Zur Implementierung höherwertiger Stoffmodelle für Böden am Beispiel der Hypoplastizität

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation

von
Dominik Zobel

aus
Bad Mergentheim

2023

1. Gutachter	Univ.-Prof. Dr.-Ing. Jürgen Grabe
2. Gutachter	Prof. Dr.-Ing. Dimitrios Kolymbas
3. Gutachter	Prof. Dr.-Ing. Alexander Düster

Tag der mündlichen Prüfung: 04.10.2023

Zur Implementierung höherwertiger Stoffmodelle für Böden am Beispiel der Hypoplastizität

von

Dominik Zobel

Herausgegeben von

J. Grabe

Technische Universität Hamburg
Institut für Geotechnik und Baubetrieb

Veröffentlichungen des Instituts für
Geotechnik und Baubetrieb

60

Herausgeber:

Univ.-Prof. Dr.-Ing. Jürgen Grabe
Technische Universität Hamburg
Institut für Geotechnik und Baubetrieb
Harburger Schloßstraße 36
D–21079 Hamburg
E-Mail: *grabe@tuhh.de*

ISBN-13: 978-3-936310-62-7 (Erstausgabe)

DOI: 10.15480/882.8708

Lizenz: CC BY-NC 4.0 (»https://creativecommons.org/licenses/by-nc/4.0/)

Druckerei

Druckzentrum Neumünster GmbH
Rungestraße 4
24537 Neumünster

In derselben Reihe erschienen:

1. Grabe, J. (Hrsg.) (2000). *Verbrennungsrückstände*. Tagungsband. ISBN: 3-936310-00-9.
2. Grabe, J. (Hrsg.) (2001). *Schaden- und Riskikomanagement*. Tagungsband. ISBN: 3-936310-01-7.
3. Grabe, J. (Hrsg.) (2004). *Bodenmechanik und Grundbau. Skriptum zur Vorlesung, 3. Auflage*. Skriptum. ISBN: 3-936310-08-4.
4. Grabe, J. (Hrsg.) (2003). *Euronormen in der Geotechnik – was ändert sich?*. Tagungsband. ISBN: 3-936310-04-1.
5. Grabe, J. (Hrsg.) (2003). *Bodenverdichtung: Experimente – Modellierung – Geräteentwicklung – Baustellenberichte – F+E-Badarf*. Tagungsband. ISBN: 3-936310-05-X.
6. Kelm, M. (2004). *Numerische Simulation der Verdichtung rolliger Böden mittels Vibrationswalzen*. Dissertation. ISBN: 3-936310-06-8.
7. Grabe, J. (Hrsg.) (2004). *Kaimauern – Numerik und Messungen*. Tagungsband. ISBN: 3-936310-07-6.
8. Stein, J. (2004). *Experimentelle und numerische Untersuchungen zum Düsenstrahlverfahren*. Dissertation. ISBN: 3-936310-09-2.
9. Grabe, J. (Hrsg.) (2005). *Grenzschicht Wasser und Boden – Phänomene und Ansätze*. Tagungsband. ISBN: 3-936310-10-6.
10. Grabe, J. (Hrsg.) (2005). *FEM in der Geotechnik – Qualität, Prüfung, Fallbeispiele*. Tagungsband. ISBN: 3-936310-11-4.
11. Mardfeldt, B. (2006). *Zum Tragverhalten von Kaikonstruktionen im Gebrauchszustand*. Dissertation. ISBN: 3-936310-12-2.
12. Grabe, J. (Hrsg.) (2006). *Optimierung in der Geotechnik – Strategien und Fallbeispiele*. Tagungsband. ISBN: 978-3-9363310-13-9.
13. Bierer, T. (2007). *Bodenschwingungen aus Straßenverkehr auf unebener Fahrbahn im Zeitbereich – experimentelle und theoretische Untersuchungen*. Dissertation. ISBN: 978-3-9363310-14-6.
14. Grabe, J. (Hrsg.) (2007). *Bemessen mit Finite-Elemente-Methoden*. Tagungsband. ISBN: 978-3-9363310-15-3.
15. Mahutka, K.-P. (2008). *Zur Verdichtung von rolligen Böden infolge dynamischer Pfahleinbringung und durch Oberflächenrüttler*. Dissertation. ISBN: 978-3-9363310-16-0.
16. Grabe, J. (Hrsg.) (2008). *Seehäfen für Containerschiffe zukünftiger Generationen*. Tagungsband. ISBN: 978-3-9363310-17-7.
17. König, F. (2008). *Zur zeitlichen Traglastentwicklung von Pfählen und der nachträglichen Erweiterung bestehender Pfahlgründungen*. Dissertation. ISBN: 978-3-9363310-18-4.
18. Henke, S. (2009). *Herstellungeinflüsse aus Pfahlrammung im Kaimauerbau*. Dissertation. ISBN: 978-3-9363310-19-1.
19. Grabe, J. (Hrsg.) (2009). *Spundwände – Profile, Tragverhalten, Bemessung, Einbringung und Wiedergewinnung. Tagungsband zum gleichlautenden Workshop an der TU Hamburg-Harburg*. Tagungsband. ISBN: 978-3-9363310-20-7.
20. Dührkop, J. (2010). *Zum Einfluss von Aufweitungen und zyklischen Lasten auf das Verformungsverhalten lateral beanspruchter Pfähle in Sand*. Dissertation. ISBN: 978-3-936310-21-4.
21. Möller, O. (2010). *Zum Langzeit-Kompressionsverhalten weicher organischer Sedimente*. Dissertation. ISBN: 978-3-936310-22-1.

22. Grabe, J. (Hrsg.) (2011). *Ports of container ships of future generations.* Tagungsband. ISBN: 978-3-936310-23-8.

23. Kinzler, S. (2011). *Zur Parameteridentifikation, Entwurfs- und Strukturoptimierung in der Geotechnik mittels numerischer Verfahren.* Dissertation. ISBN: 978-3-936310-24-5.

24. Qiu, G. (2012). *Coupled Eulerian Lagrangian Simulations of Selected Soil-Structure Problems.* Dissertation. ISBN: 978-3-936310-25-2.

25. Ma, X. (2013). *Nutzung der oberflächennahen Geothermie mittels Energiepfählen und Erdwärmesonden.* Dissertation. ISBN: 978-3-936310-26-9.

26. Grabe, J. (Hrsg.) (2013). *Proceedings of Conference on Maritime Energy (COME) 2013.* Tagungsband. ISBN: 978-3-936310-28-3.

27. Grabe, J. (Hrsg.) (2013). *Bemessen mit numerischen Methoden.* Tagungsband. ISBN: 978-3-936310-29-0.

28. Pucker, T. (2013). *Stoffmodell zur Modellierung von stetigen Materialübergängen im Rahmen der Optimierung geotechnischer Strukturen.* Dissertation. ISBN: 978-3-936310-30-6.

29. Henke, S. (2013). *Untersuchungen zur Pfropfenbildung infolge der Installation offener Profile in granularen Böden.* Habilitation. ISBN: 978-3-936310-31-3.

30. Grabe, J. (Hrsg.) (2014). *Ports for Container Ships of Future Generations.* Tagungsband. ISBN: 978-3-936310-32-0.

31. Grabe, J. (Hrsg.) (2014). *Offshore Basishäfen.* Tagungsband. ISBN: 978-3-936310-33-7.

32. Rudolph, C. (2015). *Untersuchungen zur Drift von Pfählen unter zyklischer, lateraler Last aus veränderlicher Richtung.* Dissertation. ISBN: 978-3-936310-34-4.

33. Grabe, J. (Hrsg.) (2015). *Morphodynamics 2015.* Tagungsband. ISBN: 978-3-936310-35-1.

34. Hamann, T. (2015). *Zur Modellierung wassergesättigter Böden unter dynamischer Belastung und großen Bodenverformungen am Beispiel der Pfahleinbringung.* Dissertation. ISBN: 978-3-936310-36-8.

35. Schümann, B. (2015). *Beitrag zum dynamischen Dreiphasenmodell für Boden auf Basis der Finite-Elemente-Methode.* Dissertation. ISBN: 978-3-936310-37-5.

36. Milatz, M. (2016). *Untersuchungen zum Einfluss der Kapillarität auf das hydraulisch-mechanische Verhalten von granularer Tragschichten für Verkehrswege.* Dissertation. ISBN: 978-3-936310-38-2.

37. Kaya, H. (2016). *Bodenverschleppung und Spaltbildung infolge der Einbringung von Profilen in Dichtungsschichten aus Ton.* Dissertation. ISBN: 978-3-936310-39-9.

38. Grabe, J. (Hrsg.) (2017). *Conference on Maritime Energy – Decommissioning of Offshore Geotechnical Structures.* Tagungsband. ISBN: 978-3-936310-40-5.

39. Kocak, B. (2017). *Zur numerischen Modellierung von hydraulisch-mechanisch gekoppelten Prozessen in gesättigten granularen Böden mittels Smoothed Particle Hydrodynamics.* Dissertation. ISBN: 978-3-936310-41-2.

40. Siegl, K. (2017). *Zur Pfahldynamik von gerammten Großbohrpfählen und der daraus resultierenden Wellenausbreitung in Wasser und im Meeresboden.* Dissertation. ISBN: 978-3-936310-42-9.

41. Grabe, J. (Hrsg.) (2017). *Conference Proceedings of Workshop on Numerical Methods in Geotechnics 2017.* Tagungsband. ISBN: 978-3-936310-43-6.

42. Grabe, J. (Hrsg.) (2018). *Digitale Infrastruktur und Geotechnik (DIG 2018).* Tagungsband. ISBN: 978-3-936310-44-3.

43. Osthoff, D. (2018). *Zur Ursache von Schlosssprengungen und zu einbringbedingten Lageabweichungen von Spundwänden*. Dissertation. ISBN: 978-3-936310-45-0.

44. Heins, E. (2018). *Numerical based identification of the pile-soil interaction in terms of the axial pile bearing capacity*. Dissertation. ISBN: 978-3-936310-46-7.

45. Seitz, K.-F. (2021). *Zur Topologieoptimierung von geotechnischen Strukturen und zur Tragfähigkeitssteigerung des Baugrunds durch Scherfugenverfestigung*. Dissertation. ISBN: 978-3-936310-47-4.

46. Plenker, D. (2021). *Physical and numerical investigations of the dynamic interaction of saturated granulates and fluid*. Dissertation. ISBN: 978-3-936310-48-1.

47. Grabe, J., J. O. Backhaus und P. Vogel (2021). *Bauprojektmanagement*. Skriptum. ISBN: 978-3-936310-49-8.

48. Kanitz, M. (2021). *Experimental and numerical investigations of particle-fluid systems in geotechnical engineering*. Dissertation. ISBN: 978-3-936310-50-4. DOI: 10.15480/882.3620

49. Backhaus, J. O. (2021). *A methodology for the numeric time-cost forecast and pareto optimization of large injection projects in tunneling*. Dissertation. ISBN: 978-3-936310-51-1. DOI: 10.15480/882.3840

50. Sinduri, S. N. (2021). *Optimisation of deep compaction as liquefaction mitigation measure*. Dissertation. ISBN: 978-3-936310-52-8. DOI: 10.15480/882.3945

51. Bubel, J. (2022). *Zum Versagen von Unterwasserböschungen im Seegang*. Dissertation. ISBN: 978-3-936310-53-5. DOI: 10.15480/882.4186

52. Grabe, J. (Hrsg.) (2022). *Adaptive Structures At Shore*. Tagungsband. ISBN: 978-3-936310-54-2.

53. Grabe, J. (Hrsg.) (2022). *Numerische Methoden in der Geotechnik*. Tagungsband. ISBN: 978-3-936310-55-9.

54. Grabe, J. (Hrsg.) (2022). *Processes in natural and technical Particle-Fluid-Systems (PintPFS)*. Tagungsband. ISBN: 978-3-936310-56-6.

55. Törzs, T. (2022). *Zum hydraulisch-mechanisch gekoppelten Verhalten teilgesättigter granularer Geomaterialien infolge hydraulischer und mechanischer Beanspruchungen*. Dissertation. ISBN: 978-3-936310-57-3. DOI: 10.15480/882.4611

56. Milatz, M. (2022). *Investigation of capillary effects on the grain scale by means of in situ experiments, imaging and numerical simulations*. Habilitation. ISBN: 978-3-936310-58-0. DOI: 10.15480/882.4803

57. Chmelnizkij, A. (2023). *Regularized MPM for porous media*. Dissertation. ISBN: 978-3-936310-59-7. DOI: 10.15480/882.4941.2.

58. Grabe, J. (Hrsg.) (2023). *Herausforderungen in der Geotechnik – 25 Jahre Institut für Geotechnik und Baubetrieb, Technische Universität Hamburg*. Tagungsband. ISBN: 978-3-936310-60-3.

59. Beusse, J. (2023). *Zur Einbringung und zum Tragverhalten von kombinierten Spundwänden von Ufereinfassungen*. Dissertation. ISBN: 978-3-936310-61-0. DOI: 10.15480/882.8634

Vorwort des Editors

Seit Jahren nutzen wir – und natürlich viel andere Wissenschaftler in der Geotechnik – numerische Methoden wie die Finite-Elemente-Methode. Das komplexe mechanische Verhalten von Boden wird über Stoffmodelle beschrieben. Es gibt neben einfachen eine Vielzahl höherwertiger Modelle, die in der Forschung ständig weiterentwickelt werden. Eines davon ist die Hypoplastizität. Beim Finite-Elemente-Programm Abaqus steht eine UMAT zur Verfügung, um das Stoffmodel in den Programmcode einzubinden. Nun sind diese UMAT-Routinen meist im Rahmen von Dissertationen entstanden mit der Schwäche, dass diese etwas handgestrickt sind und damit nicht von Programmversion zu Programmversion durchgängig bzw. kompatibel sind.

Die Arbeit von Herrn Zobel zielt darauf ab, Ordnung zu schaffen, indem eine klare und nachvollziehbare Containerstruktur implementiert wird. Er nennt diese neue Programmstruktur xMat. Auf diese Weise trennt er Stoffmodelle von Integrationsverfahren und definiert deren Schnittstellen. Da das in existierenden Codes nicht so klar getrennt ist, gibt es immer wieder Probleme durch unklare, sich gegebenenfalls widersprechende Angaben, was zu überraschenden Job-Abbrüchen und Ergebnissen führen kann. Durch die Containerisierung sind die Fehlermöglichkeiten deutlich kleiner. Um dorthin zu gelangen, musste Herr Zobel allerdings die vorhandenen Programmierungen nachvollziehen, im Detail verstehen, bereinigen und dann in die neue Struktur programmieren. Danach hat er die Programmierung durch Vergleichsrechnungen belegt.

Da die Arbeit mit der Entwicklung von Stoffmodellen und deren Anwendung in der Wissenschaft und Praxis immer mehr an Bedeutung gewonnen hat, ist die von Herrn Zobel vorgelegte Anamnese, die Herleitung und Programmierung einer xMat von hoher Relevanz für die Geotechnik. Wir planen uns vorliegende Stoffmodellroutinen in dieser Struktur zukünftig abzulegen und für numerische Modellierungen die xMat als Standard vorzugeben.

Die Motivation zu seiner Arbeit ist zu wesentlichen Teilen im Rahmen seiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Geotechnik und Baubetrieb entstanden. Er befasste sich zunächst mit der nachträglichen Verpfropfung von vibrierten Rohren im Rahmen eines vom BMWi geförderten Forschungsprojektes. Hier und auch bei einem nachfolgenden Projekt kämpfte er mit der Störanfälligkeit von Abaqus-Simulationen. Dies brachte ihn auf die Idee, die vorgenannte Zielsetzung mit dem Entwurf eines Konzeptes, der Erarbeitung einer klareren Containerstruktur, dessen Umsetzung und der Verifikation anhand von Vergleichssimulationen zu verfolgen. Das Ergebnis seiner intrinsischen Forschung liegt nun auf dem Tisch.

Die von Herrn Zobel vorgelegte Arbeit besticht durch die Problemanalyse, die daraus abgeleitete Zielvorgabe und dessen strukturierte Umsetzung. Zudem gefällt mir sehr, dass Herr Zobel in der Forschung eigene Wege gegangen ist. Die Thematik ist somit aus seinem eigenen Wunsch und Bedarf hervorgegangen. Es ist mitnichten das Abarbeiten einer im gewissen Maße vorgegebenen Forschungsfragestellung.

Herr Zobel hat im Bachelorstudium „Allgemeine Ingenieurwissenschaften“ und anschließend im Masterstudium „Theoretischer Maschinenbau“ studiert. Er musste sich daher erst in die Bodenmechanik einarbeiten. Aber gerade diese disziplinübergreifende Herangehens-

und Betrachtungsweise hat diese hervorragende Arbeit hervorgebracht. Es ist leicht zu erkennen, dass das Steckenpferd von Herrn Zobel die Informatik bzw. die Programmierung ist. Hier hat er eindeutig seine größten Stärken und Interessen.

Herr Zobel hat mit seiner strukturierten Programmierung einen wesentlichen Beitrag zum Stand der Wissenschaft und Technik in der numerischen Geotechnik geleistet.

Für seinen weiteren Lebensweg wünsche ich Herrn Zobel alles Gute.

Jürgen Grabe

Vorwort des Autors

In meinen sechs Jahren am Institut für Geotechnik und Baubetrieb konnte ich an vielen spannenden Arbeiten in diesem Fachbereich teilhaben: Tunnelbohrmaschinen, Schneckenförderern, Standsicherheit von Baumaschinen mit einem hohen Schwerpunkt und nachträgliche Tragfähigkeitserhöhung von Pfahlprofilen. All diese Themen sind unterschiedliche Perspektiven auf das zentrale Element in der Geotechnik: der Boden und sein Spannungs-Dehnungs-Verhalten. Und so hat sich fast von allein eine Faszination für Stoffmodelle entwickelt, die versuchen, das beobacht- und messbare Bodenverhalten in Form von mathematischen Gleichungen abzubilden. Da Stoffmodelle immer nur ausgewählte Charakteristiken von Bodenklassen beschreiben können, gibt es heutzutage viele verschiedene Stoffmodelle, die jeweils ihren Anwendungsbereich und verschiedene Vor- und Nachteile haben.

In dieser Doktorarbeit stelle ich die von mir entwickelte Containerstruktur xMat vor, mit der einmal implementierte Stoffmodelle in verschiedenen Konfigurationen bzw. von verschiedenen Programmen aus genutzt werden können. Das erlaubt in der Forschung eine einfache Anpassbarkeit an die eigenen Arbeiten und Anwender:innen eine große Flexibilität an die jeweils geforderte Aufgabe.

Natürlich baut die Arbeit auf zahlreiche Vorarbeiten auf, die im Hauptteil erwähnt werden. Ich bin dankbar für den Zugang die Fülle an Informationen, die mir einerseits von der Universität bereitgestellt worden sind und andererseits in der heutigen Zeit verfügbar sind.

Es gibt viele Menschen in meinem Umfeld, die direkt oder indirekt am Gelingen dieser Arbeit beigetragen haben. Deshalb möchte ich mich herzlich bei allen Unterstützer:innen bedanken, ohne die die folgende Arbeit so nicht möglich gewesen wäre, insbesondere

- meiner Familie, meinen Freund:innen und meiner Frau Stefanie, die meine Neugier gefördert und mich bisher auf meinem Weg durchs Leben unterstützt haben,
- den Kolleg:innen am Institut, mit denen ich nicht nur viele fachliche Diskussionen führen konnte, sondern auch neben der Arbeitszeit schöne Stunden verbracht habe,
- Jürgen Grabe für den Freiraum, den er mir in der Forschung gewährt hat, und Hans Stanford für die Begleitung auf dem Weg in die Hypoplastizität,
- Alexander Düster und Dimitrios Kolymbas für die Begutachtung der Arbeit und Robert Seifried für die Übernahme des Prüfungsvorsitzes,
- Göta Bürkner, Marek Banduch und Patrick Mayanja, die mir viele Laboruntersuchungen ermöglicht und mich bei meinen Messungen unterstützt haben und
- allen Entwickler:innen von *GNU/Linux*, *KDE* und den vielen freien Softwareprojekten wie bspw. *gcc*, *Inkscape*, *Python* (insb. *matplotlib*) und LaTeX, auf deren Grundlage diese Arbeit entstehen konnte.

