print("shape Max", shape:MaxCoeff())
shape = vec:GetCols(mode_shape)
shape1 = structure:ToAllDisplacements(shape)
shape1 = shape1/shape1[{topnode,push_dir}]
shape = structure:ToDofDisplacements(shape1)
--print("shape", shape)
```

```
89  print("shape Min", shape:MinCoeff())
90  print("shape Max", shape:MaxCoeff())
91
92
93  disp_load = tmath.Matrix(1, 2)
94  disp_load_local = tmath.Matrix(1, 2)
95
96  -- Prepare graphics
97  g = graph3d.Graph3D("Push", 255, 255, 255, 255)
98  g:SetCulling(false)
99  g:Rotate(-90, 1, 0, 0)
100 g:Rotate(0, 0, 1, 0)
101 plot = structure:Draw(100)
102 g:Triangles(plot)
103 g:Autoscale()
104 g:Render()
105
106 disp_load[{0,0}] = 0
107 disp_load[{0,1}] = 0
108 disp_load_local[{0,0}] = 0
109 disp_load_local[{0,1}] = 0
110
111 -- setup Force-Displacement Graph
112 v2=graph.Graph("Response", "Bright")
113 v2:AxisLabels("Displacement [m]", "Base Shear [N]")
114
115 U = UG+shape*0 -- Initial displacement is only due to dead
       load
116
117 for i=1,N-1 do
118
119 --adaptation of geomentric stiffness
120 UG = KK:Solve(G)
121 structure:SetDofDisplacements(UG)
122 KG = structure:SparseGeoStiffness()
123
124 -- Assign displacement
125 U1 = UG+shape*i*du
126 structure:SetDofDisplacements(U1)
127
128 U = structure:ToAllDisplacements(U1-UG)
129 disp = structure:GetAllDisplacements()
130 disp_load_local[{0,0}] = disp[{topnode,push_dir}]
131
132 -- compute internal forces
133 Fr = structure:GlobalResForce()
134
135 -- add P_Delta effect
136 Fg = KG*U1
```

```
137 F = Fr-Fg
138
139 -- check for singularity in K
140 KK = structure:SparseStiffness()
141 Ucheck=KK:Solve(F)
142
143 -- Convert to matrix representation and add up all x-
    components
144 F1 = structure:ToAllDisplacements(F)
145 sum = tmath.Sum(F1:GetCols(push_dir))
146 disp_load_local[{0,1}] = sum
147 disp_load = disp_load:AppendRows(disp_load_local)
148
149 g:Clear()
150 plot = structure:Draw(100) -- Deformation with scale 100
151 g:Triangles(plot)
152 g:Render()
153
154 v2:Plot(disp_load:GetCols(0), disp_load:GetCols(1), 1)
155 v2:Render()
156
157 -- write vector disp_load in txt file
158 tmath.Output(disp_load ,"disp_load.txt")
159
160 -- write failure matrices in txt file
161 tmath.Output(structure:GetFailureTension(),"
    FailureTension.txt")
162 tmath.Output(structure:GetFailureCompression(),"
    FailureCompression.txt")
163
164 -- print("Failure Tension", structure:GetFailureTension())
165 -- print("Failure Compression", structure:
    GetFailureCompression())
166
167 end --for i
```

E.4 Nichtlineare Zeitverlaufsberechnung

dynam_RQuad.tng

```
1 --time integraion analysis (adaptiert nach Bucher)
2
3 --load structur modell
4 dofile("modell_sym.tng")
5
6 -- direction of ground accelaration
```

```
-- 0 = x, 1 = y
a_dir = 1

--defination of topnode and basepoint
topnode = topnode10
basepoint = basepoint10

g = 9.81

--setup displacement vector
disp_top_bottom = tmath.Matrix(1, 2)
disp_top_bottom_local = tmath.Matrix(1, 2)
res_force = tmath.Matrix(1, 2)
res_force_local = tmath.Matrix(1, 2)

-- Assemble stiffness and mass matrices (sparse)
KK = structure:SparseStiffness()
MM = structure:SparseMass()

-- Influence vector gravity load
F1=structure:GetAllDisplacements()
col = F1:GetCols(2)
F1:SetZero()
col:SetConstant(1)
F1:SetCols(col, 2)


-- Bring load case into vector form and multiply unit
   acceleration with mass matrix
F=structure:ToDofDisplacements(F1)

G = MM*F*(-g)
UG = KK:Solve(G)
structure:SetDofDisplacements(UG)
KG = structure:SparseGeoStiffness()

-- Read ground acceleration data
Earthquake = tmath.MatrixInput("Earthquake.txt");
dt = Earthquake[1] - Earthquake[0]
N = Earthquake:Rows()
quake=Earthquake:GetCols(1)*1

--graph acceleration-timestep
v0=graph.Graph("Acceleration", "Bright")
v0:AxisLabels("Time Step [s]", "Acceleration [m/s2]")
v0:Plot(Earthquake:GetCols(0), Earthquake:GetCols(1), 1)
v0:Render()