Hamburg, Oktober 2023

Dominik Zobel

Schlagwörter:

Hypoplastizität, Barodesie, Stoffmodelle, Stoffroutine, Objektorientierte Implementierung, Elementversuche, Numerische Simulation, Finite-Elemente-Methode

Keywords:

Hypoplasticity, Barodesy, Constitutive Models, Constitutive Routine, Object-oriented Implementation, Element Tests, Numerical Simulations, Finite-Element-Method

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **5**
2.1 Notation und mathematische Grundlagen 6
2.2 Bodenverhalten und geotechnische Grundlagen 9
2.2.1 Spannungs-Dehnungs-Verhalten von Boden 11
2.2.2 Geotechnische Laborversuche zur Charakterisierung eines Bodens . 12
2.2.3 Genauigkeit der Messdaten 15
2.3 Stoffmodelle für Böden 17
2.3.1 Definitionen aus der Kontinuumsmechanik 18
2.3.2 Definitionen für Bodenmaterialien 21
2.3.3 Hypoplastizität 26
2.3.4 Barodesie 33
2.4 Integrationsverfahren 37
2.4.1 Explizite Runge-Kutta-Verfahren 38
2.4.2 Schrittweite und Fehlerabschätzung 40
2.4.3 Implizite Verfahren 41
2.4.4 Gleitkommazahlen und Rundungsfehler 41

3 Ausgangssituation **45**
3.1 Benutzerroutinen 46
3.1.1 Verwendete Benutzerroutinen 49
3.1.2 Weitere Benutzerroutinen und Schnittstellen 52
3.2 Überlegungen zur Implementierung von Benutzerroutinen 53

4 Implementierung der xMat **59**
4.1 Struktur der xMat 60
4.1.1 Schnittstellen und interne Abläufe 60
4.1.2 Modularisierung 62
4.2 Implementierung der einzelnen Module 65
4.2.1 Allgemeine Einstellungen (Modul `General_Settings`) 65
4.2.2 Stoffmodelle (Modul `Constitutive_Models`) 71
4.2.3 Integrationsverfahren (Modul `Solver`) 74
4.3 Anbindung an andere Programme 77
4.3.1 Schnittstellen zu Abaqus 78
4.3.2 Schnittstelle zu Tochnog 83
4.3.3 Fortran-Schnittstelle 85
4.3.4 Schnittstelle zu MATLAB 87

4.3.5 Schnittstelle zu Plaxis . . . 88
4.3.6 C-Schnittstelle . . . 91
4.4 Reflexionen . . . 95
4.4.1 Bereiche im Code mit Verbesserungspotential . . . 95
4.4.2 Abhängigkeit von den Einheiten . . . 96

5 Erweiterung der xMat 99
5.1 Entwicklungsumgebung der xMat . . . 100
5.2 Repräsentation von Matrizen und S4-Tensoren . . . 100
5.3 Hinzufügen weiterer Stoffmodelle . . . 104
5.4 Hinzufügen weiterer Integrationsverfahren . . . 110
5.5 Hinzufügen weiterer Schnittstellen . . . 111
5.6 Unit-Tests . . . 113

6 Untersuchte Materialien und deren Parameter 117
6.1 Hamburger Sand . . . 119
6.2 Glaskugeln . . . 120
6.3 Randporen und Membrandurchdringungseffekt . . . 121
6.4 Parametersätze aus der Literatur . . . 123
6.5 Optimierung der Materialparameter . . . 126

7 Überprüfungen 129
7.1 Teststrategie . . . 129
7.2 Untersuchung der Stoffmodelle . . . 132
7.2.1 Eigenschaften der Stoffmodelle in ausgewählten Berechnungen . . . 132
7.2.2 Vergleiche in der Triaxialebene . . . 139
7.2.3 Weitere Spannungszustände . . . 147
7.3 Untersuchung der Schnittstellen . . . 156
7.3.1 Überprüfung der Funktionalität . . . 156
7.3.2 Einfache FE-Simulationen . . . 158
7.4 Untersuchung der Integrationsverfahren . . . 160

8 Anwendungsbeispiel 165
8.1 Planung der experimentellen Versuche . . . 165
8.2 Auswertung der Modellversuche . . . 169
8.3 Idealisierte Verteilungsformen . . . 176
8.4 Planung der numerischen Simulationen . . . 178
8.5 Auswertung der numerischen Simulationen . . . 180

9 Zusammenfassung und Ausblick 185

Literatur 187

Häufig verwendete Symbole 201

Anhang **203**

A Mögliche Visualisierung von S4-Tensoren und Rechenoperationen **205**

B Sensitivitätsstudie **213**

C Explizite Runge-Kutta-Methoden **221**

D Dokumentation der xMat-Funktionalitäten **229**

E Materialparameter und Statusvariablen der Benutzerroutinen **243**

F Beispiele zur Nutzung der xMat **253**

G Zur Bestimmung der Unsicherheit von Laborversuchen **281**

H Eigene Abschätzungen **285**

I Daten der Laborversuche **289**

J xMat Referenz **297**

Index **303**

1 Einleitung

The secret to doing good research is always to be a little underemployed.
You waste years by not being able to waste hours.

Amos Tversky (zugeschrieben in Lewis, 2017)

Hintergrund

Es gibt viele Herausforderungen beim Versuch, das Spannungs-Dehnungs-Verhalten von Böden realitätsnah in einem Modell abzubilden. Bereits der Begriff Boden ist sehr allgemein und beschreibt eine große Bandbreite an möglichen Materialien, Zuständen (Sättigung, Vorverdichtung, ...) und die Zusammensetzung ebenjener. Für die Entwicklung von Stoffmodellen ist es daher zweckmäßig, sich auf einzelne Klassen von Böden zu konzentrieren, die ein ähnliches Verhalten aufweisen. So gibt es heutzutage viele verschiedene Stoffmodelle, die jeweils ihren Anwendungsbereich und verschiedene Vor- und Nachteile haben.

In Kolymbas und Herle (2008), S. 245, wird zur Wahl des passenden Stoffmodells darauf hingewiesen, dass es „die Effekte, auf die es jeweils ankommt, möglichst treffend beschreiben [sollte]". Ergänzend weisen Benz und Bergholz (2017), S. 73, darauf hin, dass „[d]ie Realitätsnähe der Modellantwort im Verhältnis zur Komplexität des Rechenmodells [...] ein wichtiges Maß für die einem Modell aufseiten der Anwender potenziell entgegengebrachte Akzeptanz [ist]." Für den praktischen Anwender stellt sich außerdem die Frage, ob ein Programm zur Verfügung steht, dass das gewünschte Stoffmodell unterstützt oder eine Bereitstellung in Form einer Benutzerroutine möglich ist.

Gleichzeitig liegen einige Stoffmodelle in unterschiedlichen Implementierungen als Benutzerroutine vor, bspw. durch verschiedene Autoren oder nachträgliche Ergänzungen und Verbesserungen. Hier stellt sich die Frage, welche Implementierung für die eigenen Untersuchungen gewählt werden sollte. Oder welche Benutzerroutine angepasst werden sollte, wenn keine alle erforderlichen Merkmale enthält.

Im Rahmen dieser Arbeit ist eine neue Benutzerroutine entwickelt worden, die als Basis für die Arbeit mit Stoffmodellen und die Weiterentwicklung ebenjener genutzt werden kann. Sie baut auf den Erfahrungen anderer Benutzerroutinen auf, ist flexibel nutzbar und erlaubt eine einfache Anpassbarkeit. Während der Entwicklung sind die folgenden Anforderungen an die Benutzerroutine gestellt worden:

- Ein bereits implementiertes Stoffmodell soll über verschiedene Schnittstellen aus verschiedenen Programmen nutzbar sein. Es ist nicht zielführend, wenn für verschiedene Programme jeweils eigene Benutzerroutinen mit dem gleichen Stoffmodell bereitgestellt und gepflegt werden müssen.

- Je nach Situation kann der Einsatz unterschiedlicher Stoffmodelle sinnvoll sein. Schnittstellen und Stoffmodelle müssen entkoppelt sein, um alle implementierten Stoffmodelle gleichwertig über eine beliebige Schnittstelle nutzen zu können.
- Darüber hinaus sollen von Anfang an Stoffmodelle und Schnittstellen zu mehreren Programmen und Programmiersprachen verfügbar sein, die von Ingenieuren für die Untersuchung des Bodenverhaltens verwendet werden.
- Das Integrationsverfahren ratenabhängiger Stoffmodelle soll frei wählbar sein.
- Durch die Kapselung in verschiedene Komponenten soll die Modifikation von Stoffmodellen, Integrationsverfahren oder Schnittstellen ohne nennenswerte Anpassungen in den anderen Bereichen möglich sein.
- Um eine möglichst einfache und intuitive Nutzung zu ermöglichen, soll der Fokus nicht allein auf die Effizienz der bereitgestellten Komponenten gelegt werden, sondern auch darauf, dass ein gut lesbarer und verständlicher Code entsteht.

Übersicht

Im Rahmen dieser Arbeit ist eine Containerstruktur namens *xMat*[1] entwickelt worden, die mithilfe objektorientierter Programmierung Stoffmodelle, Integrationsverfahren und Schnittstellen an andere Programme modular in sich vereint. Dadurch können einmal implementierte Stoffmodelle in verschiedenen Konfigurationen bzw. von verschiedenen Programmen aus genutzt werden, was dem Forscher eine einfache Anpassbarkeit und dem Anwender eine große Flexibilität an die jeweils geforderte Aufgabe bietet.

So baut die vorliegende Arbeit in gewisser Weise auf der Dissertation von Mittendorfer (2010) auf, in der eine modulare Implementierung einer Benutzerroutine beschrieben wird (v. a. S. 57 und S. 61 f.). Dort wird eine Trennung in die einzelnen Bereiche FE-Programm, Integrator und Stoffmodell vorgenommen, wobei die letzten beiden Bereiche in einer Benutzerroutine integriert werden (siehe Bild 1.1). Einerseits wird dort die Hypoplastizität nach von Wolffersdorff (1996) als Stoffmodell bereitgestellt, optional mit dem Konzept der intergranularen Dehnung nach Niemunis und Herle (1997). Andererseits sind zwei Integrationsverfahren verfügbar, ein explizites Runge-Kutta-Verfahren zweiter Ordnung und das semi-implizite Euler-Verfahren, jeweils mit adaptiver Schrittweiten-Kontrolle (siehe auch Fellin, Mittendorfer u. a., 2009). Außerdem wird die Anbindung zu Abaqus/Standard, MATLAB und Fortran aufgezeigt.

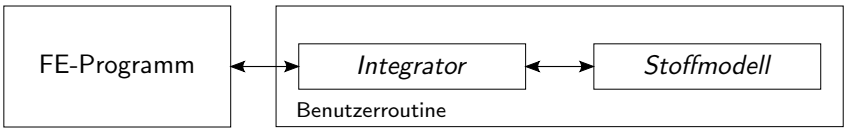


Bild 1.1: Modulare Struktur einer Benutzerroutine nach Mittendorfer (2010), S. 61.

[1] xMat ©D. Zobel: »https://github.com/d-zo/xMat
Der Name xMat setzt sich in Anspielung auf UMAT und VUMAT von Abaqus aus dem Platzhalter x und *Mat*erial zusammen, also beliebige Stoffmodelle und Schnittstellen in Form einer einzigen Benutzerroutine.

Containerstruktur xMat

Die modulare Struktur der xMat ermöglicht nicht nur eine flexible Nutzung der verfügbaren Komponenten, sondern erlaubt auch ein einfaches Hinzufügen oder Entfernen von weiteren Stoffmodellen, Integrationsverfahren oder Schnittstellen zu anderen Programmen. Deshalb sind bereits im Rahmen dieser Arbeit die folgenden Komponenten mit einstellbaren Parametern in die xMat integriert worden:

- Die ratenabhängigen, hypoplastischen Stoffmodelle[2]
  - `Hypo-Wu92`: Hypoplastizität nach Wu (1992),
  - `Hypo-VW96`: Hypoplastizität nach von Wolffersdorff (1996) mit dem optional zuschaltbaren Konzept der intergranularen Dehnung nach Niemunis und Herle (1997),
  - `Baro-Ko15`: Barodesie nach Kolymbas (2015),
  - `Baro-Sc18`: Barodesie nach Schranz (2018) und
  - `Baro-Ko21`: Barodesie nach Kolymbas (2021).
- Die expliziten Integrationsverfahren mit adaptiver Schrittweiten-Kontrolle
  - `Eul-Exp`: Euler-Explizit ohne Schrittweiten-Kontrolle,
  - `Richard`: Euler-Explizit mit Richardson-Extrapolation,
  - `RK-S 23`: Simpsonregel (Runge-Kutta 2/3),
  - `RK-BS23`: Bogacki-Shampine (Runge-Kutta 2/3),
  - `RK-F 45`: Runge-Kutta-Fehlberg (Runge-Kutta 4/5),
  - `RK-CK45`: Cash-Karp (Runge-Kutta 4/5) und
  - `RK-DP45`: Dormand-Prince (Runge-Kutta 4/5).
- Die Programmschnittstellen mit Integration
  - `UMAT()`: Abaqus/Standard[3],
  - `VUMAT()`: Abaqus/Explizit,
  - `mexFunction()`: MATLAB[4],
  - `USER_MOD()`: Plaxis[5],
  - `xmat_console()`: Fortran und
  - `xmat_console_CInter()`: C und darüber GNU Octave[6] und Python.

[2] Zusätzlich enthält die xMat noch ein ideal-elastisches Modell `Elas-0000` sowie die Viskohypoplastizität `ViHy-Ni03` nach Niemunis (2003), die aber in dieser Arbeit nicht näher betrachtet werden.
[3] Abaqus ©Dássault Systemès: »https://www.3ds.com/products-services/simulia/products/abaqus/
[4] MATLAB ©The MathWorks, Inc.: »https://www.mathworks.com/products/matlab.html
[5] Plaxis ©Bentley Systems, Inc.: »https://www.bentley.com/software/plaxis-3d/
[6] GNU Octave ©J. Eaton u. a.: »https://octave.org/

- Die Programmschnittstellen ohne Integration
  - `Dot_Values()`: Fortran und
  - `Dot_Values_CInter()`: C-Schnittstelle und u. a. Simulink[7].

Neben der Implementierung wird auch die Funktionstüchtigkeit und Korrektheit der Komponenten überprüft. Für die verschiedenen Stoffmodelle werden Parametersätze aus der Literatur und aus eigenen Laborversuchen genutzt und mit Referenzdaten und den Ergebnissen anderer Benutzerroutinen aus Tabelle 3.1 auf S. 48 verglichen.

Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen für die restliche Arbeit zusammengetragen. Die zentralen Konzepte aus Boden- und Kontinuumsmechanik finden sich in Abschnitt 2.2, Abschnitt 2.3 legt den Fokus auf die Stoffmodelle selbst und in Abschnitt 2.4 werden die Integrationsverfahren beleuchtet. Anschließend werden in Kapitel 3 die betrachteten Benutzerroutinen vorgestellt, die einen maßgeblichen Einfluss auf die Entwicklung der xMat hatten und zum späteren Vergleich genutzt werden.

Kapitel 4 konzentriert sich auf die Implementierung der xMat und zeigt auf, welche Struktur gewählt worden ist (Abschnitt 4.1) und wie die einzelnen Module Stoffmodelle, Integrationsverfahren und Schnittstellen umgesetzt worden sind (Abschnitt 4.2). Abschnitt 4.3 dokumentiert die Schnittstellen.

Aufbauend darauf werden in Kapitel 5 Anpassungs- und Erweiterungsmöglichkeiten der xMat-Containerstruktur aufgezeigt, wie die Anpassung der internen Repräsentation von Tensoren in Abschnitt 5.2 und das Hinzufügen weiterer Stoffmodelle, Integrationsverfahren oder Schnittstellen in Abschnitten 5.3 bis 5.5.

In Kapitel 6 werden die Materialien und Materialparameter vorgestellt, die in experimentellen und numerischen Untersuchungen zum Einsatz kommen. Zusammen mit weiteren Parametersätzen aus der Literatur werden in Kapitel 7 Nachrechnungen durchgeführt und mit Referenzlösungen verglichen, um die Komponenten und Funktionsweise der xMat zu testen. Der Hauptfokus liegt auf der Überprüfung der implementierten Stoffmodelle in Abschnitt 7.2. Die Schnittstellen und Integrationsverfahren werden in den Abschnitten 7.3 und 7.4 untersucht.

In Kapitel 8 wird als Anwendungsbeispiel ein Experiment im Erddruckversuchsstand aufgezeigt. Hier kommt die xMat in den numerischen Simulationen mit Abaqus/Explizit zum Einsatz.

Die Arbeit wird durch einen umfangreichen Anhang mit weiteren Ergebnissen, Skripten und einer Dokumentation der xMat ergänzt.

[7] Simulink ©The MathWorks, Inc.: »https://www.mathworks.com/products/simulink.html

2 Grundlagen

In diesem Kapitel werden die Grundlagen beschrieben, die für die Diskussion der Ergebnisse in den nächsten Kapiteln relevant sind. Nach der Notation von Formelzeichen und den mathematischen Grundlagen in Abschnitt 2.1 folgen die geotechnischen Grundlagen zum Spannungs-Dehnungs-Verhalten von Boden in Abschnitt 2.2. Abschnitt 2.3 vertieft, wie durch phänomenologische Stoffmodelle einzelne, idealisierte Eigenschaften des Bodens beschrieben werden können. In Abschnitt 2.4 werden Grundlagen zur Integration von Differentialgleichungen (wie ratenabhängigen Stoffmodelle) vorgestellt.

Wichtige Erkenntnisse aus diesem Kapitel sind:

Bodenmechanik

- Elementversuche stellen immer einen idealisierten Zustand dar, der mit realen Messungen nur angenähert werden kann.
- Zur Bewertung von Messergebnissen muss die Messunsicherheit berücksichtigt werden.

Stoffmodelle

- Es gibt kein allgemein gültiges Stoffmodell für Boden.
- Die Eignung eines Stoffmodells hängt maßgeblich von den Rahmenbedingungen ab, bspw. Komplexität der Aufgabenstellung, Berechnungsgebiet, gesuchte Zielgrößen, Aufwand bei Vorbereitung (Parameterbestimmung) und Durchführung (Rechenzeit).

Integrationsverfahren

- Wenn ein Verfahren mit höherer Ordnung zum Einsatz kommt, bedeutet das nicht, dass die damit berechnete Lösung eine höhere Genauigkeit hat.
- Durch die Abschätzung des Integrationsfehlers ist eine automatische Schrittweiten-Kontrolle bei der Integration möglich.

2.1 Notation und mathematische Grundlagen

Symbole

Skalare Größen werden kursiv dargestellt (c, φ). Vektoren werden fettgedruckt und kursiv symbolisiert ($\boldsymbol{u}, \boldsymbol{v}$) und sind im Allgemeinen als Spaltenvektoren zu sehen. Matrizen werden als fettgedruckte Großbuchstaben dargestellt ($\mathbf{D}, \boldsymbol{\sigma}$). *Tensoren* vierter Stufe werden als kalligrafische Großbuchstaben gesetzt ($\mathcal{J}, \mathcal{L}$).

Annahmen

- Bei allen nicht-skalaren Größen wird von einer Standardbasis ausgegangen, d. h. kartesische Koordinaten im jeweiligen euklidischen Vektorraum. Damit stimmen Tensoren erster Ordnung mit ihrer Vektorrepräsentation und Tensoren zweiter Ordnung mit ihrer Matrixrepräsentation überein.
- Tensoren vierter Ordnung mit Standardbasis werden in dieser Arbeit als *S4-Tensoren* bezeichnet.
- Sofern nicht anders vermerkt, wird immer von reellen Größen im dreidimensionalen Raum ausgegangen, sodass folgende Zusammenhänge gelten: Vektoren $\boldsymbol{v} \in \mathbb{R}^3$, Matrizen $\mathbf{M} \in \mathbb{R}^{3\times3}$ und S4-Tensoren $\mathcal{T} \in \mathbb{R}^{3\times3\times3\times3}$.

Darstellung von Rechenoperationen

Bei Multiplikationen wird der Operator ($\cdot$) i. d. R. nur notiert, wenn dadurch die Verständlichkeit erhöht wird.[1] Multiplikationen (bzw. verallgemeinert Summen über Indizes) können nach der Einsteinschen Summationskonvention in Indexschreibweise dargestellt werden.[2] Die Summation über zwei Indizes (:) entspricht einer *doppelten Kontraktion*.[3] Das *dyadische Produkt* ($\otimes$) erweitert den Rang eines Tensors (siehe auch Golub und Van Loan, 2013, S. 724 f.). Beispielsweise gilt

$$\begin{aligned}
\mathbf{AB} = \mathbf{D} \quad &\Leftrightarrow \quad A_{ij}B_{jk} = D_{ik} \quad , \qquad \mathbf{M}\boldsymbol{u} = \boldsymbol{v} \quad \Leftrightarrow \quad M_{ij}u_j = v_i \quad , \\
\mathcal{J} : \mathbf{M} = \mathbf{K} \quad &\Leftrightarrow \quad J_{ijkl}M_{kl} = K_{ij} \quad , \qquad \mathbf{A} : \mathbf{B} = c \quad \Leftrightarrow \quad A_{ij}B_{ij} = c \quad , \\
\boldsymbol{u} \otimes \boldsymbol{v} = \mathbf{N} \quad &\Leftrightarrow \quad u_i v_j = N_{ij} \quad \text{und} \quad \mathbf{A} \otimes \mathbf{B} = \mathcal{K} \quad \Leftrightarrow \quad A_{ij}B_{kl} = K_{ijkl} \quad .
\end{aligned}$$

[1] Beispielsweise wird das Multiplikationszeichen beim Produkt einer Variable mit einer Klammer gesetzt, um die Auswertung einer Funktion $f(a + b)$ von der Multiplikation mit der Variablen $f \cdot (a + b)$ zu unterscheiden.

[2] A. Einstein führt die Indexschreibweise 1916 in seiner Abhandlung „Die Grundlage der Allgemeinen Relativitätstheorie“ (S. 781–786) zur Vereinfachung der Notation ein. Typischerweise wird dabei durch die Position der Indizes zwischen Kovarianz und Kontravarianz unterschieden, aber für den euklidischen Vektorraum mit einem kartesischen Koordinatensystem sind beide identisch. Deshalb werden nur kovariante (tiefer gestellte) Indizes verwendet.

[3] Ein visuelles Beispiel zur doppelten Kontraktion eines S4-Tensors findet sich in Anhang A.

Der natürliche *Logarithmus* wird als $\ln(c) = \log_e(c)$ angegeben und der dekadische Logarithmus als $\log_{10}(c)$.

Funktionen und Definitionen

In der restlichen Arbeit werden u. a. die folgenden Größen verwendet:

- δ_{ij} ist das *Kronecker-Delta*, das folgendermaßen definiert ist:

$$\delta_{ij} = \begin{cases} 1 & \text{falls} \quad i = j \\ 0 & \text{sonst} \end{cases} ,$$

- Damit kann die Einheitsmatrix definiert werden als $\mathbf{I}$ mit $I_{ij} = \delta_{ij}$.
- Die *Spur* einer $n \times n$-Matrix ist definiert als Summe der Diagonalelemente: $\mathrm{tr}(\mathbf{A}) = A_{ij}\delta_{ij}$. Insbesondere gilt für zwei $n \times n$-Matrizen: $\mathrm{tr}\left(\mathbf{A}\mathbf{B}^T\right) = \mathbf{A} : \mathbf{B} = A_{ij}B_{ij}$.
- Der *Deviator* einer Matrix $\mathbf{A}^{\mathrm{dev}} \in \mathbb{R}^{n\times n}$ ist $\mathbf{A}^{\mathrm{dev}} = \mathrm{dev}(\mathbf{A}) = \mathbf{A} - \frac{\mathrm{tr}(\mathbf{A})}{n}\mathbf{I}$.
- Die *dimensionslose* Variante einer $n \times n$-Matrix ist definiert als $\hat{\mathbf{A}} = \frac{\mathbf{A}}{\mathrm{tr}(\mathbf{A})}$.
- Die *Determinante* einer Matrix $\mathbf{A} \in \mathbb{R}^{3\times 3}$ ist $\det(\mathbf{A}) = A_{11}\left(A_{22}A_{33} - A_{23}A_{32}\right) + A_{12}\left(A_{23}A_{31} - A_{21}A_{33}\right) + A_{13}\left(A_{21}A_{32} - A_{22}A_{31}\right)$.
- Für Vektoren wird die *euklidische Norm* (2-Norm) $||\boldsymbol{v}|| = \sqrt{\sum_i v_i^2}$ und analog dazu für Matrizen die *Frobeniusnorm* $||\mathbf{A}|| = \sqrt{\mathrm{tr}\left(\mathbf{A}\mathbf{A}^T\right)} = \sqrt{A_{ij}A_{ij}}$ verwendet.
- Die *Richtung* ist definiert als $\vec{\mathbf{A}} = \frac{\mathbf{A}}{||\mathbf{A}||}$.
- Eine *symmetrische* Matrix hat die Eigenschaft $\mathbf{A} = \mathbf{A}^T$ und somit $A_{ij} = A_{ji}$. Bei Tensoren vierter Stufe sind mehrere Symmetrien möglich, wobei $\mathcal{T} = \mathcal{T}^T$ die Bedingung $T_{ijkl} = T_{klij}$ („*major symmetry*“) bezeichnet. Eine weitere Symmetrie ist $T_{ijkl} = T_{jikl} = T_{ijlk} = T_{jilk}$ („*minor symmetry*“).[4] Liegen beide Symmetrieformen vor, wird der Tensor als total symmetrisch bezeichnet („*total symmetry*“ oder „*supersymmetric*“) und hat höchstens 21 unabhängige Einträge.[5] Die doppelte Kontraktion eines solchen Tensors $\mathcal{T}_S$ mit einer beliebigen Matrix $\mathbf{A}$ erzeugt immer eine symmetrische Matrix, d. h. $\mathcal{T}_S : \mathbf{A} = \mathbf{B}$ impliziert $\mathbf{B} = \mathbf{B}^T$ (siehe Itskov, 2019, S. 130).[6]
- Eine *reguläre* Matrix $\mathbf{A}$ ist quadratisch und hat eine Inverse $\mathbf{A}^{-1}$, mit der $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$ erfüllt ist.
- Eine *diagonalisierbare* Matrix $\mathbf{A}$ ist quadratisch und kann durch eine Ähnlichkeitstransformation mit einer regulären Matrix $\mathbf{Q}$ in eine Diagonalmatrix $\mathbf{D}$ transformiert werden, also $\mathbf{Q}\mathbf{A}\mathbf{Q}^{-1} = \mathbf{D}$.

[4] Zu Symmetrien an Tensoren siehe auch Tabelle A.1 in Anhang A.

[5] Backus (1970), S. 641, zeigt allgemein, dass ein Tensor $\mathcal{T} \in \mathbb{R}^{n\times n\times n\times n}$ mit totaler Symmetrie nur $\left[n\left(n+1\right)\left(n^2+n+2\right)\right]/8$ unabhängige Einträge hat.

[6] Genau genommen muss nur ein Teil der *minor symmetry* ($T_{ijkl} = T_{jikl}$) vorliegen, damit die doppelte Kontraktion mit einer beliebigen Matrix eine symmetrische Matrix erzeugt.

- Zusammen mit dem dyadischen Produkt können drei Tensoren vierter Stufe definiert werden, die für diese Arbeit von Interesse sind. Eine visuelle Darstellungshilfe der jeweiligen S4-Tensoren findet sich am Ende dieser Arbeit in Anhang A. $\mathcal{I}^{\mathrm{id}} : \mathbf{A}$ erzeugt die identische Abbildung einer Matrix $\mathbf{A}$, $\mathcal{I}^{\mathrm{sym}} : \mathbf{A}$ bildet den symmetrischen Teil der Matrix ab und $\mathcal{I}^{\mathrm{tr}} : \mathbf{A}$ erzeugt eine Diagonalmatrix mit der Spur der Matrix in den Diagonalelementen:[7]

$$
\begin{aligned}
I^{\mathrm{id}}_{ijkl} &= \delta_{ik} \otimes \delta_{jl} && \Leftrightarrow && \mathcal{I}^{\mathrm{id}} : \mathbf{A} = \mathbf{A} \quad , \\
I^{\mathrm{sym}}_{ijkl} &= \frac{1}{2} \left(\delta_{ik} \otimes \delta_{jl} + \delta_{il} \otimes \delta_{jk} \right) && \Leftrightarrow && \mathcal{I}^{\mathrm{sym}} : \mathbf{A} = \frac{1}{2} \left(\mathbf{A} + \mathbf{A}^T \right) = \mathrm{sym}(\mathbf{A}) \quad \text{und} \\
I^{\mathrm{tr}}_{ijkl} &= \delta_{ij} \otimes \delta_{kl} && \Leftrightarrow && \mathcal{I}^{\mathrm{tr}} : \mathbf{A} = \mathbf{B} \quad \text{mit} \quad B_{ij} = \mathrm{tr}(\mathbf{A}) \cdot \delta_{ij} \quad .
\end{aligned}
$$

- Isotrope Tensoren können in der Form $T_{ijkl} = \alpha \delta_{ij} \otimes \delta_{kl} + \beta \delta_{ik} \otimes \delta_{jl} + \gamma \delta_{il} \otimes \delta_{jk}$ mit drei beliebigen Skalaren α, β und γ dargestellt werden (siehe Hodge Jr., 1961). Der Elastizitätstensor (siehe Abschnitt 2.3.2) ist ein spezieller isotroper Tensor und kann bspw. als $\mathcal{T} = r\mathcal{I}^{\mathrm{sym}} + \frac{s}{3}\mathcal{I}^{\mathrm{tr}}$ mit zwei beliebigen (reellen) Skalaren r und s dargestellt werden (siehe Itskov, 2019, S. 134).
- Eine Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ hat n *Eigenwerte*. Die Eigenwerte λ erfüllen die Bedingung $\mathbf{A}\boldsymbol{v} = \lambda \boldsymbol{v}$ mit einem *Eigenvektor* $\boldsymbol{v} \neq 0$ und somit $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$. Die Eigenwerte (und Invarianten) von zwei regulären, quadratischen Matrizen $\mathbf{A}$ und $\mathbf{B}$ sind identisch, wenn mit einer regulären Matrix $\mathbf{Q}$ der Zusammenhang $\mathbf{B} = \mathbf{Q}^{-1}\mathbf{A}\mathbf{Q}$ gilt (Ähnlichkeitstransformation, siehe auch Golub und Van Loan, 2013, S. 349).
- Die *Invarianten einer quadratischen Matrix* $\mathbf{A}$ sind die Nullstellen des charakteristischen Polynoms $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$ (Satz von Cayley-Hamilton). Für $\mathbf{A} \in \mathbb{R}^3$ ergibt sich $\mathbf{A}^3 - I_1 \cdot \mathbf{A}^2 + I_2 \cdot \mathbf{A} - I_3 \cdot \mathbf{I} = \mathbf{0}$ mit den drei Invarianten

$$
\begin{aligned}
I_1 &= \mathrm{tr}(\mathbf{A}) && = \lambda_1 + \lambda_2 + \lambda_3 \quad , \\
I_2 &= \frac{1}{2} \left(\mathrm{tr}(\mathbf{A})^{\,2} - \mathrm{tr}\left(\mathbf{A}^2\right) \right) && = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_1\lambda_3 \quad \text{und} \\
I_3 &= \det(\mathbf{A}) && = \lambda_1\lambda_2\lambda_3 \quad .
\end{aligned}
$$

- Die *Matrixexponentialfunktion* $e^{\mathbf{M}}$ ist ähnlich zur normalen Exponentialfunktion definiert als $e^{\mathbf{M}} = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{M}^k$ mit $\mathbf{M}^0 = \mathbf{I}$. Mit der Bedingung $e^{\mathbf{Q}\mathbf{R}\mathbf{Q}^{-1}} = \mathbf{Q}e^{\mathbf{R}}\mathbf{Q}^{-1}$ für invertierbare $\mathbf{Q}$ (siehe auch Golub und Van Loan, 2013, S. 517) kann die Matrixexponentialfunktion für eine diagonalisierbare Matrix $\mathbf{M}$ umgeschrieben werden zu

$$
e^{\mathbf{M}} = e^{\mathbf{V}\mathbf{D}\mathbf{V}^{-1}} = \mathbf{V}e^{\mathbf{D}}\mathbf{V}^{-1} = \mathbf{V} \begin{pmatrix} e^{\lambda_1} & & \\ & \ddots & \\ & & e^{\lambda_n} \end{pmatrix} \mathbf{V}^{-1}
$$

mit der Diagonalmatrix der Eigenwerte $\mathbf{D} = \lambda_i \delta_{ij}$ und der Matrix der (rechten) Eigenvektoren $\mathbf{V}$.

[7] Itskov (2019), S. 132–134, gibt eine ausführliche Beschreibung zu diesen und weiteren Tensoren.

Statistische Parameter

Aus n-fach erhobenen Datensätzen $\boldsymbol{x}$ werden Referenzdatensätze $\boldsymbol{x}_{\text{ref}}$ mit Mittelwert und Unsicherheit erzeugt. Dabei sind *Mittelwert* μ und die (empirische) *Standardabweichung* σ wie folgt definiert (siehe DIN 1319-3:1996):

$$\mu = \frac{\sum_{i=1}^{n} X_i}{n} \quad \text{und} \qquad \sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \mu)^2} \quad .$$

Zusätzlich sind die *Standardunsicherheit* u und die Fortpflanzung der Unsicherheit u_f mit Modellfunktion $f(\boldsymbol{x})$ und den (unkorrelierten) Eingangswerten x_i definiert als

$$u = \frac{\sigma}{\sqrt{n}} \quad \text{und} \qquad u_f = \sqrt{\sum_{i=1}^{n}\left(\frac{\partial f}{\partial x_i}\right)^2 u_i^2} \quad .$$

Die meisten Messwerte werden in dieser Arbeit in der Form $23{,}478\,(\sigma = 0{,}032)$ m angegeben, wobei in diesem Beispiel $\mu = 23{,}478$ m die mittlere Länge und $u = 0{,}032$ m die dazugehörige Unsicherheit ist. Wie in Möhrke und Runge (2020), S. 47, empfohlen, werden Messwerte in dieser Arbeit auf zwei signifikante Stellen angegeben. Die *Unsicherheit* entspricht entweder der Standardabweichung oder der Standardunsicherheit durch die Erhebung der Daten, falls diese größer als die Standardabweichung der Messwerte ist (siehe Abschnitt 2.2.3).

2.2 Bodenverhalten und geotechnische Grundlagen

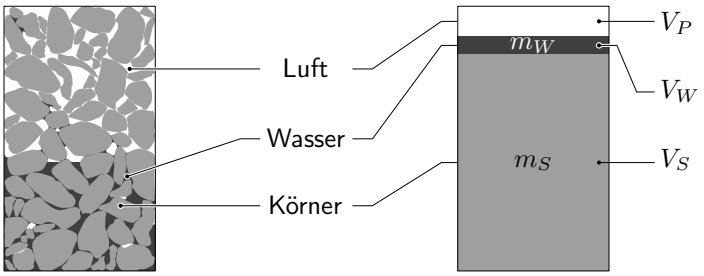
Einiges ist in der Geotechnik – wie zu Beginn des Bauens –
geprägt von Beobachtung, Probieren und Intuition.

Grabe (2004), S. 16

Mit Boden wird eine Mischung verschiedener Gesteinspartikel bezeichnet, die durch Verwitterung von Fels entstanden sind (Helwany, 2007, S. 2). Die einzelnen Partikel oder Körner haben im Allgemeinen unterschiedliche Formen und Größen und können aus verschiedenen Materialien bzw. Materialzusammensetzungen bestehen. Die Charakterisierung eines Bodens wird über die hauptsächlich vorherrschende Partikel- oder Korngröße zu Kies, Sand oder Schluff/Ton vorgenommen.

Das Volumen zwischen den einzelnen Körnern ist der *Porenraum*, der mit Luft, Wasser oder einem anderen Fluid gefüllt sein kann (siehe Bild 2.1). Für einen Bodenkörper mit Volumen V und einer Masse m ergibt sich somit für die Körner ein Feststoffvolumen V_S („*solid*“) sowie eine Trockenmasse von m_S und für den Porenraum ein Volumen $V_P = V - V_S$.[8] Daraus kann die *Porenzahl* $e = V_P/V_S$, d. h. das Verhältnis des Porenraumvolumens zum Feststoffvolumen, bestimmt werden. Der *Porenanteil* $n = V_P/V$ beschreibt den Anteil des Porenraumvolumens am Gesamtvolumen.

[8] Für eine trockene Bodenprobe (Porenräume mit Luft gefüllt) gilt $m = m_S$ und für eine vollständig gesättigte Bodenprobe $m = m_S + m_W$.



$$V = V_S + V_W + V_P \quad ,$$

$$\rho = \frac{m}{V} \quad ,$$

$$\rho_d = \frac{m_S}{V} \quad \text{und}$$

$$\rho_S = \frac{m_S}{V_S} \quad .$$

Bild 2.1: Links: Beispielhafte Bodenzusammensetzung mit den drei Phasen Körner, Wasser und Luft. Wie angedeutet sind Luftblasen und Wassereinschlüsse zwischen Körnern möglich. Mitte: Idealisierte Aufteilung in einzelne Phasen mit dem Gesamtvolumen V (siehe Grundlagenbücher wie bspw. Helwany, 2007, S. 3). Rechts: Definition von Dichte/Feuchtdichte ρ, Trockendichte ρ_d und Korndichte ρ_S.

Während Trockenmasse m_S, Feststoffvolumen V_S und Korndichte $\rho_S = m_S/V_S$ einer Bodenprobe als konstant angenommen werden, kann sich das Porenraumvolumen V_P abhängig von der Anordnung der Körner (Konfiguration) ändern. Die Konfiguration mit dem kleinsten Porenraumvolumen und der *dichtesten Lagerung* entspricht somit der kleinsten Porenzahl $e_{\min} = V_{P,\min}/V_S$. Analog ergibt sich die *lockerste Lagerung* aus dem größten Porenraumvolumen mit der größten Porenzahl $e_{\max} = V_{P,\max}/V_S$. Bei Bodenproben ist die tatsächliche Lagerung abhängig von Faktoren wie Bodenzustand und Lastgeschichte.

In dieser Arbeit wird der Fokus auf Sande und andere *nichtbindige*[9] Böden gelegt, die trocken oder vollständig wassergesättigt und drainiert angenommen werden. Die verwendeten Sande und eine als Feinkies eingestufte Kugelmischung werden in Abschnitt 6 vorgestellt. Nichtbindige Böden zeigen u. a. folgende Eigenschaften (siehe auch Kolymbas, 2011b, S. 13; und Hügel, 1995, S. 10 f.):

- Der Bodenkörper besteht aus einer Vielzahl unterschiedlicher Körner, die einzeln ein anderes Verhalten zeigen als der gesamte Bodenkörper.
- Bei einer Änderung der Belastung sind im Allgemeinen Kornumlagerungen möglich. Vor allem bei Scherbelastungen sind – je nach Art und Lagerungsdichte des Bodenkörpers – zwei unterschiedliche Verhaltensweisen möglich: Typischerweise erfolgt mit zunehmender Druckbelastung bei locker gelagerten Bodenkörpern eine Volumenverkleinerung (*kontraktantes* Verhalten) und bei dicht gelagerten Bodenkörpern eine Volumenvergrößerung (*dilatantes* Verhalten).
- Verformungen basieren ausschließlich auf Kornumlagerungen und sind im Allgemeinen weder elastisch noch reversibel.
- Proportionale Spannungspfade mit spannungslosem Ausgangszustand werden allein durch die Verzerrung definiert. Alle anderen Spannungspfade streben asymptotisch den proportionalen Spannungspfad an, der mit derselben Verzerrung von einem spannungslosen Ausgangszustand ausgeht (siehe Goldscheider, 1976, S. 465).

[9] Synonym zu nichtbindig werden auch granular, rollig oder nicht-kohäsiv verwendet.

- Unter hohen Lasten ist auch ein Aufbrechen einzelner Körner möglich, was als *Kornbruch* bezeichnet wird. Dadurch ändert sich die Kornverteilung einer Bodenprobe und davon abgeleitete Größen, wie Dichte und Porenzahlen. Streng genommen sind nach dem Eintritt von Kornbruch keine aussagekräftigen Wiederholungsversuche mehr möglich.
- Die betrachteten Bodenkörper können keine Zugspannungen aufnehmen, da weder bindige Anteile, noch Teilsättigung oder Verkittung betrachtet werden.

2.2.1 Spannungs-Dehnungs-Verhalten von Boden

Das Spannungs-Dehnungs-Verhalten eines Bodenkörpers hängt nicht nur vom aktuellen Spannungszustand ab, sondern wird auch von weiteren Größen beeinflusst. Volumetrische Größen wie die Porenzahl beschreiben nach Wood (1990), S. 9 f., vereinfachend die makroskopisch und phänomenologisch messbare Änderung der mikroskopischen Bodenstruktur. Porenzahlen hängen vom umgebenden Druck bzw. Spannungszustand der Bodenprobe ab. Nach Wood (1990), S. 11, kann höchstens für Böden mit einem kleinen Bereich unterschiedlicher Korngrößen erwartet werden, dass sie mit Spannungszustand und Porenzahl als einziger volumetrischer Zustandsgröße aussagekräftig beschrieben werden können.

Mit den passenden Anfangswerten kann der Porenzahlverlauf aus dem Volumen (bzw. der Volumenänderung) einer Bodenprobe abgeleitet werden. Dabei wird angenommen, dass die Volumenänderung die Anordnung der Kornkonfiguration widerspiegelt, d. h. inkompressible Körner, kein Kornbruch und keine Makroporen.

Zusätzlich zu den Spannungen auf die Körner der Bodenstruktur kann auch das Fluid in den Poren unter Druck u stehen. Beide Spannungen ergeben die gesamte oder totale Spannung $\sigma = \sigma' + u$ auf das Bodenvolumen. Die *effektive Spannung* σ' bezeichnet die Spannung auf die Kornstruktur.

Annahmen und Einschränkungen in dieser Arbeit

- Eine eindimensionale Spannung wird mit σ bezeichnet und der allgemeine Spannungszustand mit $\mathbf{T}$.
- Im Rahmen dieser Arbeit wird das mechanische Verhalten von Bodenkörpern betrachtet. Biologische, chemische oder thermische Prozesse sowie deren Einfluss auf das Spannungs-Dehnungs-Verhalten werden nicht berücksichtigt.
- Alle Bodenproben sind ungesättigt oder werden vollständig gesättigt betrachtet (keine Teilsättigung).
- Alle Spannungen außerhalb dieses Abschnitts werden als effektive Spannungen angesehen und wie in Mašín (2019) wird auf eine explizite Kennzeichnung effektiver Spannungen durch einen Hochstrich verzichtet.