-- Prepare Newmark
```

```
55  a0 = 4/dt^2
56  a1 = 2/dt
57  a2 = 4/dt
58  a3 = 1
59  a4 = 1
60  a5 = 0
61  a6 = dt/2
62  a7 = dt/2
63
64  -- Effective "stiffness" for Newmark method
65  Keff1 = KK:Add(MM, a0)
66
67  val, vec = KK:Eigen(MM, 1)
68  omega = math.sqrt(val[0])
69  zeta = 0.05
70  beta = 2*zeta*omega
71  CC = MM*beta
72  Keff = Keff1:Add(CC, a1)
73
74  -- Define a load case (unit acceleration in x or y-direction)
75  F1=structure:GetAllDisplacements()
76  col = F1:GetCols(a_dir)
77  F1:SetZero()
78  col:SetConstant(1)
79  F1:SetCols(col, a_dir)
80
81  -- Bring load case into vector form and multiply unit
       acceleration with mass matrix
82  F2=structure:ToDofDisplacements(F1)
83  P = MM*F2
84
85  U = tmath.Matrix(UG)
86  V = tmath.Matrix(U)
87  V:SetZero()
88  A = tmath.Matrix(V)
89  A:SetZero()
90
91  -- Prepare graphics
92  g = graph3d.Graph3D("structure", 220, 220, 255)
93  g:SetCulling(false)
94  g:Rotate(-60, 10, 0, 10)
95  plot = structure:Draw(1)
96  g:Triangles(plot)
97  g:Autoscale()
98  g:Zoom(0.75)
99
100 -- prepare element stress vector
101 Nelem = structure:GetNumberOfElements()
102 print("Nelem", Nelem)
```

```
103 -- elementstress = tmath.Matrix(Nelem)
104
105 -- setup Force-Displacement Graph
106 v1=graph.Graph("Response", "Bright")
107 v1:AxisLabels("Time Step [s]", "Displacement [m]")
108 v2=graph.Graph("Response", "Bright")
109 v2:AxisLabels("Time Step [s]", "Base Shear [N]")
110
111 disp = structure:GetAllDisplacements()
112 rel_disp_x = disp[{topnode,0}] - disp[{basepoint,0}]
113 rel_disp_y = disp[{topnode,1}] - disp[{basepoint,1}]
114
115 print("rel_disp_x_0", rel_disp_x)
116 print("rel_disp_y_0", rel_disp_y)
117
118 -- Loop over time steps
119 for i=0,N-1 do
120 accel = quake[i]
121 R1 =  P*(-accel) + MM*(A + V*a2 + U*a0) + CC*(U*a1 + V)
122 U1 = Keff:Solve(R1)
123 for k=0,10 do -- Iteration for dynamic equilibrium
124 structure:SetDofDisplacements(U1)
125 V1 = U1*a1 - U*a1 - V
126 structure:SetDofVelocities(V1)
127 R = R1 - structure:GlobalResForce() - KG*U*0 - MM*U1*a0 - CC*
       U1*a1
128 Rnorm = tmath.Norm(R)
129 if (Rnorm < 1) then break end
130 DU = Keff:Solve(R)
131 U1 = U1+DU
132 end
133 V1 = U1*a1 - U*a1 - V
134 A1 = V1*a1 - V*a1 - A
135 U = tmath.Matrix(U1)
136 V = tmath.Matrix(V1)
137 A = tmath.Matrix(A1)
138 structure:SetDofDisplacements(U)
139 structure:SetDofVelocities(V)
140
141 structure:GlobalUpdate()
142
143 disp = structure:GetAllDisplacements()
144
145 if (i%1==0) then
146 plot = structure:Draw(500) -- Deformation with scale 500
147 g:Clear()
148 g:Triangles(plot)
149 g:Render()
150 end
```

```
151
152 -- print("Failure Tension", structure:GetFailureTension())
153
154 -- write vector disp in txt file
155 disp_top_bottom_local[{0,0}] = disp[{topnode,a_dir}] - disp[{
       basepoint,a_dir}]
156 disp_top_bottom_local[{0,1}] = i*dt
157 disp_top_bottom = disp_top_bottom:AppendRows(
       disp_top_bottom_local)
158
159 --    tmath.Output(disp_top_bottom ,"disp_top_bottom.txt")
160
161 Fr = structure:GlobalResForce()
162
163 -- check for singularity in K
164 KK = structure:SparseStiffness()
165 U=KK:Solve(Fr)
166
167 -- Convert to matrix representation and add up all x-
       components
168 F1 = structure:ToAllDisplacements(Fr)
169 sum = tmath.Sum(F1:GetCols(a_dir))
170 res_force_local[{0,0}] = sum
171 res_force_local[{0,1}] = i*dt
172 res_force = res_force:AppendRows(res_force_local)
173
174 --    tmath.Output(res_force ,"res_force.txt")
175
176 -- write failure matrices in txt file
177 tmath.Output(structure:GetFailureTension()," 
       FailureTension.txt")
178 --    tmath.Output(structure:GetFailureCompression()," 
       FailureCompression.txt")
179
180 --displacement-timestep graph
181 v1:Plot(disp_top_bottom:GetCols(1), disp_top_bottom:GetCols
       (0), 1)
182 v1:Render()
183
184 --base shear-timestep graph
185 v2:Plot(res_force:GetCols(1), res_force:GetCols(0), 1)
186 v2:Render()
187
188 collectgarbage()
189
190 end --for
191 print("rel_disp_x", rel_disp_x)
192 print("rel_disp_y", rel_disp_y)
```