- Zugspannungen werden (wie in der Mechanik üblich) als positiv angenommen. Die Invarianten p, q und ε_v nach Roscoe und Burland (1968), S. 571 f., werden so definiert, dass sie positiv für Druckspannungen sind (siehe Abschnitt 2.3.2).
- Es wird angenommen, dass das Bodenverhalten nur von den effektiven Spannungen und Zustandsgrößen wie der Porenzahl sowie der intergranularen Dehnung abhängt und allein damit beschrieben werden kann.[10] Insbesondere hängen Steifigkeit und Festigkeit des Bodenmaterials vom mittleren Druck $p = -\operatorname{tr}(\mathbf{T})/3$ (*Barotropie*) und der Porenzahl e (*Pyknotropie*) ab.

2.2.2 Geotechnische Laborversuche zur Charakterisierung eines Bodens

Das mechanische Verhalten eines Bodens kann im bodenmechanischen Labor untersucht werden. Typische Grundlagenversuche für rollige Böden sind die Bestimmung

- der Korngrößenverteilung nach DIN EN ISO 17892-4:2017,
- der Korndichte nach DIN EN ISO 17892-3:2016 und
- der lockersten und dichtesten Lagerung nach DIN 18126:2022.

Darauf aufbauend kann das mechanische Verhalten bspw. am drainierten Triaxialversuch oder Ödometerversuch (mit stufenweiser Belastung) beobachtet werden. Diese Versuche werden im Folgenden nicht nur als reale Laborversuche, sondern auch als *Elementversuche* betrachtet, d. h. idealisierte Versuche mit homogenen Probenkörpern, gleichmäßig verteilten Lasten und Verformungen bei reibungsfreiem Versuchsablauf.

Um eine möglichst gute Vergleichbarkeit zu gewährleisten, sind die Randbedingungen bei der Versuchsdurchführung normativ geregelt. Sorgfältig durchgeführte Laborversuche bilden die Grundlage, um das Bodenverhalten zu charakterisieren. Typischerweise werden Ergebnisse von Laborversuchen für die Bestimmung der Bodenparameter vieler Stoffmodelle benötigt (siehe bspw. Abschnitt 6).

Die Herstellung einer möglichst homogenen Probe mit einer definierten Lagerungsdichte ist herausfordernd, vor allem kann auch die Homogenität der Verformung während der Versuchsdurchführung nicht erzwungen werden (Kolymbas und Herle, 2008, S. 247–248 und S. 281–282; sowie Kolymbas, 1988, S. 28–29).

Eine identisch reproduzierbare Probenherstellung wirkt aufgrund der unterschiedlichen Körner und ihrer Konfiguration nicht realisierbar. Zudem gibt es eine natürliche Streuung in einer Bodenprobe und eine inhärente Unsicherheit in der Bestimmung der Parameter (siehe nächster Abschnitt und bspw. Schneider-Muntau u. a., 2020, S. 277).

[10] Nach Wood (1990), S. 16, nur für ungesättigte/vollständig gesättigte Böden.

Drainierter Triaxialversuch

Der grundsätzliche Versuchsaufbau und die Durchführung von Triaxialversuchen mit zylindrischen Bodenproben ist in DIN EN ISO 17892-9:2018 geregelt.[11] Im Rahmen dieser Arbeit sind (isotrop) konsolidierte drainierte Versuche mit schlanken Proben durchgeführt worden (Probenhöhe $h_{\mathrm{ini}} \approx 125\,\mathrm{mm}$ und Verhältnis zum Durchmesser $h_{\mathrm{ini}}/d_{\mathrm{ini}} \approx 2{,}5$). Jede Probe ist nach Möglichkeit in der gewünschten Lagerungsdichte angefertigt und in das Triaxialgerät eingebaut worden (Messdaten und Ergebnisse in Kapitel 6 sowie Anhang I).

Eine Versuchsreihe besteht aus drei Proben mit möglichst gleicher Lagerungsdichte. Alle Proben werden nach dem Einbau ins Triaxialgerät bei einem Druck von 400 kPa gesättigt und anschließend mit entweder 450, 500 oder 600 kPa konsolidiert. Nach der Konsolidierung werden die Proben um 20 % abgeschert, während die folgenden Werte über die Versuchsdauer aufgezeichnet werden (siehe DIN EN ISO 17892-9:2018, S. 14):

- Axialkraft F,
- Zelldruck σ_r und Porenwasserdruck u,
- Längenänderung/Stauchung der Bodenprobe Δh und
- die Volumenänderung δV über das Porenwasservolumen, das sich in/aus der Probe bewegt (unter Annahme inkompressibler Körner).

Die prinzipiellen Spannungen und Dehnungen sind idealisiert in Bild 2.2 dargestellt. Für eine numerische Betrachtung des idealisierten Triaxialversuchs gilt in Anlehnung an Wu (1992), S. 41

$$\mathbf{T} = \begin{pmatrix} T_1 & \cdot & \cdot \\ \cdot & T_2 & \cdot \\ \cdot & \cdot & T_2 \end{pmatrix} \quad , \quad \mathbf{D} = \begin{pmatrix} D_1 & \cdot & \cdot \\ \cdot & D_2 & \cdot \\ \cdot & \cdot & D_2 \end{pmatrix} \quad \text{und} \quad \mathring{\mathbf{T}} = \begin{pmatrix} \mathring{T}_1 & \cdot & \cdot \\ \cdot & \mathring{T}_2 & \cdot \\ \cdot & \cdot & \mathring{T}_2 \end{pmatrix} \tag{2.1}$$

für den Spannungszustand $\mathbf{T}$, die Deformationsrate $\mathbf{D}$ und die (objektive) Spannungsänderung $\mathring{\mathbf{T}}$ (siehe Definitionen in Abschnitt 2.3.1).

Mit Korndichte ρ_S, Anfangshöhe h_{ini}, Anfangsdurchmesser d_{ini}, Trockenmasse m_S und der Annahme, dass sich das axial gestauchte Volumen gleichmäßig auf den Durchmesser verteilt, werden u. a. Axialspannung, mobilisierter Reibungswinkel und Porenzahlentwicklung berechnet. Üblicherweise neigen schlanke Proben ab einer gewissen Stauchung zum Ausbauchen oder Versagen durch eine Scherfuge und verlieren spätestens im Peak-Zustand $\sigma_1 = \sigma_{1,\max}$ ihre zylindrische Form.[12] Typischerweise werden die Volumenänderung $\Delta V/V_0$, die Deviatorspannung $\sigma_1 - \sigma_3$ oder der mobilisierte Reibungswinkel φ über der Dehnung $\varepsilon_1 = \Delta h/h_{\mathrm{ini}}$ aufgetragen.

[11] Im Gegensatz zum echten Triaxialversuch liegt in diesen Versuchen ein zweidimensionaler Spannungszustand vor (zylindrische Probe in einer Triaxialzelle).

[12] Untersuchungen mit CT-Aufnahmen zeigen, dass sich schon deutlich früher in der Probe ein inhomogenes Verformungsfeld einstellt. Nach Desrues, Andò, Mevoli Ando u. a. (2018) bei ungefähr 8 % Stauchung; siehe auch das dazugehörige Video unter »https://doi.org/10.13140/RG.2.2.15390.02880 sowie Desrues, Andò, Bésuelle u. a. (2017).

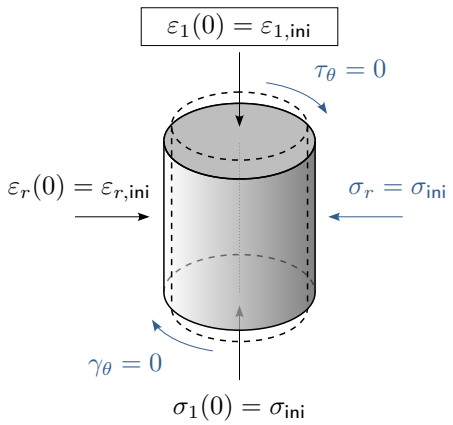


Bild 2.2: Spannungen und Dehnungen am idealisierten Triaxialversuch. Die vertikale Spannung sowie vertikale und radiale Dehnung werden zu Versuchsbeginn vorgegeben oder abgelesen. Der Versuch wird über die axiale Dehnung ε_1 gesteuert. Die radiale Spannung σ_r ist über den gesamten Versuch konstant während Scherspannung τ_θ und Scherdehnung γ_θ über den gesamten Versuch null sind.

Ödometerversuch mit stufenweiser Belastung

Ödometerversuche sind in DIN EN ISO 17892-5:2017 geregelt. Eine zylindrische Bodenprobe (Probenhöhe $h_{\text{ini}} \approx 30\,\text{mm}$, Probendurchmesser $d_{\text{ini}} = 100\,\text{mm}$) wird unter behinderter seitlicher Ausdehnung stufenweise vertikal belastet.[13]

Zuerst wird eine Probe in der gewünschten Lagerungsdichte angefertigt und in das Ödometergerät eingebaut. Anschließend wird über einen Stempel, an den alle 30 min Gewichte angehängt werden, schrittweise die vertikale Druckspannung angepasst (siehe Tabelle 2.1). Aufgrund des möglichst starren Rings um die Bodenprobe ist die radiale Dehnung im Vergleich zur axialen Dehnung vernachlässigbar.[14]

Im Ödometerversuch wird die Belastung vorgegeben und die Probenhöhe über die Versuchsdauer gemessen, die ebenfalls aufzuzeichnen ist (siehe bspw. DIN EN ISO 17892-5:2017, S. 17). Aus der Trockendichte ρ_d und der Korndichte ρ_S sowie den Anfangsdaten der Probe (Volumen V_{ini} und Anfangsporenzahl e_{ini}) kann die Entwicklung der Porenzahl über den anliegenden Druck ausgegeben werden.

Tabelle 2.1: Gewählte Druckspannungen bei den Ödometerversuch in Abhängigkeit von der Belastungsstufe. Bei Versuchen mit Be-, Ent- und Wiederbelastung werden die Bodenproben von Stufe 1 bis 5 belastet, wieder auf Stufe 2 entlastet und bis Stufe 7 erneut belastet.

S1 (kPa)	S2 (kPa)	S3 (kPa)	S4 (kPa)	S5 (kPa)	S6 (kPa)	S7 (kPa)
15	30	60	125	250	500	1000

[13] DIN EN ISO 17892-5:2017, S. 15, fordert mindestens sieben Belastungsstufen.
[14] In DIN EN ISO 17892-5:2017, S. 9, wird gefordert, dass sich der Durchmesser des Ödometerrings um die Bodenprobe bei maximaler Horizontalspannung im Versuch nicht mehr als 0,05 % ändern darf.

Die prinzipiellen Spannungen und Dehnungen sind idealisiert in Bild 2.3 dargestellt. Ebenfalls in Anlehnung an Wu (1992), S. 41, gilt für eine numerische Betrachtung des idealisierten Ödometerversuchs

$$\mathbf{T} = \begin{pmatrix} T_1 & \cdot & \cdot \\ \cdot & T_2 & \cdot \\ \cdot & \cdot & T_2 \end{pmatrix} \quad , \qquad \mathbf{D} = \begin{pmatrix} D_1 & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \quad \text{und} \qquad \mathring{\mathbf{T}} = \begin{pmatrix} \mathring{T}_1 & \cdot & \cdot \\ \cdot & \mathring{T}_2 & \cdot \\ \cdot & \cdot & \mathring{T}_2 \end{pmatrix} \quad .$$

Neben dem beschriebenen (kraftgesteuerten) Ödometerversuch mit stufenweiser Belastung, werden bspw. zur Bestimmung der Materialparameter für das hypoplastische Stoffmodell nach von Wolffersdorff (1996) auch weggesteuerte Ödometerversuche mit konstanter Dehnungsrate (*„CRS: Constant rate of strain“*) durchgeführt. Bei diesen CRS-Ödometerversuchen sind im Rahmen dieser Arbeit Bodenproben mit einer Geschwindigkeit von 0,1 mm/min bis zum Erreichen einer vertikalen Druckspannung von 2000 kPa belastet worden.

2.2.3 Genauigkeit der Messdaten

Bei Messungen kann aufgrund verschiedener Faktoren nicht erwartet werden, dass ein Messwert exakt ist.[15] Es wird zwischen zufälligen und systematischen Abweichungen unterschieden (siehe bspw. Bornath und Walter, 2020, S. 3), wobei zumindest die systematischen Abweichungen durch geeignete Maßnahmen minimiert werden können.[16] Bild 2.4 zeigt einige Einflussfaktoren auf die Messunsicherheit bei geotechnischen Laborversuchen.

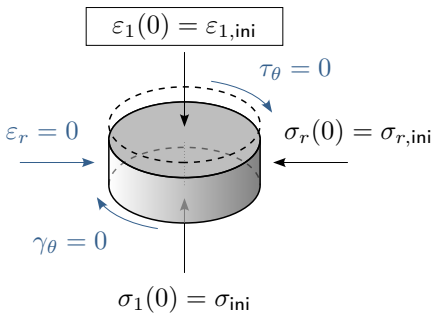


Bild 2.3: Spannungen und Dehnungen am idealisierten Ödometerversuch. Die vertikale Spannung sowie vertikale und radiale Dehnung werden zu Versuchsbeginn vorgegeben bzw. abgelesen. Der Versuch wird über die axiale Dehnung ε_1 gesteuert. Die radiale Dehnung ε_r, Scherspannung τ_θ und Scherdehnung γ_θ sind über den gesamten Versuch null.

[15] Der „wahre“ Wert ist im Allgemeinen unbekannt. Bornath und Walter (2020), S. 9, führen als anschauliches Beispiel einen Schützen an, der auf eine weiße Wand schießt. Für den Schützen wird eine Zielscheibe an die Wand projiziert, aber für die anschließende Auswertung ist die Projektion ausgeschaltet.

[16] Aus diesem Grund bezieht sich der Begriff Unsicherheit in Möhrke und Runge (2020), S. 42, nur auf zufällige Abweichungen.

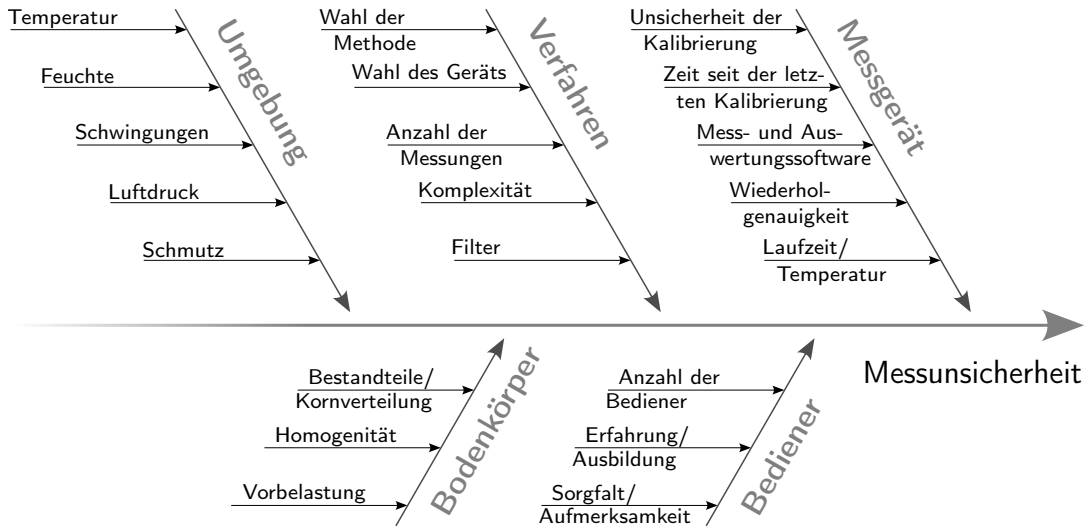


Bild 2.4: Eine Auswahl verschiedener Einflussfaktoren auf die Messunsicherheit geotechnischer Laborversuche (inspiriert von Testo, 2020, S. 14). Durch geeignete Maßnahmen kann der Einfluss der meisten dargestellten Faktoren auf das Messergebnis in Form einer Unsicherheit bewertet werden.

Aussagekräftige Messdaten werden nicht nur mit einer ausreichend hohen Genauigkeit ermittelt, die bspw. durch möglichst präzise Instrumente, eine störungsfreie Messumgebung und eine sorgfältige Durchführung gefördert wird. Sie erfordern auch eine realistische Abschätzung der Unsicherheit bei der Erhebung der Daten (siehe bspw. JCGM 100:2008 oder DIN 1319-3:1996).

Im Folgenden wird die Messunsicherheit für Messgeräte und Messverfahren, mit denen Daten erhoben worden sind, abgeschätzt (siehe Tabelle 2.2). Bei den späteren Messungen wird aus Wiederholungsversuchen die Standardabweichung der Messdaten bestimmt. Der größere Wert der beiden wird bei Messdaten als Unsicherheit angegeben. In Anhang G sind die dafür genutzten partiellen Ableitungen für die Bestimmung der Unsicherheit bei den Laborversuchen aufgelistet.

Die Einflussfaktoren des Bodenkörpers werden in der Geotechnik als *Unschärfe* bezeichnet. Da einige Faktoren wie das Korngefüge granularer Böden praktisch nicht kontrolliert werden können, bieten sich hierfür statistische Abschätzungen an.

Tabelle 2.2: Übersicht der Messunsicherheit der verwendeten Laborgeräte basierend auf den Genauigkeitsangaben der Hersteller und einer konservativen Abschätzung für eine sorgfältige Versuchsdurchführung.

Versuch	Messgröße	Unsicherheit
Schüttkegelversuch	(freier) Reibungswinkel	0,1 Grad
Korndichte	Gewicht Probe/Behälter	0,01 g
	Wassertemperatur	0,1 °C
Lockerste/dichteste Lagerung	Radius/Höhe Zylinderbauteil	0,05 mm
	Setzung der Probe	0,1 mm
	Gewicht Probe	0,01 g
Ödometerversuch	Kraft auf die Probe	1 N
	Radius/Höhe Zylinderbauteil	0,05 mm
	Setzung der Probe	0,1 mm
	(Referenzgewichte	0,6 %)
	(Gewicht Probe	0,01 g)
Triaxialversuch	Radius/Höhe Zylinderbauteil	0,05 mm
	Setzung der Probe	0,005 mm
	Kraft auf die Probe	1 N
	Druck (Zelle/Backpressure)	0,1 %
	Porenwasser/Backvolume	1 mm^3
	Gewicht Probe	0,01 g

2.3 Stoffmodelle für Böden

Die Frage nach der Eindeutigkeit eines Stoffgesetzes
ist nicht nur mathematisch interessant,
sondern hat auch gravierende Implikationen
für Numerik und Versuchstechnik.

Kolymbas und Herle (2008), S. 262

Stoffmodelle („*constitutive equations*“) werden benötigt, um das Verformungsverhalten eines Körpers (Kinematik) mit den dazugehörigen Kräften/Spannungen (Kinetik) zu verknüpfen, also das Spannungs-Dehnungs-Verhalten eines spezifischen Materials zu beschreiben. Sie bestimmen die mechanischen Eigenschaften eines Materials (das durch zusätzliche Materialparameter konkretisiert wird) und stellen somit eine Näherung an das tatsächliche Bodenverhalten dar. Zur Einordnung werden zunächst grundlegende Definitionen aus Kinematik und Kinetik sowie zur Materialabhängigkeit betrachtet.

2.3.1 Definitionen aus der Kontinuumsmechanik

Materialunabhängige Definitionen aus der Kinematik

Basierend auf Altenbach (2018), S. 81–110, können die folgenden Definitionen aus der Kinematik eines Kontinuums hergeleitet werden:

Die Verformung eines Körpers bzw. jedes Punktes darin kann zwischen zwei Zeitpunkten mit einem (materiellen) Deformationsgradienten $\mathbf{F}$ über

$$F_{ij} = \frac{\partial x_i}{\partial X_j} \quad \text{für} \quad i,j = 1,\ldots,3$$

beschrieben werden (siehe Bild 2.5). Dabei ist X_j die j-te Koordinate des Punktes in der Referenzkonfiguration und x_i die i-te Koordinate des Punktes nach der Verformung (Momentankonfiguration).[17]

Mithilfe des polaren Zerlegungssatzes kann der Deformationsgradient, wie jeder reguläre Tensor zweiter Stufe mit einer Determinante größer als null, eindeutig zerlegt werden in

$$\mathbf{F} = \mathbf{R}\mathbf{U}$$

mit einer orthogonalen Matrix $\mathbf{R}$ und einer symmetrisch, positiv definiten Matrix $\mathbf{U}$ (siehe auch Altenbach, 2018, S. 43). Dabei entspricht $\mathbf{R}$ einer Starrkörperrotation und $\mathbf{U}$ dem (rechten) Strecktensor[18]. Aus dem Strecktensor oder dem Deformationsgradienten können der (rechte) Cauchy-Green-Deformationstensor $\mathbf{C}$ und der Green-Langrange-Verzerrungstensor $\mathbf{G}$ berechnet werden als

$$\mathbf{C} = \mathbf{U}^2 = \mathbf{F}^T\mathbf{F} \quad \text{und} \qquad \mathbf{G} = \frac{1}{2}\left(\mathbf{C} - \mathbf{I}\right) \quad .$$

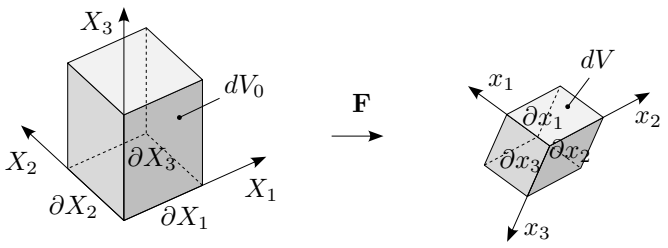


Bild 2.5: Der Deformationsgradient $\mathbf{F}$ beschreibt die Änderung zwischen zwei Zuständen und für ein Testvolumen gilt $\mathrm{d}V = |\det(\mathbf{F})|\mathrm{d}V_0$ (siehe Altenbach, 2018, S. 83).

[17] In der lagrangeschen oder materiellen Betrachtungsweise bewegt sich die Referenzkonfiguration mit, in der eulerschen (räumlichen) Betrachtungsweise ist die Referenzkonfiguration ortsfest. Für die folgenden Herleitungen wird die lagrangesche Betrachtungsweise genutzt.

[18] Viele Zusammenhänge aus der Kontinuumsmechanik sind für allgemeine Tensoren zweiter Stufe gültig, was sich in der Namensgebung widerspiegelt (bspw. Strecktensor oder Spannungstensor). Wie in Abschnitt 2.1 beschrieben, werden sie hier als Matrizen mit Standardbasis im dreidimensionalen Raum behandelt.

Der Verzerrungstensor ist dimensionslos und hat den Wert null für Starrkörperbewegungen. Der Geschwindigkeitsgradient $\mathbf{L}$ ist

$$L_{ij} = \frac{\partial v_i}{\partial x_j} \quad \text{mit} \qquad v_i = \frac{\mathrm{d}x_i}{\mathrm{d}t} \quad \text{für} \quad i,j = 1,\dots,3 \quad \text{und es gilt} \qquad \mathbf{L} = \dot{\mathbf{F}}\mathbf{F}^{-1} \quad .$$

Der symmetrische Teil des Geschwindigkeitsgradienten ist die Deformations- bzw. Dehnungsrate $\mathbf{D}$ und der antisymmetrische Teil die Drehmatrix $\mathbf{W}$, also

$$\mathbf{D} = \frac{1}{2}\left(\mathbf{L} + \mathbf{L}^T\right) \quad \text{und} \qquad\qquad \mathbf{W} = \frac{1}{2}\left(\mathbf{L} - \mathbf{L}^T\right) \quad .$$

Im Folgenden wird die linearisierte Dehnungsrate $\dot{\boldsymbol{\varepsilon}} \approx \mathbf{D}$ genutzt. Insbesondere wird in einem Zeitschritt mit konstanter Dehnung $\boldsymbol{\varepsilon} \approx \mathbf{D} \cdot dt$ verwendet. Für die Definition des Dehnungsmaßes gibt es mehrere Möglichkeiten[19], bspw.

$$\begin{aligned}
\varepsilon_{\text{Cauchy}} &= \frac{|\mathrm{d}\boldsymbol{x}| - |\mathrm{d}\boldsymbol{X}|}{|\mathrm{d}\boldsymbol{X}|} = \frac{|\mathbf{F}\,\mathrm{d}\boldsymbol{X}|}{|\mathrm{d}\boldsymbol{X}|} - 1 \quad , \\
\varepsilon_{\text{Green}} &= \frac{1}{2}\left(\frac{|\mathrm{d}\boldsymbol{x}|^2 - |\mathrm{d}\boldsymbol{X}|^2}{|\mathrm{d}\boldsymbol{X}|^2}\right) = \frac{1}{2}\left(\frac{\mathrm{d}\boldsymbol{X} \cdot \mathbf{C}\,\mathrm{d}\boldsymbol{X}}{|\mathrm{d}\boldsymbol{X}|^2} - 1\right) \quad \text{oder} \\
\varepsilon_{\text{Hencky}} &= \ln\left(\frac{|\mathrm{d}\boldsymbol{x}|}{|\mathrm{d}\boldsymbol{X}|}\right) = \ln\left(\frac{|\mathbf{F}\,\mathrm{d}\boldsymbol{X}|}{|\mathrm{d}\boldsymbol{X}|}\right) \quad .
\end{aligned}$$

Solange von kleinen Dehnungen ausgegangen werden kann, sind die Ergebnisse für die Dehnungsmaße annähernd gleich. Für finite Dehnungen wird üblicherweise ein logarithmisches Dehnungsmaß wie $\varepsilon_{\text{Hencky}}$ verwendet.

Materialunabhängige Definitionen aus der Kinetik

Basierend auf der Kinetik eines Kontinuums gibt es mehrere Möglichkeiten, den Spannungszustand auszudrücken. In der Momentankonfiguration kann der Spannungszustand mit dem symmetrischen[20] Cauchy-Spannungstensor $\mathbf{T}$ beschrieben werden, wie bspw. in Altenbach (2018), S. 145–148, definiert. Dazu wird ein Spannungsvektor $\boldsymbol{t}$ in einem Schnitt mit Flächenelement ΔA, Flächennormalen-Einheitsvektor $\boldsymbol{n}$ und dem Schnittkraftvektor $\Delta\boldsymbol{f}$ betrachtet mit

$$\boldsymbol{t}(\boldsymbol{x},\boldsymbol{n},t) = \lim_{\Delta A \to 0} \frac{\Delta \boldsymbol{f}}{\Delta A} = \boldsymbol{n}(\boldsymbol{x},t) \cdot \mathbf{T}(\boldsymbol{x},t) \quad .$$

Alternativ können mit dem ersten Piola-Kirchhoff-Spannungstensor ${}^{I}\mathbf{P}$ die Spannungen eines (materiellen) Punktes auf die Referenzkonfiguration bezogen werden, was im Allgemeinen zu

[19] Wobei beachtet werden sollte, dass keines dieser Dehnungsmaße das tatsächliche Integral der (linearisierten) Dehnungsrate ist.

[20] Der Cauchy-Spannungstensor ist immer symmetrisch, sofern die Drehimpulsbilanz erfüllt ist (siehe bspw. Altenbach, 2018, S. 188 f.).

einem unsymmetrischen Spannungstensor führt. Durch Modifikation mit einem zusätzlichen Kraftvektor kann daraus der symmetrische zweite Piola-Kirchhoff-Spannungstensor ${}^{II}\mathbf{P}$ hergeleitet werden (siehe bspw. Altenbach, 2018, S. 154–157). Es ergeben sich

$$
\begin{aligned}
&{}^{I}\boldsymbol{t}(\boldsymbol{X},\boldsymbol{n}_0,t) = \boldsymbol{n}_0(\boldsymbol{X},t)\cdot {}^{I}\mathbf{P}(\boldsymbol{X},t) \quad \text{mit} \quad \boldsymbol{n}_0 = \mathbf{F}^{-1}\boldsymbol{n} \quad \text{und} \quad \boldsymbol{t} = {}^{I}\boldsymbol{t}\,(\det(\mathbf{F})\,)^{-1} \quad \text{sowie} \\
&{}^{II}\boldsymbol{t}(\boldsymbol{X},\boldsymbol{n}_0,t) = \boldsymbol{n}_0(\boldsymbol{X},t)\cdot {}^{II}\mathbf{P}(\boldsymbol{X},t) \quad \text{mit} \quad {}^{II}\mathbf{P} = {}^{I}\mathbf{P}\left(\mathbf{F}^{-1}\right)^{T} \quad .
\end{aligned}
$$

Für den Kirchhoff-Spannungstensor gilt $\mathbf{T}^K = \det(\mathbf{F})\,\mathbf{T}$. Im Folgenden ist mit dem Begriff Spannungstensor der Cauchy-Spannungstensor gemeint.

Materialabhängige Definitionen

Truesdell und Noll (2004), S. 58, definieren eine allgemeine Formulierung für ein Stoffmodell aus den beiden Bedingungen,

- dass der zeitliche Verlauf der Spannungen an einem Materialpunkt allein aus der Verformungsgeschichte einer beliebig kleinen Umgebung um den Materialpunkt bestimmt werden kann und
- dass das Materialverhalten unabhängig vom Koordinatensystem des Betrachters ist und beliebigen (zeitabhängigen) Drehungen davon, was als *Objektivität* (*„Principle of material frame-indifference“*) bezeichnet wird.[21]

Für die meisten Materialien kann das Materialverhalten allein aus dem Verlauf des Deformationsgradienten ausreichend gut beschrieben werden. Truesdell und Noll (2004), S. 60, definieren solche Materialien als *einfache Materialien* mit

$$
\mathbf{T}(t) = \mathfrak{G}\left(\mathbf{F}^{(t)}\right) \quad . \tag{2.2}
$$

Dabei ist $\mathfrak{G}$ ein Funktional, das einer tensorwertigen Funktion des Deformationsgradienten $\mathbf{F}^{(t)}$ einen symmetrischen Spannungstensor $\mathbf{T}$ zuweist.

Alle im Folgenden beschriebenen Stoffmodelle erfüllen diese Definition eines einfachen Materials. Beim Hookeschen Gesetz liegt bspw. eine lineare Kopplung des Spannungstensors und den Dehnungen vor. Bei der Hypoplastizität und der Barodesie ist die objektive Spannungsrate mit der Dehnungsrate gekoppelt.

Wie bspw. Altenbach (2018), S. 223 f., aufzeigt, ist die Dehnungsrate $\mathbf{D}$ objektiv, für den (Cauchy-)Spannungstensor $\mathbf{T}$ wird Objektivität angenommen, aber die Spannungsrate $\dot{\mathbf{T}}$ ist nicht objektiv.[22] An dieser Stelle kann eine objektive Spannungsrate, wie bspw. die Jaumann-Zaremba-Spannungsrate $\overset{\circ}{\mathbf{T}}{}^{J}$ oder die Green-McInnis-Naghdi-Spannungsrate $\overset{\circ}{\mathbf{T}}{}^{G}$, verwendet werden (siehe bspw. Simo und Hughes, 1998, S. 255), also

[21] Ein geschichtlicher Hintergrund zur Objektivität findet sich bspw. in Truesdell und Noll (2004), S. 45–47.

[22] „Die materielle Zeitableitung objektiver Tensoren beliebiger Stufe führt nicht zwangsläufig auf objektive Tensorraten“ (Altenbach, 2018, S. 224).

$$\overset{\circ}{\mathbf{T}}{}^{J} = \dot{\mathbf{T}} - \mathbf{W}\mathbf{T} + \mathbf{T}\mathbf{W} \quad \text{bzw.}$$
$$\overset{\circ}{\mathbf{T}}{}^{G} = \dot{\mathbf{T}} - \boldsymbol{\Omega}\mathbf{T} + \mathbf{T}\boldsymbol{\Omega} \quad \text{mit} \quad \boldsymbol{\Omega} = \dot{\mathbf{R}}\mathbf{R}^{T} \quad .$$

Für die Diskussion der Stoffmodelle werden noch folgende Definitionen verwendet:

- Sofern die Materialeigenschaften an jedem Punkt in einem betrachteten Teilkörper gleich sind, wird der Teilkörper als *homogen* bezeichnet.
- Wenn die Materialeigenschaften keine Richtungsabhängigkeit aufweisen, wird das Material als *isotrop* bezeichnet. In Teilkörpern mit isotropem Materialverhalten ist bspw. die Reaktion auf eine äußere Kraft identisch mit der (gedrehten) Reaktion der gedrehten äußeren Kraft am gleichen Teilkörper (siehe dazu auch Truesdell und Noll, 2004, S. 23).

2.3.2 Definitionen für Bodenmaterialien

Alle in dieser Arbeit betrachteten Stoffmodelle sind für einen allgemeinen, dreidimensionalen Zustand definiert. Dazu ist es zweckmäßig, den Bodenkörper als Kontinuum anzunehmen, was u. a. nach Kolymbas (2022), S. 38, eine weitreichende Idealisierung für ein granulares Material darstellt. Einfache Stoffmodelle wie die Elastizität nutzen eine direkte Abhängigkeit zwischen Dehnung und Spannung in der Form $\mathbf{T} = f(\boldsymbol{\varepsilon}, \dots)$.

Andere Stoffmodelle, wie Hypoplastizität und Barodesie, sind *ratenabhängig* formuliert, d. h. sie beschreiben die Änderung der Spannungsrate basierend auf der Dehnungsrate $\dot{\mathbf{T}} = f(\mathbf{D}, \dots)$. Oftmals wird die Bezeichnung „inkrementell nichtlinear“ verwendet, um eine nichtlineare Ratenabhängigkeit der tangentialen Steifigkeit von der Lastrichtung zu beschreiben (siehe auch Mašín, 2019, S. 3).

Kolymbas (2022), S. 86, weist darauf hin, dass die zuvor beschriebene Annahme $\dot{\boldsymbol{\varepsilon}} \approx \mathbf{D}$ in Stoffmodellen wie Hypoplastizität und Barodesie nur eine eingeschränkte Gültigkeit hat und nur für Bewegungen ohne Rotation der Hauptspannungsachsen mit einem logarithmischen Dehnungsmaß gültig ist bzw. sein kann.

Invarianten des Spannungstensors

Der Spannungstensor $\mathbf{T}$ kann in beliebigen Koordinatensystemen ausgedrückt werden. Wie in Zienkiewicz u. a. (2014), S. 101 f., können die Spannungszustände für ein isotropes Material in Abhängigkeit von drei Spannungsinvarianten ausgedrückt werden (siehe bspw. Abschnitt 2.1):

$$I_1 = \operatorname{tr}(\mathbf{T}) \quad , \qquad 2J_2 = \mathbf{T}^{\mathrm{dev}} : \mathbf{T}^{\mathrm{dev}} \quad \text{und} \qquad 3J_3 = \det\left(\mathbf{T}^{\mathrm{dev}}\right) \quad .$$

Da die Invarianten direkt von den Hauptspannungen abhängen, erfüllt eine solche Formulierung das Prinzip der Objektivität (siehe auch Truesdell und Noll, 2004, S. 26).

Dadurch ergeben sich bspw. die ursprünglich in Roscoe und Burland (1968), S. 571 f., definierten Größen

$$p = -\frac{I_1}{3} = -\frac{T_{11} + T_{22} + T_{33}}{3} \quad ,$$

$$q = \sqrt{3J_2} = \sqrt{T_{11}^2 + T_{22}^2 + T_{33}^2 - T_{11}T_{22} - T_{22}T_{33} - T_{11}T_{33} + 3\left(T_{12}^2 + T_{23}^2 + T_{13}^2\right)} \quad \text{und}$$

$$\cos(3\theta) = -\frac{3\sqrt{3}J_3}{2J_2^{3/2}} \qquad \text{oder} \qquad \theta^* = \frac{1}{3}\operatorname{asin}\left(\frac{3\sqrt{3}J_3}{2J_2^{3/2}}\right)$$

mit der mittleren Spannung p, der Deviatorspannung q und dem sog. Lode-Winkel θ mit $-30\,\text{Grad} < \theta \leq 30\,\text{Grad}$ für die Beschreibung eines Spannungszustands (siehe auch Darstellung in Bild 2.6).

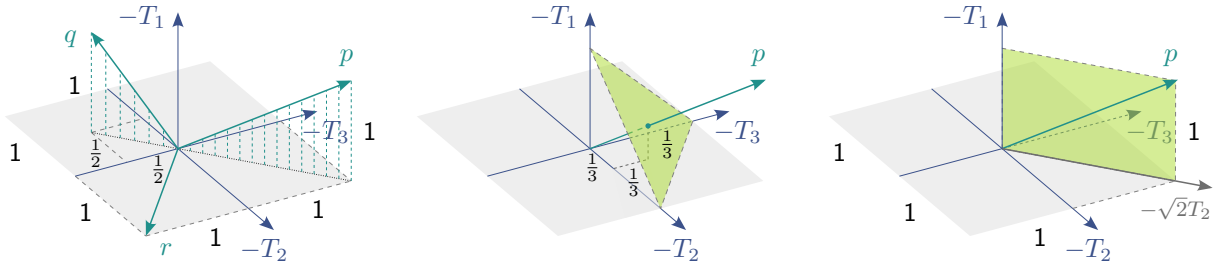
Die Wahl eines Koordinatensystems mit p als Koordinatenrichtung hat den Vorteil, dass der hydrostatische Spannungszustand auf der Achse selbst liegt. Dadurch können bspw. Grenzbedingungen, die nicht vom hydrostatischen Spannungszustand abhängen, in der Deviatorebene senkrecht zu p dargestellt werden.

Elastizität und unabhängige Tensoreinträge

Der einfachste Zusammenhang zwischen dem Spannungstensor und dem Dehnungstensor ist eine lineare Abbildung mit einem S4-Tensor als

$$\boldsymbol{\sigma} = \mathcal{C} : \boldsymbol{\varepsilon} \quad .$$

Der Materialtensor $\mathcal{C}$ hat 81 Komponenten, aber höchstens 36 linear unabhängige Einträge, da ein symmetrischer Dehnungstensor auf einen symmetrischen Spannungstensor abgebildet wird (siehe bspw. Sadd, 2014, S. 83).[23]



(a) Skizze **(b)** Deviatorebene **(c)** Triaxialebene

Bild 2.6: Ein Spannungszustand kann in verschiedenen Koordinaten ausgedrückt werden. Anstelle der Raumachsen kann der mittlere Spannungszustand mit p, q und θ ausgedrückt werden, wobei q und θ die Deviatorebene orthogonal zu p aufspannen. Da p sich zu gleichen Anteilen aus T_1, T_2 und T_3 zusammensetzt, liegt die Achse in der Triaxialebene, für die $T_2 = T_3$ gilt. Die durchgeführten Laborversuche mit Ödometer- und Triaxialgerät sind definitionsgemäß immer in der Triaxialebene.

[23] Die notwendige Bedingung lautet $C_{ijkl} = C_{jikl} = C_{ijlk} = C_{jilk}$.

Wird ein isotropes Material angenommen (Drehnungsinvarianz und Spieglungsinvarianz), so hat $\mathcal{C}$ höchstens drei unabhängige Einträge (siehe auch Abschnitt 2.1). Das führt auf die Definition der linearen *Elastizität* eines isotropen Materials mit dem Materialtensor $\mathcal{C}^{\mathrm{el}}$ (Hookesches Gesetz). Dieser Materialtensor hat nur zwei unabhängige Einträge, das Elastizitätsmodul E und die Querdehnzahl ν. Es gilt

$$\mathcal{C}^{\mathrm{el}} = \frac{E}{1+\nu}\mathcal{I}^{\mathrm{sym}} + \frac{E\nu}{1-\nu+2\nu^2}\mathcal{I}^{\mathrm{tr}} \quad .$$

Grenzbedingungen und ideale Plastizität

Für die ideale Plastizität wird das elastische Verhalten mit einer Grenzbedingung erweitert (bspw. Mohr-Coulomb oder Matsuoka-Nakai). Dadurch sind im Allgemeinen zusätzliche Parameter, wie Reibungswinkel φ_c und Kohäsion c, zum Beschreiben des Materialverhaltens notwendig. Das Materialverhalten ist elastisch bis die Grenzbedingung erfüllt wird ($f \leq 0$), anschließend kann eine plastische Verformung über eine Fließregel definiert werden.[24]

Mit den Invarianten aus diesem Abschnitt und Abschnitt 2.1 gilt nach von Wolffersdorff (1996), S. 256, und Bardet (1990), S. 500,

$$f_{\mathrm{MC}} = \sqrt{J_2}\cdot\left(3\cos(\theta^*) - \sqrt{3}\sin(\varphi_c)\sin(\theta^*)\right) + I_1\sin(\varphi_c) - 3c\cos(\varphi_c) = 0 \quad \text{und}$$
$$f_{\mathrm{MN}} = -\frac{I_3}{I_1 I_2} + \frac{\cos(\varphi_c)^2}{9-\sin(\varphi_c)^2} = 0 \quad .$$

So ergibt sich bspw. nach Wood (1990), S. 177, für die triaxiale Kompression und Extension mit der Grenzbedingung nach Mohr-Coulomb (und an diesen Stellen auch nach Matsuoka-Nakai, siehe Bild 2.7)

$$M_{\mathrm{komp}} = \frac{6\sin(\varphi_c)}{3-\sin(\varphi_c)} \quad \text{und} \qquad M_{\mathrm{ext}} = \frac{6\sin(\varphi_c)}{3+\sin(\varphi_c)} \quad \text{mit} \qquad q = Mp \quad .$$

In der Hypoplastizität nach von Wolffersdorff (1996) ist die Grenzbedingung nach Matsuoka-Nakai direkt eingebaut, während in der Barodesie ungefähr die gleiche Grenzbedingung gilt (zumindest für Reibungswinkel $\varphi_c < 40\,\mathrm{Grad}$, siehe Fellin und Ostermann, 2013, S. 308).

Antwortumhüllende

Das Konzept der Antwortumhüllenden kann genutzt werden, um das lokale Verhalten ratenabhängiger Stoffmodelle zu betrachten. Wie in Gudehus und Mašín (2009), S. 147 f., beschrieben, wird die Änderung eines Spannungszustands in der Triaxialebene aus verschiedenen Richtungen betrachtet mit einer Referenzdehnungsrate

$$|D_{\mathrm{ref}}| = \sqrt{\dot{\varepsilon}_1^2 + 2\dot{\varepsilon}_2^2} = 1 \quad .$$

[24] Für die Abschätzung einer Volumenänderung kommt in der Fließregel meistens der Dilatanzwinkel ψ zum Einsatz. Sofern keine Daten dazu vorliegen, empfiehlt DGGT (2014), S. 15, $\psi = \varphi_c - 30\,\mathrm{Grad}$ für Reibungswinkel $\varphi_c \geq 30\,\mathrm{Grad}$ bzw. $\psi = 0$ sonst.

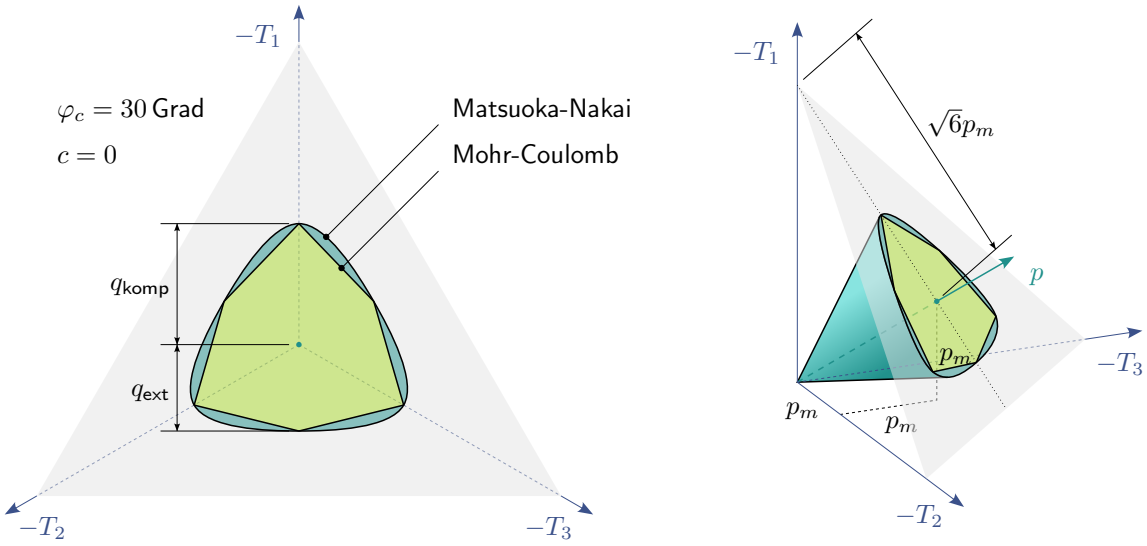


Bild 2.7: Grenzbedingungen nach Mohr-Coulomb und Matsuoka-Nakai in der Deviatorebene für $\varphi_c = 30\,\text{Grad}$, $c = 0$ und der Referenzspannung $p_{\text{ref}} = \text{tr}(\mathbf{T})/3 = \sqrt{3}p_m$ und $q_{\text{ref}} = M\sqrt{\frac{2}{3}}p_m$. Die gültigen Spannungszustände sind innerhalb der gekennzeichneten Flächen (innerhalb der Deviatorebene) bzw. Volumina im Spannungsraum.

Je nach Formulierung des Stoffmodells ergeben sich unterschiedliche Formen für eine Kontur durch die Spannungsänderungen (siehe Bild 2.8). In Chambon u. a. (1994), S. 258, wird darauf hingewiesen, dass die Antwortumhüllenden nur zweidimensionale Projektionen der tatsächlichen Spannungs-Dehnungsbeziehung sind. Folglich können die Beziehungen der Stoffgleichungen in der Projektion kontinuierlich wirken, ohne es tatsächlich zu sein.

Phänomenologische Stoffmodelle

Stoffmodelle können eingeteilt werden in physikalische und phänomenologische Modelle. Physikalische Stoffmodelle lassen sich von Gesetzmäßigkeiten der Thermodynamik bzw. der Energieerhaltung herleiten. Im Gegensatz dazu basieren *phänomenologische Stoffmodelle* auf Zusammenhängen, die aus dem beobachtbaren Materialverhalten abgeleitet werden können. Die als zentral bewerteten Eigenschaften eines Materials werden im Rahmen des Stoffmodells durch mathematische Zusammenhänge ausgedrückt. So sollten nach Kolymbas und Herle (2008), S. 260–262, Stoffmodelle für Böden

- objektiv und somit unabhängig vom Beobachtungsrahmen sein,
- ratenabhängig in der Form $\mathring{\mathbf{T}} = \mathring{\mathbf{T}}(\mathbf{D})$ formuliert sein, um einen Einfluss der vorangegangenen Belastungsgeschichte zu ermöglichen,
- vom aktuellen Spannungszustand abhängen, also $\mathring{\mathbf{T}}(\mathbf{D},\mathbf{T})$, damit die Steifigkeit von der Spannung abhängt und
- nichtlinear in $\mathbf{D}$ zur Beschreibung von irreversiblen Verformungen sein, sodass $\mathring{\mathbf{T}}(\mathbf{T},\mathbf{D}) \neq -\mathring{\mathbf{T}}(\mathbf{T},-\mathbf{D})$ gilt.

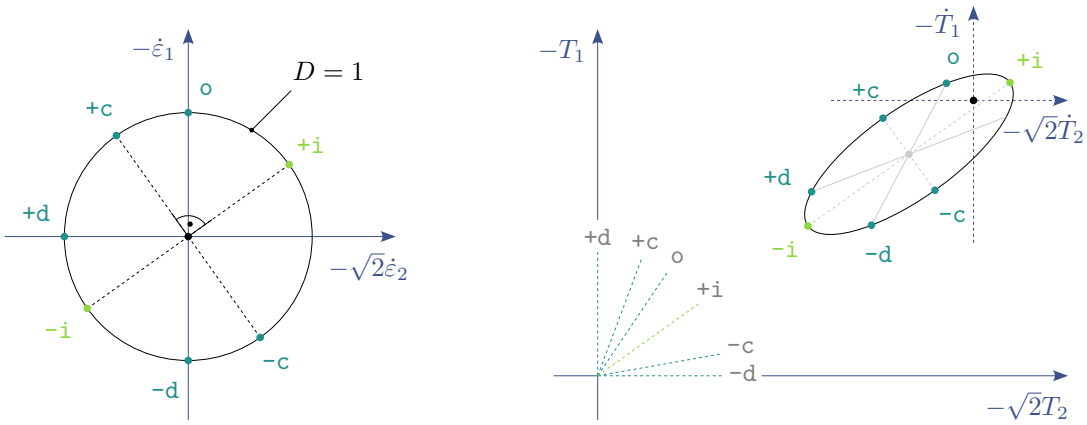


Bild 2.8: Für Referenzdehnungsraten der Länge $D = 1$ in der Triaxialebene (links) werden die Änderungen der Spannungen rund um einen Spannungszustand in Form einer Antwortumhüllenden untersucht (rechts). In beiden Plots sind charakteristische Spannungs- und Dehnungspfade dargestellt: Die isotrope Kompression mit `+i` und (isochor) triaxiale Kompression mit `+c` (Extensionen mit negativem Vorzeichen), das theoretische asymptotische Verhalten mit `+d`/`-d` sowie die ödometrische Kompression mit `o`. Im dargestellten Beispiel (orientiert am hypoplastischen Stoffmodell) setzt sich die Antwort im Spannungsraum aus einer linearen Transformation und einer Verschiebung zusammen.

Die im Rahmen dieser Arbeit genutzten hypoplastischen und barodetischen Stoffmodelle sind phänomenologisch und haben die aufgelisteten Eigenschaften. Für eine allgemeinere Abhandlung von Stoffmodellen sei auf Simo und Hughes (1998) verwiesen. Eine Übersicht zu Stoffmodellen in der Geotechnik findet sich bspw. im entsprechenden Kapitel des Grundbau-Taschenbuchs (Kolymbas und Herle, 2008, S. 243–287) oder den Empfehlungen des Arbeitskreises Numerik (DGGT, 2014, S. 11–19). Weitere Materialmodelle für Böden, für die eine Implementierung in der xMat interessant wäre, sind

- Cam Clay (siehe Roscoe und Burland, 1968; oder Wood, 1990, S. 112–136),
- SANISAND nach Dafalias und Manzari (2004),
- Hardening Soil (siehe Schanz u. a., 1999; und Benz, 2007),
- Hypoplastizität auf Basis des CLoE-Modells (siehe bspw. Chambon u. a., 1994; oder Tamagnini, Viggiani und Chambon, 2000, für eine Übersicht),
- Hypoplastizität für bindige Böden (siehe Mašín, 2005; Mašín, 2012; oder Weifner und Kolymbas, 2007, für ein hypoplastisches Stoffmodell für bindige und nichtbindige Böden),
- zyklische Hypoplastizität, wie ISA („*Intergranular Strain Anisotropy*“) nach Fuentes Lacouture (2014) oder HCA („*High Cycle Accumulation*“) nach Niemunis, Wichtmann u. a. (2005) und
- Barodesie für bindige Böden (siehe bspw. Medicus, Fellin und Kolymbas, 2012; Medicus, 2014; und auch die Variante in Kolymbas, 2021).

2.3.3 Hypoplastizität

Zusätzlich zu den bereits aufgelisteten Eigenschaften aus dem vorangegangenen Abschnitt werden nach Wu (1992), S. 12, für ein hypoplastisches Stoffmodell Homogenität erster Ordnung in $\mathbf{D}$ gefordert (Geschwindigkeitsunabhängigkeit), sodass $\mathring{\mathbf{T}}(\mathbf{T}, x\mathbf{D}) = x\mathring{\mathbf{T}}(\mathbf{T}, \mathbf{D})$ für $x > 0$ gilt, und Homogenität n-ter Ordnung in $\mathbf{T}$ (Proportionalität) mit $\mathring{\mathbf{T}}(x\mathbf{T}, \mathbf{D}) = x^n\mathring{\mathbf{T}}(\mathbf{T}, \mathbf{D})$. Je nach Formulierung können noch weitere Abhängigkeiten bestehen.

Nach Kolymbas (2011a), S. 1220, ist die Hauptcharakteristik der Hypoplastizität, dass sie auf rationaler Mechanik beruht und ohne Merkmale der Elastoplastizität wie bspw. Grenzfläche oder plastisches Potenzial auskommt. Die im Folgenden betrachteten hypoplastischen Stoffmodelle basieren auf der Gleichung

$$\mathring{\mathbf{T}} = \mathcal{L} : \mathbf{D} + f_d \mathbf{N} \, ||\mathbf{D}|| \tag{2.3}$$

mit einer Beschreibung durch einen Materialtensor $\mathcal{L}$, einer zusätzlichen Matrix $\mathbf{N}$ und einem Faktor f_d. Eine allgemeinere Definition nach Mašín (2019), S. 5, lautet

$$\mathring{\mathbf{T}} = \mathcal{M}\,(\mathbf{T},\mathbf{q},\vec{\mathbf{D}}) : \mathbf{D} \tag{2.4}$$

mit einem Materialtensor $\mathcal{M}$, der neben der Spannung explizit von Statusvariablen $\mathbf{q}$ (bspw. Porenzahl und intergranulare Dehnung) sowie der Lastrichtung $\vec{\mathbf{D}}$ abhängt.

2.3.3.1 Hypoplastisches Modell nach Wu (1992)

In der geschichtlichen Einordnung (siehe Kolymbas und Medicus, 2016; und Mašín, 2019, S. 73–84) beginnt die Entwicklung der hypoplastischen Stoffmodelle mit ratenabhängigen Stoffmodellen in Kolymbas (1977).[25] Eine überarbeitete Version des Stoffmodells wird in der Dissertation von Kolymbas (1988) vorgestellt. Darauf aufbauend wird das folgende Modell von Wu (1992) definiert als

$$\mathring{\mathbf{T}}(\mathbf{T}, \mathbf{D}) = C_1 \operatorname{tr}(\mathbf{T})\, \mathbf{D} + C_2 \frac{\operatorname{tr}(\mathbf{TD})\, \mathbf{T}}{\operatorname{tr}(\mathbf{T})} + C_3 \frac{\mathbf{T}^2}{\operatorname{tr}(\mathbf{T})} \, ||\mathbf{D}|| + C_4 \frac{\left(\mathbf{T}^{\mathrm{dev}}\right)^2}{\operatorname{tr}(\mathbf{T})} \, ||\mathbf{D}|| \tag{2.5}$$

mit den Materialparametern C_1, C_2, C_3 und C_4. Laut Wu (1992), S. 11, sind die einzelnen Terme aus dem Darstellungstheorem tensorieller Funktionen ausgewählt, womit eine objektive Formulierung ermöglicht wird.

Mit $\mathcal{I}^{id}$ und der Beziehung

$$\operatorname{tr}(\mathbf{TD})\, \mathbf{T} = (\mathbf{T} \otimes \mathbf{T}) : \mathbf{D} = T_{ij} T_{kl} D_{kl}$$

[25] Die Bezeichnung *Hypoplastizität* geht auf Dafalias (1986) zurück.

kann das Modell relativ einfach in die Form von Gleichung (2.3) übertragen werden, d. h.

$$\begin{aligned}\mathring{\mathbf{T}} &= \mathcal{L} : \mathbf{D} + f_d \mathbf{N}\,||\mathbf{D}|| \qquad \text{mit}\\ \mathcal{L} &= C_1\,\mathrm{tr}(\mathbf{T})\,\mathcal{I}^{id} + C_2\frac{\mathbf{T}\otimes\mathbf{T}}{\mathrm{tr}(\mathbf{T})} \qquad \text{und}\\ \mathbf{N} &= C_3\frac{\mathbf{T}^2}{\mathrm{tr}(\mathbf{T})} + C_4\frac{\left(\mathbf{T}^{\mathrm{dev}}\right)^2}{\mathrm{tr}(\mathbf{T})}\end{aligned}$$

mit $f_d = 1$ und den vier Materialparametern.

Die Bestimmung der Materialparameter kann, wie in Anhang A von Wu (1992), S. 142–144, beschrieben, an einem triaxialen Kompressionsversuch durchgeführt werden.[26] Aus dem Versuch werden Spannung, Spannungsänderung und Dehnungsinkremente von zwei unterschiedlichen Zuständen A und B extrahiert und können analog zu Gleichung (2.1) dargestellt werden. Die entkoppelten Gleichungen können mit der Matrixnorm aus Abschnitt 2.1 dargestellt werden als

$$\mathring{T}_1 = C_1 \underbrace{(T_1+2T_2)}_{M_1} D_1 + C_2 \underbrace{\frac{T_1D_1+2T_2D_2}{T_1+2T_2}}_{M_2} T_1 + C_3 \underbrace{\frac{\sqrt{D_1^2+2D_2^2}}{T_1+2T_2}}_{M_3} T_1^2 + C_4 \underbrace{\frac{\sqrt{D_1^2+2D_2^2}}{T_1+2T_2}}_{M_3} \frac{4}{9}(T_1-T_2)^2 \quad \text{und}$$

$$\mathring{T}_2 = C_1 \underbrace{(T_1+2T_2)}_{M_1} D_2 + C_2 \underbrace{\frac{T_1D_1+2T_2D_2}{T_1+2T_2}}_{M_2} T_2 + C_3 \underbrace{\frac{\sqrt{D_1^2+2D_2^2}}{T_1+2T_2}}_{M_3} T_2^2 + C_4 \underbrace{\frac{\sqrt{D_1^2+2D_2^2}}{T_1+2T_2}}_{M_3} \frac{1}{9}(T_1-T_2)^2 \quad .$$

Damit sind die Parameter C_1 bis C_4 die Lösung des folgenden linearen Gleichungssystems

$$\begin{bmatrix}\mathring{T}_1^A\\ \mathring{T}_2^A\\ \mathring{T}_1^B\\ \mathring{T}_2^B\end{bmatrix} = \begin{pmatrix} M_1^A D_1^A & M_2^A T_1^A & M_3^A\cdot\left(T_1^A\right)^2 & M_3^A\cdot\frac{4}{9}\left(T_1^A-T_2^A\right)^2\\ M_1^A D_2^A & M_2^A T_2^A & M_3^A\cdot\left(T_2^A\right)^2 & M_3^A\cdot\frac{1}{9}\left(T_1^A-T_2^A\right)^2\\ M_1^B D_1^B & M_2^B T_1^B & M_3^B\cdot\left(T_1^B\right)^2 & M_3^B\cdot\frac{4}{9}\left(T_1^B-T_2^B\right)^2\\ M_1^B D_2^B & M_2^B T_2^B & M_3^B\cdot\left(T_2^B\right)^2 & M_3^B\cdot\frac{1}{9}\left(T_1^B-T_2^B\right)^2\end{pmatrix}\begin{bmatrix}C_1\\ C_2\\ C_3\\ C_4\end{bmatrix} \quad .$$

Bei Triaxialversuchen mit konstantem Seitendruck gilt außerdem $\mathring{T}_2^A = \mathring{T}_2^B = 0$.

Das Stoffmodell ist in Wu, Lin u. a. (2017), S. 1376, überarbeitet worden zu

$$\mathring{\mathbf{T}}(\mathbf{T},\mathbf{D}) = \hat{C}_1\,\mathrm{tr}(\mathbf{T})\,\mathbf{D} + \hat{C}_2\mathbf{T}\,\mathrm{tr}(\mathbf{D}) + \hat{C}_3\frac{\mathrm{tr}(\mathbf{TD})\,\mathbf{T}}{\mathrm{tr}(\mathbf{T})} + \hat{C}_4\cdot\left(\mathbf{T}+\mathbf{T}^{\mathrm{dev}}\right)||\mathbf{D}||$$

mit vier anderen Materialparametern $\hat{C}_1$, $\hat{C}_2$, $\hat{C}_3$ und $\hat{C}_4$.

[26] Siehe auch Wu und Bauer (1994), S. 840–844, für die Bestimmung der Parameter. Alternativ können die Parameter auch aus einem Ödometerversuch bestimmt werden (siehe Bauer, 1992, S. 69–72).

2.3.3.2 Hypoplastisches Modell nach von Wolffersdorff (1996)

In den Jahren 1992 bis 1996 sind zahlreiche Arbeiten an hypoplastischen Stoffgleichungen erschienen. Insbesondere die folgenden Arbeiten führen zu dem in diesem Abschnitt vorgestellten Referenzmodell:

- Kolymbas, Herle und von Wolffersdorff (1995) mit der Implementierung zusätzlicher interner Parameter,
- Bauer (1996) mit einer Abschätzungsformel für Grenzporenzahlen,
- Gudehus (1996) mit einer überarbeiteten Modellgleichung und
- von Wolffersdorff (1996) mit der Implementierung der Grenzbedingung nach Matsuoka-Nakai.

Die Stoffgleichung für das Referenzmodell lautet

$$\mathring{\mathbf{T}}(\mathbf{T},\mathbf{D},e) = \frac{f_b f_e}{\mathrm{tr}\left(\hat{\mathbf{T}}^2\right)}\left[F^2\mathbf{D} + a^2\,\mathrm{tr}\left(\hat{\mathbf{T}}\mathbf{D}\right)\hat{\mathbf{T}} + f_d a F\cdot\left(\hat{\mathbf{T}}+\hat{\mathbf{T}}^{\mathrm{dev}}\right)||\mathbf{D}||\right] \tag{2.6}$$

und in der Form von Gleichung (2.3)

$$\begin{aligned}
\mathring{\mathbf{T}} &= \mathcal{L}:\mathbf{D} + f_d\mathbf{N}\,||\mathbf{D}|| \qquad \text{mit}\\
\mathcal{L} &= \frac{f_b f_e}{\hat{\mathbf{T}}:\hat{\mathbf{T}}}a^2\left[\left(\frac{F}{a}\right)^2\mathcal{I}^{\mathrm{sym}} + \hat{\mathbf{T}}\otimes\hat{\mathbf{T}}\right] \qquad \text{und}\\
\mathbf{N} &= \frac{f_b f_e}{\hat{\mathbf{T}}:\hat{\mathbf{T}}}a^2\left(\frac{F}{a}\right)\left(\hat{\mathbf{T}}+\hat{\mathbf{T}}^{\mathrm{dev}}\right) \quad .
\end{aligned}$$

Für das Stoffmodell werden die acht Materialparameter φ_c, h_s, n_H, e_{d0}, e_{c0}, e_{i0}, α_H und β_H benötigt, die über die folgenden Zusammenhänge in das Stoffmodell eingehen:

$$F = \sqrt{\frac{1}{8}\tan(\psi)^2 + \frac{2-\tan(\psi)^2}{2+\sqrt{2}\tan(\psi)\cos(3\theta)}} - \frac{1}{2\sqrt{2}}\tan(\psi) \quad ,$$

$$\tan(\psi) = \sqrt{3}\,||\hat{\mathbf{T}}^{\mathrm{dev}}|| \quad , \qquad \cos(3\theta) = -\sqrt{6}\frac{\mathrm{tr}\left(\left(\hat{\mathbf{T}}^{\mathrm{dev}}\right)^3\right)}{\left(\hat{\mathbf{T}}^{\mathrm{dev}}:\hat{\mathbf{T}}^{\mathrm{dev}}\right)^{\frac{3}{2}}} \quad ,$$

$$f_d = \left(\frac{e-e_d}{e_c-e_d}\right)^{\alpha_H} = r_e^{\alpha_H} \quad , \qquad f_e = \left(\frac{e_c}{e}\right)^{\beta_H} \quad \text{und}$$

$$f_b = \left(\frac{e_{i0}}{e_{c0}}\right)^{\beta_H}\frac{h_s}{n_H}\frac{1+e_i}{e_i}\left(\frac{-\mathrm{tr}(\mathbf{T})}{h_s}\right)^{1-n_H}\left[3+a^2-a\sqrt{3}\left(\frac{e_{i0}-e_{d0}}{e_{c0}-e_{d0}}\right)^{\alpha_H}\right]^{-1}$$

mit

$$a = \frac{\sqrt{3}\,[3-\sin(\varphi_c)]}{2\sqrt{2}\sin(\varphi_c)} \quad .$$

Für das Referenzmodell nach von Wolffersdorff (1996) ergeben sich durch die Symmetrien sieben unabhängige Werte in $\mathcal{L}$ und drei unabhängige Werte in $\mathbf{N}$ (siehe auch Niemunis, 2003, S. 54).

Die Zusammenhänge zwischen den Porenzahlen ergeben sich nach Bauer (1996) als

$$\frac{e_d}{e_{d0}} = \frac{e_c}{e_{c0}} = \frac{e_i}{e_{i0}} = e^{-\left(\frac{-\operatorname{tr}(\mathbf{T})}{h_s}\right)^{n_H}} \quad , \tag{2.7}$$

also eine Abhängigkeit zum mittleren Druck p und Verringerung mit zunehmendem Druck. Bauer (1996), S. 18, weist darauf hin, dass die Materialparameter e_{d0}, e_{c0} und e_{i0} nicht unbedingt den tatsächlichen (d. h. messbaren) Porenzahlen für verschwindende Kornspannungen $-\operatorname{tr}(\mathbf{T}) \approx 0$ entsprechen.

Konzept der intergranularen Dehnung

Ein Nachteil des Referenzmodells nach von Wolffersdorff (1996) ist die übermäßige Akkumulation der Verformung bei zyklischer Belastung (*„Ratcheting“*). Deshalb wird das Stoffmodell in Niemunis und Herle (1997) um das Konzept der intergranularen Dehnung erweitert. Basierend auf Gleichung (2.4) lautet der Materialtensor $\mathcal{M}$

$$\mathcal{M} = \left[\rho^\chi m_T + (1-\rho^\chi) m_R\right] \mathcal{L} + \begin{cases} \rho^\chi (1-m_T)\mathcal{L} : \overset{\rightarrow}{\mathbf{h}} \otimes \overset{\rightarrow}{\mathbf{h}} + \rho^\chi \mathbf{N} \otimes \overset{\rightarrow}{\mathbf{h}} & \text{für} \quad \overset{\rightarrow}{\mathbf{h}} : \mathbf{D} > 0 \quad , \\ \rho^\chi (m_R - m_T)\mathcal{L} : \overset{\rightarrow}{\mathbf{h}} \otimes \overset{\rightarrow}{\mathbf{h}} & \text{sonst} \quad . \end{cases} \tag{2.8}$$

Folglich hängt der Tensor $\mathcal{M}$ und darüber die Spannungsrate von der Richtung der intergranularen Dehnung $\mathbf{h}$ relativ zum Verformungstensor ab. Ein Interpolationsterm ρ^χ mit

$$\rho = \frac{||\mathbf{h}||}{R_{\max}} \quad \text{mit} \quad 0 < \rho \leq 1 \tag{2.9}$$

und Materialparameter χ erzeugt einen gleichmäßigen Übergang zwischen den möglichen Zuständen in Gleichung (2.8). Mit maximaler intergranularer Dehnung $||\mathbf{h}|| = R_{\max}$ und somit $\rho = 1$ ergeben sich die folgenden drei Fälle (siehe auch Niemunis und Herle, 1997, S. 286):

- $\mathcal{M} = \mathcal{L} + \mathbf{N} \otimes \overset{\rightarrow}{\mathbf{h}}$ für $\mathbf{D} \sim \overset{\rightarrow}{\mathbf{h}}$ (bzw. $\overset{\rightarrow}{\mathbf{h}} : \mathbf{D} > 0$, gleichbleibende Verformung). Da in diesem Fall $\mathbf{D} = \overset{\rightarrow}{\mathbf{h}} \otimes ||\mathbf{D}||$ gilt, ergibt sich aus $\overset{\rightarrow}{\mathbf{h}} \otimes \overset{\rightarrow}{\mathbf{h}} : \mathbf{D} = \mathbf{D}$ und $\mathbf{N} \otimes \overset{\rightarrow}{\mathbf{h}} : \mathbf{D} = \mathbf{N}||\mathbf{D}||$ das hypoplastische Referenzmodell aus Gleichung (2.6) nach von Wolffersdorff (1996),
- $\mathcal{M} = m_R \mathcal{L}$ für $\mathbf{D} \sim -\overset{\rightarrow}{\mathbf{h}}$ (bzw. $\overset{\rightarrow}{\mathbf{h}} : \mathbf{D} < 0$, Verformungsumkehr) und
- $\mathcal{M} = m_T \mathcal{L}$ für $\mathbf{D} \perp \overset{\rightarrow}{\mathbf{h}}$ (bzw. $\overset{\rightarrow}{\mathbf{h}} : \mathbf{D} = 0$).

Die Entwicklung der intergranularen Dehnung selbst folgt

$$\overset{\circ}{\mathbf{h}} = \begin{cases} \left(\mathcal{I} - \vec{\mathbf{h}} \otimes \vec{\mathbf{h}} \rho^{\beta_R}\right) : \mathbf{D} & \text{für} \quad \vec{\mathbf{h}} : \mathbf{D} > 0 \\ \mathbf{D} & \text{sonst} \quad . \end{cases}$$

Insgesamt erfordert das Konzept der intergranularen Dehnung die fünf zusätzlichen Materialparameter m_T, m_R, $R_{\max}$, β_R und χ.

Erweiterungsmöglichkeiten für das hypoplastische Modell und das Konzept der intergranularen Dehnungen (z. B. in Form der Viskohypoplastizität) werden u. a. in Niemunis (2003) diskutiert. Bode u. a. (2020) weisen darauf hin, dass das Konzept der intergranularen Dehnungen für beliebige ratenabhängige Stoffmodelle genutzt werden kann.

Weitere Anpassungen

Ergänzend zum beschriebenen Referenzmodell werden in Niemunis (2003) weitere Anpassungen vorgeschlagen. Im ursprünglichen Modell kann eine isotrope Extension dazu führen, dass die Porenzahl kleiner wird als die untere Grenze e_d (unabhängig von der intergranularen Dehnung) und zu ungültigen Zuständen in einer Simulation führen kann (siehe dazu Niemunis, 2003, S. 172–176). Dort wird die folgende Anpassung von f_d vorgeschlagen, um die Einhaltung der unteren Grenze für $e = e_d$ (auf Kosten möglicher zyklischer Akkumulationseffekte) zu erzwingen:

$$f_d = f_{d,\text{adj}} + [1 - f_{d,\text{adj}}]^z \, \bar{f}_d \quad \text{mit} \quad z = 5$$

mit

$$\bar{f}_d = -\frac{M_e^{(d)} \sqrt{3}(1+e) + M_T^{(d)} f_b f_e \frac{3}{\sqrt{3}}(3+a^2)}{M_T^{(d)} f_b f_e 3a} \quad ,$$

$$M_T^{(d)} = -\frac{e_d \, n_H}{h_s} \left(-\frac{\operatorname{tr}(\mathbf{T})}{h_s}\right)^{n_H - 1} \quad \text{und} \quad M_e^{(d)} = 1$$

sowie

$$f_{d,\text{adj}} = \begin{cases} \left(\frac{e - e_d}{e_c - e_d}\right)^{\alpha_H} & \text{falls} \quad e > e_d \\ -\left(\frac{|e - e_d|}{e_c - e_d}\right)^{\alpha_H} & \text{sonst} \quad . \end{cases} \tag{2.10}$$

Bestimmung der Materialparameter

Die Materialparameter können aus den Daten von zwei eindimensionalen und drei triaxialen Kompressionsversuchen berechnet werden. Dazu werden noch der kritische Reibungswinkel[27], die Korndichte und Angaben zur lockersten und dichtesten Lagerung benötigt. Die folgende Beschreibung orientiert sich an Herle (1997), S. 23–54, Herle und Gudehus (1999) und der Beschreibung im Anhang von Karcher (2003), S. 167–176.

Aus den Ödometerversuchen wird die Abhängigkeit der Spannung σ zur Porenzahl e gemessen. Für die Berechnung der hypoplastischen Parameter wird aus dem Ödometerversuch mit locker gelagerter Bodenprobe der Spannungsbereich ausgewählt, der auch für spätere Versuche maßgebend ist (untere Grenze σ_C und obere Grenze σ_F). In beiden Punkten wird die Tangente der σ-e-Kurve über eine Sekante durch zwei benachbarte Punkte angenähert ($\sigma_A < \sigma_C < \sigma_B$ und $\sigma_D < \sigma_F < \sigma_E$). Die mittlere Spannung p_s an diesen Punkten ist

$$p_{s,i} = \frac{1 + 2\,[1 - \sin(\varphi_c)]}{3}\sigma_i \qquad \text{für } i = A, \ldots, E$$

und dient zur Berechnung der Kompressionsbeiwerte C_{cC} sowie C_{cF}. Mit den entsprechenden Porenzahlen der σ-e-Kurve ergeben sich

$$C_{cC} = \frac{e_{\sigma B} - e_{\sigma A}}{\ln(p_{sA}) - \ln(p_{sB})} \quad \text{und} \qquad C_{cF} = \frac{e_{\sigma E} - e_{\sigma D}}{\ln(p_{sD}) - \ln(p_{sE})} \quad .$$

Anschließend können der Exponent n_H und die Granulathärte h_s bestimmt werden über

$$n_H = \frac{\ln(e_{\sigma C} C_{cF}) - \ln(e_{\sigma F} C_{cC})}{\ln(p_{sF}) - \ln(p_{sC})} \quad \text{und} \qquad h_s = 3 p_{sC} \left(\frac{n_H\, e_{\sigma C}}{C_{cC}} \right)^{\frac{1}{n_H}} \quad .$$

Über das Kompressionsverhältnis von Bauer (1996) aus Gleichung (2.7) kann die Peak-Porenzahl $e_{\text{peak},0}$ im Nullzustand, d. h. bei verschwindendem Korndruck, abgeschätzt werden. Dabei ist

$$p_{\text{peak}} = \sigma_{\text{peak}} \frac{1 + 2\,[1 - \sin(\varphi_c)]}{3}$$

die Maximalspannung eines Triaxialversuchs mit dicht gelagerter Bodenprobe und e_{peak} die dazugehörige Porenzahl. Zur Bestimmung von α_H werden die relative Porenzahl r_e, eine Art druckbezogene Lagerungsdichte, und die Variablen K_p, a und A sowie der Tangens des Dilatanzwinkels ψ_p benötigt, die wie folgt bestimmt werden können

$$r_e = \frac{e_{\text{peak},0} - e_{\min}}{e_{\max} - e_{\min}} \quad , \qquad K_p = \frac{1 + \sin\left(\varphi_{\text{peak}}\right)}{1 - \sin\left(\varphi_{\text{peak}}\right)} \quad ,$$

[27] In Miura u. a. (1997), S. 95, ist der Einfluss unterschiedlicher Faktoren beim Schüttkegelversuch zur Bestimmung des Reibungswinkels untersucht worden. Es wird gezeigt, dass eine kleinere Materialmenge, kleinere Fallgeschwindigkeit und größere Rauigkeit der Unterlage zu einer Erhöhung des gemessenen Reibungswinkels führen können.

$$a = \frac{\sqrt{3}\,[3 - \sin(\varphi_c)]}{2\sqrt{2}\sin(\varphi_c)} \quad \text{und} \qquad A = \frac{a^2}{(2+K_p)^2}\left[1 - \frac{K_p(4-K_p)}{5K_p - 2}\right] \quad .$$

Der Dilatanzwinkel ψ_p kann entweder aus einem Triaxialversuch mit dicht gelagerter Bodenprobe bestimmt werden, oder laut Karcher (2003), S. 173, mit

$$\tan(\psi_p) = 2\frac{K_p - 4 + 5AK_p^2 - 2AK_p}{(5K_p - 2)(1 + 2A)} - 1$$

abgeschätzt werden. Schließlich kann α_H berechnet werden mit

$$\alpha_H = \frac{\ln\left(6\frac{(2+K_p)^2 + a^2 K_p \cdot [K_p - 1 - \tan(\psi_p)]}{a \cdot (2+K_p)(5K_p - 2)\sqrt{4 + 2[1 + \tan(\psi_p)]^2}}\right)}{\ln(r_e)} \quad .$$

Zur Bestimmung von β_H werden die Hilfsgrößen

$$m_i = (2 + K_{0i})^2 + a^2 \quad , \qquad n_i = \frac{a \cdot (2 + K_{0i})(5 - 2K_{0i})}{3} \quad ,$$
$$f_{d,i} = (r_{e,i})^{\alpha_H} \quad \text{und} \qquad E_{s,i} = \frac{1 + e_i}{C_{c,i}}$$

verwendet mit der Abschätzung $K_{0i} = 1 - \sin(\varphi_{\text{peak},i})$ nach Jáky.[28] Damit kann

$$\beta_H = \frac{\ln(E_{s2} \cdot (m_1 - n_1 f_{d1})) - \ln(E_{s1} \cdot (m_2 - n_2 f_{d2}))}{\ln(e_1) - \ln(e_2)}$$

berechnet werden.

Mašín (2019), S. 129 f., empfiehlt, für die Bestimmung der erweiterten hypoplastischen Parameter, $R_{\max} = 10^{-4}$ als Materialkonstante anzunehmen und χ und β_R über Kurvenanpassung zu bestimmen ($R_{\max}$, β_R und χ steuern alle den Verlauf der Steifigkeitskurve). Abgeleitet aus experimentellen Arbeiten von Atkinson u. a. (1990) wird die Approximation $m_T = 0{,}7 m_R$ gegeben.

Für die Bestimmung der anderen Materialparameter geben Herle und Gudehus (1999), S. 474, die Abschätzung $e_{i0} \approx 1{,}2 e_{c0}$ (bzw. $e_{i0} = 1{,}15 e_{c0}$ in Herle, 1997, S. 53). Basierend auf der kritischen Porenzahl e_{c0} kann die Porenzahl bei dichtester Lagerung als $e_{d0} = 0{,}6 e_{c0}$ angenommen werden (Mašín, 2019, S. 98, empfiehlt $e_{d0} = 0{,}5 e_{c0}$).

[28] Á. Kézdi beschreibt 1962 in seinem Buch „Erddrucktheorien“ (S. 37 f.) die Abschätzung $\lambda_0 = 1 - \sin(\rho')$ mit Ruhedruckziffer λ_0 und Reibungswinkel ρ'. Er bezieht sich dabei auf eine Formel aus J. Jáky (1944): „Anyugalmi nyomás tényezöje“.

2.3.4 Barodesie

Nach Kolymbas (2011a), S. 1220, hat die Barodesie einen hypoplastischen Ansatz und basiert auf den zwei Hypothesen aus Goldscheider (1976), S. 465, (siehe Abschnitt 2.2), womit

- proportionale Dehnungspfade ausgehend von $T_i = 0$ entsprechende proportionale Spannungspfade haben und
- proportionale Dehnungspfade ausgehend von einem anderen Spannungszustand sich asymptotisch an die entsprechenden proportionalen Spannungspfade ausgehend von $T_i = 0$ annähern.

In der Barodesie wird eine sog. **R**-Funktion verwendet, die eine exponentielle Abhängigkeit zum Verzerrungstensor beschreibt.[29] Damit kann die erste Hypothese nach Medicus, Fellin und Schranz (2018), S. 624, mit

$$\vec{\mathbf{R}}\left(\vec{\mathbf{D}}\right) = \vec{\mathbf{T}}$$

beschrieben werden und die zweite z. B. aus der geometrischen Betrachtung in Bild 2.9 als

$$\vec{\mathbf{T}} + \eta\dot{\vec{\mathbf{T}}} = \mu\vec{\mathbf{R}}$$

mit zwei skalaren Größen η und μ. Daraus kann das allgemeine Referenzmodel für die Entwicklung der Spannungsrate in der Barodesie

$$\mathring{\mathbf{T}} = h \cdot \left(f\vec{\mathbf{R}} + g\vec{\mathbf{T}}\right) ||\mathbf{D}|| \tag{2.11}$$

mit den drei Größen f, g und h hergeleitet werden.

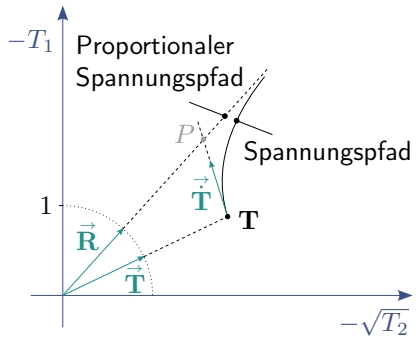


Bild 2.9: Asymptotische Annäherung von einem Spannungszustand $\mathbf{T}$ mit Tangente $\dot{\vec{\mathbf{T}}}$ an einen proportionalen Spannungszustand definiert durch $\vec{\mathbf{R}}$ in der Triaxialebene (nach Schranz, 2018, S. 27; bzw. Medicus, Fellin und Schranz, 2018, S. 627).

[29] Mit der Matrixexponentialfunktion wird eine reelle und symmetrische Matrix wieder auf eine reelle und symmetrische Matrix abgebildet.

2.3.4.1 Barodesie nach Kolymbas (2015)

Die erste Version der Barodesie ist in Kolymbas (2011a) veröffentlicht worden. In Fellin und Ostermann (2013) wird aufgezeigt, dass die Grenzbedingung für dieses Stoffmodell annähernd mit der Grenzbedingung nach Matsuoka-Nakai übereinstimmt. Das ist auch für die überarbeitete Version aus Kolymbas (2015) gültig. Die verwendeten Parameter dieses Stoffmodells sind

$$\mathbf{R} = -e^{\alpha_K \vec{\mathbf{D}}} \quad , \qquad f = \mathrm{tr}(\vec{\mathbf{D}}) + c_3 e_c \quad ,$$
$$g = -c_3 e \quad \text{und} \qquad h = -\frac{c_4 + c_5 ||\mathbf{T}||}{e - e_{\min}}$$

mit

$$\alpha_K = c_1 e^{c_2 \,\mathrm{tr}(\vec{\mathbf{D}})} \quad , \quad e_c = \frac{e_{\min} + B}{1 - B} \quad \text{und} \quad B = \frac{e_{c0} - e_{\min}}{e_{c0} + 1} \cdot \left(\frac{c_4 + c_5 ||\mathbf{T}||}{c_4} \right)^{-\frac{1+e_{\min}}{c_5}}$$

sowie der Konstanten

$$c_1 = \sqrt{\frac{2}{3}} \ln\left(\frac{1 - \sin(\varphi_c)}{1 + \sin(\varphi_c)} \right) \quad .$$

Für das Stoffmodell werden die Materialparameter φ_c, c_2, c_3, c_4, c_5, e_{c0} und $e_{\min}$ benötigt.

Die Bestimmung der Materialparameter basiert teilweise auf gemessenen Werten aus einem Ödometerversuch mit locker gelagerter Bodenprobe und einem undrainierten Triaxialversuch. Nach Kolymbas (2015), S. 56 f., kann der kritische Reibungswinkel φ_c aus einem undrainierten Triaxialversuch bestimmt werden. c_2 wird anhand der Krümmung der Spannungs-Dehnungs-Kurve des Triaxialversuchs abgeschätzt.

c_4 sowie c_5 können mit $\sigma = ||\mathbf{T}||$, $\delta = \mathrm{tr}(\vec{\mathbf{D}})$ und der Approximation

$$\frac{\mathrm{d}e}{\mathrm{d}\sigma} = \frac{1+e}{1 + c_3 \left(\frac{e_c - e}{\delta} \right)} \cdot \frac{1}{h} = \frac{1+e}{1 + c_3 \left(\frac{e_c - e}{\delta} \right)} \cdot \frac{e - e_{\min}}{c_4 + c_5 \sigma}$$
$$\approx (1+e) \, \frac{e - e_{\min}}{c_4 + c_5 \sigma}$$

aus der Spannungs-Porenzahl-Beziehung des Ödometerversuchs abgeleitet werden. Die Bestimmung von Parameter c_3 und die Feinabstimmung von c_2 erfolgt durch Ausprobieren („trial and error“, siehe Kolymbas, 2015, S. 56). e_{c0} kann nach Gleichung (2.7) abgeschätzt werden und $e_{\min}$ ist die dichteste Porenzahl für den betrachteten Spannungszustand.

2.3.4.2 Barodesie nach Schranz (2018)

Diese Version der Barodesie ist in der allgemeinen Form identisch zur Grundform in Gleichung (2.11). Die verwendeten Parameter sind jedoch

$$\mathbf{R} = -e^{\alpha_S \vec{\mathbf{D}}} \quad , \qquad f = c_8 b\delta + c_6 \quad ,$$
$$g = (1 - c_8)\, b\delta + c_5 \left[\left(\frac{1+e}{1+e_c} \right)^{\zeta} - 1 \right] - c_6 \quad \text{und} \qquad h = c_4 ||\mathbf{T}||^{\xi}$$

mit

$$\alpha_S(\delta) = c_{\min} + c_3 \frac{|\delta - \sqrt{2}|^{c_2}}{\left(1 + |\delta - \sqrt{2}|\right)^{c_1}} \quad , \qquad \delta = \mathrm{tr}(\vec{\mathbf{D}}) \quad ,$$
$$e_c = (1 + e_{c0})\, e^{-\frac{p^{(1-\xi)}}{c_4(1-\xi)}} - 1 \quad \text{und} \qquad b = (b_{\text{ext}} - b_{\text{comp}}) \left(\frac{\delta + \sqrt{3}}{2\sqrt{3}} \right)^{c_7} + b_{\text{comp}}$$

und den Konstanten

$$b_{\text{comp}} = \frac{c_5 \cdot \left(c_e^{\zeta} - 1 \right) - 3}{\sqrt{3}} \quad , \qquad b_{\text{ext}} = \frac{c_5 \cdot \left(1 - c_e^{\zeta} \right) - 3\kappa}{\sqrt{3}} \quad ,$$
$$c_1 = \frac{f_1}{f_4} \ln\!\left(\frac{\alpha_p - c_{\min}}{\alpha_c - c_{\min}} \right) - \frac{f_2}{f_4} \ln\!\left(\frac{\alpha_0 - c_{\min}}{\alpha_c - c_{\min}} \right) \quad , \qquad c_2 = \frac{1}{f_1} \left[\ln\!\left(\frac{\alpha_0 - c_{\min}}{\alpha_c - c_{\min}} \right) - f_3 c_1 \right] \quad ,$$
$$c_3 = (\alpha_c - c_{\min}) \frac{\left(1 + \sqrt{2}\right)^{c_1}}{\sqrt{2}^{c_2}} \quad , \qquad c_4 = K_r \left(\sqrt{3} p_r \right)^{-\xi} \quad ,$$

$$c_8 = -\frac{c_6}{\sqrt{3} b_{\text{ext}}} \quad , \qquad \zeta = -\frac{1 + \sin(\varphi_c)}{1 - \sin(\varphi_c)} \quad \text{und} \quad ,$$
$$f_1 = \ln\!\left(\frac{1 + \sqrt{2}}{\sqrt{2}} \right) \quad , \qquad f_2 = \ln\!\left(\frac{3\sqrt{2} - \sqrt{3}}{3\sqrt{2}} \right) \quad ,$$
$$f_3 = \ln\!\left(\frac{1 + \sqrt{2}}{2 + \sqrt{2}} \right) \quad , \qquad f_4 = f_1 \ln\!\left(\frac{3 + 3\sqrt{2}}{3 + 3\sqrt{2} - \sqrt{3}} \right) - f_3 f_2 \quad ,$$
$$\alpha_p = \ln\!\left(\frac{9 - 11\sin(\varphi_c)}{9 + 13\sin(\varphi_c)} \right) \frac{\sqrt{3}}{2} \quad , \qquad \alpha_c = \frac{\sqrt{2} \ln\!\left(\frac{1 - \sin(\varphi_c)}{1 + \sin(\varphi_c)} \right)}{\sqrt{3}} \quad ,$$
$$\alpha_0 = \ln(1 - \sin(\varphi_c)) \quad \text{und} \qquad c_{\min} = -30 \quad .$$

Zusätzlich werden die Materialparameter φ_c, K_r, ξ, κ, e_{c0}, c_e, c_5, c_6 und c_7 für einen Referenzdruck (wie $p_r = 1\,\text{kPa}$) benötigt.

Eine kurze Beschreibung der Materialparameter findet sich in Schranz (2018), S. 64 f.[30] Die Bestimmung der Materialparameter wird dort nicht thematisiert.

[30] In Schranz (2018) gibt es kleine Unterschiede zwischen den Stoffgleichungen auf S. 165 f. und der Implementierung auf S. 173–175 (bspw. ζ und α_p). Die hier vorgestellte Version orientiert sich an der Implementierung und zeigt gute Übereinstimmungen zu den Referenzergebnissen (siehe Kapitel 7).

2.3.4.3 Barodesie nach Kolymbas (2021)

Auch die überarbeitete Version aus Kolymbas (2021) ist in der allgemeinen Form identisch zu Gleichung (2.11) mit folgenden Parametern:

$$\mathbf{R}(\mathbf{D}) = -\exp\left(c_1\vec{\mathbf{D}}\right) + c_2\delta\mathbf{1} \quad , \qquad f = e_c + c_3\delta \quad ,$$
$$g = -e + c_3\delta \quad \text{und} \qquad h = c_4||\mathbf{T}||^{c_5}$$

mit

$$\begin{aligned} \epsilon_1(p) &= k_1 - k_2\ln(p) \quad , \\ \epsilon_2(p) &= \epsilon_1(p)\cdot(1+\kappa_2\delta) \quad \text{und} \\ \dot{e}_c &= \kappa_1\cdot(\epsilon_2(p)-e_c)\cdot||\mathbf{D}|| + (e_{c0}-e_c)\cdot\frac{\dot{\lambda}_D}{2} \quad , \end{aligned} \tag{2.12}$$

sowie der Konstanten

$$c_1 = \sqrt{\frac{2}{3}}\ln\left(\frac{1-\sin(\varphi_c)}{1+\sin(\varphi_c)}\right) \quad ,$$

den Termen

$$\delta = \mathrm{tr}(\vec{\mathbf{D}}) \quad , \qquad p = -\frac{\mathrm{tr}(\mathbf{T})}{3} \quad \text{und} \qquad \lambda_D = ||\vec{\mathbf{D}}_i - \vec{\mathbf{D}}_{i-1}||$$

und den Parametern φ_c, c_2, c_3, c_4, c_5, κ_1, κ_2, k_1, k_2 und e_{c0}.

Eine Besonderheit dieser Formulierung ist Gleichung (2.12), die sich aus zwei Teilen zusammensetzt: Der erste Teil hängt von der Verzerrungsrate $\mathbf{D}$ ab, der zweite von der Differenz der Verzerrungsrate λ_D im aktuellen Schritt gegenüber der aus dem vorherigen Schritt. Dies muss bei der Integration beachtet werden.

Für die Bestimmung der Materialparameter wird in Kolymbas (2022), S. 122 f., auf die Kalibrierung von numerischen Nachrechnungen ausgehend von einem Referenzdatensatz verwiesen, der in Abschnitt 6.4 aufgelistet ist (Tabelle 6.12 auf S. 125).

2.4 Integrationsverfahren

Man hüte sich, auf Grund einzelner Beispiele allgemeine Schlüsse über den Wert oder Unwert einer Methode zu ziehen. Dazu gehört sehr viel Erfahrung.

Collatz (1950), S. 1

Um die Spannungsantwort von Stoffmodellen zu berechnen, die ratenabhängig formuliert sind, ist eine Integration über das betrachtete Zeitintervall notwendig. Für die numerische Integration von Differentialgleichungen wie in einem Stoffmodell gibt es zahlreiche Verfahren, die sich je nach Art und Eigenschaften der Differentialgleichungen mehr oder weniger eignen.[31] Die folgende Übersicht orientiert sich maßgeblich an Hairer, Nørsett u. a. (2008).

Im Gegensatz zur symbolischen Integration wird bei der numerischen Integration keine Stammfunktion oder kontinuierliche Lösung ermittelt, sondern das Integral an einzelnen, diskreten Stellen ausgewertet. Die allgemeine Form lautet

$$y_{n+1} = y_n + \int_n^{n+1} \dot{y}(y,t)\,\mathrm{d}t \quad ,$$

um das Integral y_{n+1} zum Zeitpunkt $n+1$ basierend auf dem aktuellen Funktionswert y_n und den Differentialgleichungen $\dot{y}(y,t)$ auszuwerten. Eine mögliche Vorgehensweise ist die Approximation der Lösung y_{n+1} durch eine abgeschnittene Taylorreihe mit

$$\hat{y}_{n+1} = y_n + (t_{n+1} - t_n)\,\dot{y}_n = y_n + h\dot{y}_n \quad . \tag{2.13}$$

Gleichung (2.13) wird als *explizites Verfahren* bezeichnet, da für die Berechnung des nächsten Funktionswertes $\hat{y}_{n+1}$ nur Informationen aus dem aktuellen Zeitschritt n verwendet werden. $\dot{y}_n$ symbolisiert, dass die Differentialgleichung $\dot{y}(y,t)$ zum Zeitpunkt n ausgewertet wird. Im Gegensatz dazu nutzen *(semi-)implizite Verfahren* Informationen aus dem zu berechnenden Zeitschritt $n+1$. Das implizite Euler-Verfahren hat die Form

$$\hat{y}_{n+1} = y_n + h\dot{y}_{n+1} \quad . \tag{2.14}$$

Durch den Term $\dot{y}_{n+1}$ auf der rechten Seite ist die iterative Lösung eines nichtlinearen Gleichungssystems erforderlich.

Das Ergebnis einer numerischen Integration ist im Allgemeinen fehlerbehaftet und Integrationsfehler können sich insbesondere über viele Zeitschritte aufsummieren (siehe auch Anhang C). Kleinere Zeitschritte können zwar die Genauigkeit erhöhen, aber nicht die Konvergenzgeschwindigkeit des Verfahrens.

[31] Eine anschauliche Einstufung verschiedener Integrationsverfahren findet sich in Butcher (2016), S. 95.

2.4.1 Explizite Runge-Kutta-Verfahren

Eine höhere Konvergenzgeschwindigkeit kann bspw. durch Integrationsverfahren höherer Ordnung erreicht werden. Es ist wichtig zu beachten, dass eine hohe Ordnung des Integrationsverfahrens nicht das Gleiche wie eine hohe Genauigkeit der Lösung ist (siehe Press u. a., 1997, S. 140 und S. 705 f.).

In dieser Arbeit werden vor allem explizite *Runge-Kutta-Verfahren* betrachtet (siehe z. B. Hairer, Nørsett u. a., 2008, S. 129–140).[32] Runge-Kutta-Verfahren zählen zu den Einschrittverfahren, da nur Werte aus dem aktuellen Zeitschritt zur Abschätzung des nächsten Zeitschritts genutzt werden. Bei diesen Verfahren wird die Differentialgleichung an Zwischenstellen ausgewertet und diese Teillösungen werden mit geeigneten Gewichtungsfaktoren kombiniert. Für ein explizites Runge-Kutta-Verfahren mit s Stufen gilt

$$\begin{aligned} k_1 &= \dot{y}(t_n, y_n) \\ k_2 &= \dot{y}(t_n + c_2 h, y_n + h a_{21} k_1) \\ k_3 &= \dot{y}(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)) \\ &\dots \\ k_s &= \dot{y}(t_n + c_s h, y_n + h(a_{s1} k_1 + \dots + a_{s,s-1} k_{s-1})) \\ \hat{y}_{n+1} &= y_n + h(b_1 k_1 + b_2 k_2 + \dots + b_s k_s) \quad . \end{aligned} \tag{2.15}$$

bzw. in Form eines sog. Butcher-Tableaus

$$\begin{array}{c|cccccc} 0 & & & & & & \\ c_2 & a_{2,1} & & & & & \\ c_3 & a_{3,1} & a_{3,2} & & & & \\ \vdots & \vdots & \vdots & \ddots & & & \\ c_s & a_{s,1} & a_{s,2} & \cdots & a_{s,s-1} & & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array} \quad . \tag{2.16}$$

Der globale Integrationsfehler ϵ eines Verfahrens der Ordnung $p \leq s$ ist für ausreichend glatte Probleme durch $\epsilon \leq K h^{p+1}$ mit einer Konstanten K beschränkt.[33] Dadurch können mit einem Verfahren der Ordnung p polynomielle Gleichungen bis zu dieser Ordnung exakt integriert werden (siehe Hairer, Nørsett u. a., 2008, S. 134).

Eine Möglichkeit zur Abschätzung des Integrationsfehlers ist die Anwendung der *Richardson-Extrapolation*. Dabei wird ein Integral mit dem gleichen Verfahren einmal mit der vorgegebenen Schrittweite h und einmal mit zwei Teilschritten der Schrittweite $h/2$ ausgewertet.

[32] Neben den Referenzen Hairer, Nørsett u. a. (2008) und Hairer und Wanner (2002) ist auch Butcher (2016) für die Diskussion von expliziten Runge-Kutta-Methoden empfehlenswert. Geometrische Beispiele finden sich in Hairer, Lubich u. a. (2006) und praktische Tipps zur Implementierung in Press u. a. (1997).

[33] Der tatsächliche Integrationsfehler und die exakte Lösung des Integrals sind im Allgemeinen nicht bekannt.

Aus beiden Abschätzungen (und der Ordnung des Verfahrens) kann eine Abschätzung für den Integrationsfehler angegeben werden (siehe Hairer, Nørsett u. a., 2008, S. 164 f.).

Eine andere Herangehensweise zur Abschätzung des Integrationsfehlers sind *eingebettete Runge-Kutta-Verfahren*. Für die Abschätzung eines Integrals mit einem Verfahren der Ordnung $p > 1$ müssen $s \geq p$ aufeinander folgende Stufen ausgewertet werden. Durch geschickte Konstruktion eines Verfahrens ist es möglich, dass aus dem Zwischenergebnis der ersten $s^* < s$ Stufen ein Verfahren der Ordnung $p^* < p$ (oftmals $p^* = p - 1$) gebildet werden kann (für Details siehe bspw. Butcher, 2016, S. 214–226).

Beide Verfahren liefern eine Approximation für das Integral und aus der Differenz davon $|\hat{y}_{n+1} - \hat{y}_{p,n+1}|$ kann der Integrationsfehler abgeschätzt werden. Da sich die Abschätzung des Integrationsfehlers (aus den beiden Auswertungen) auf das Verfahren mit der geringeren Ordnung bezieht, ist als Ergebnis die Abschätzung der geringeren Ordnung zu verwenden (siehe Press u. a., 1997, S. 708 f.).

Ein bekanntes Verfahren der Ordnung 2/3 ist das Verfahren von Bogacki-Shampine, das auch in MATLAB als `ode23` integriert ist:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{3}{4} & 0 & \frac{3}{4} & & \\ 1 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \\ \hline & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \\ & \frac{7}{24} & \frac{1}{4} & \frac{1}{3} & \frac{1}{8} \end{array} . \tag{2.17}$$

Dabei werden mit höchstens vier Funktionsauswertungen Zwischenwerte an den Stellen 0, 1/2, 3/4 und 1 bestimmt. Die unteren beiden Zeilen im Tableau von Gleichung (2.17) definieren, dass das Integral mit den berechneten Zwischenwerten an zwei Stellen ausgewertet wird: in der vorletzten Zeile durch ein Verfahren zweiter Ordnung und in der letzten Zeile durch ein Verfahren dritter Ordnung zur Fehlerschätzung.

Am Beispiel des Verfahrens von Bogacki-Shampine kann auch die *FSAL*-Eigenschaft („*First State As Last*“) gesehen werden. Wenn als letzte Zwischenstufe des Verfahrens $\dot{y}(t_n + h, y_{n+1})$ ausgewertet wird (d. h. $c_s = 1$ und $a_{s,i} = b_i$ im Butcher-Tableau), dann kann die Stufe nicht nur für die aktuelle Abschätzung genutzt werden, sondern ist gleichzeitig auch Startwert im nächsten Zeitschritt. Für jeden akzeptierten Schritt ist somit eine Auswertung weniger erforderlich (siehe Hairer, Nørsett u. a., 2008, S. 167).

2.4.2 Schrittweite und Fehlerabschätzung

Ein geeignetes Verfahren zeichnet sich dadurch aus, dass es

- eine gute Balance zwischen Stabilität des Verfahrens und Aufwand der Berechnung bietet,
- eine Abschätzung der Genauigkeit des Ergebnisses ermöglicht und
- den Zeitschritt adaptiv anpassen kann, um die gewünschte Genauigkeit zu erreichen.

Wenn der Integrationsfehler (bzw. eine Abschätzung davon) bekannt und zulässige Toleranzen definiert sind, kann automatisch entschieden werden, ob die ermittelte Lösung des Integrals akzeptiert wird oder die Berechnung mit einem kleineren Zeitschritt wiederholt werden muss. Darüber hinaus kann in beiden Fällen eine neue Schrittweite berechnet werden. Verfahren, die eine derartige automatische Schrittweiten-Kontrolle ermöglichen, werden als *adaptive Verfahren* bezeichnet. Eine Empfehlung zur Berechnung der Schrittweite basierend auf dem abgeschätzten Fehler wird z. B. in Hairer, Nørsett u. a. (2008), S. 168, oder Butcher (2016), S. 290, gegeben. Der Faktor r_{new} zum Skalieren der aktuellen Schrittweite lautet

$$r_{\text{new}} = \min\Big(f_{\max},\ \max\Big(f_{\min},\ S \cdot \hat{f}^{\frac{1}{p+1}}\Big)\Big) \tag{2.18}$$

mit dem empfohlenen Sicherheitsfaktor $S = 0{,}9$, der Ordnung des Verfahrens p sowie dem minimalen und maximalen Faktor zur Vergrößerung ($f_{\max}$) und Verkleinerung ($f_{\min}$) der Schrittweite. Der relative Fehler $\hat{f}$ wird aus den aktuellen Abschätzungen und vorgegebenen Toleranzen berechnet, wobei ein Berechnungsschritt für $\hat{f} \leq 1$ akzeptiert und ansonsten abgelehnt wird. In Press u. a. (1997), S. 712, wird empfohlen, den Exponent in Gleichung (2.18) abhängig von einer Vergrößerung oder Verkleinerung der Schrittweiten unterschiedlich zu wählen. Dargestellt ist der Exponent $1/(p+1)$ für eine Vergrößerung, bei einer Verkleinerung sollte $1/p$ gewählt werden.

Die Faktoren $f_{\min}$ und $f_{\max}$ werden in Press u. a. (1997) abhängig von der Ordnung des Verfahrens gewählt. Butcher (2016), S. 66, schlägt $f_{\max} = 2$ und $f_{\min} = 0{,}5$ für das maximale bzw. minimale Verhältnis aufeinander folgender Schrittgrößen vor. Für den relativen Fehler $\hat{f}$ wird eine Fehlernorm verwendet, bspw.

$$\begin{aligned} \hat{f}_P &= \frac{1}{t_R} \max\left(\frac{|\hat{y}_{n+1,i} - \hat{y}_{p,n+1,i}|}{|y_{n,i}| + |h\dot{y}(t_n, y_n)| + \epsilon_f}\right) \quad \text{oder} \\ \hat{f}_I &= \max\left(\frac{|\hat{y}_{n+1,i} - \hat{y}_{p,n+1,i}|}{t_A + t_R \max\big(|y_n|, |2\hat{y}_{n+1,i} - \hat{y}_{p,n+1,i}|\big)}\right) \end{aligned} \tag{2.19}$$

mit einem Wert $0 < \epsilon_f \ll 1$, der relativen Toleranz t_R und der absoluten Toleranz t_A.[34] Die Fehlernorm $\hat{f}_I$ wird z. B. in Fellin, Mittendorfer u. a. (2009), S. 700, beschrieben und in der Benutzerroutine `ABQS02` für das explizite Euler-Verfahren mit Richardson-Extrapolation verwendet (siehe Abschnitt 3.1).

[34] Die Toleranzen können auch für jeden Eintrag in $\hat{\boldsymbol{y}}$ unterschiedlich definiert sein.

In dieser Arbeit kommt die Fehlernorm $\hat{f}_P$ zum Einsatz, die in Press u. a. (1997), S. 711–715, für Runge-Kutta-Verfahren diskutiert und dort beispielhaft implementiert wird. Außerdem sind das explizite Euler-Verfahren ohne und mit Richardson-Extrapolation, zwei Runge-Kutta-2/3-Verfahren (nach Bogacki-Shampine und in Form der Simpsonregel), sowie drei Runge-Kutta-4/5-Verfahren (mit Parametern nach Fehlberg, Cash-Karp und Dormand-Prince) in der xMat integriert. Die jeweiligen Anpassungen sind im Rahmen der Implementierung in Abschnitt 4.2.3 beschrieben (sowie in Anhang C).

Explizite Verfahren höherer Ordnung mit Schrittweiten-Kontrolle für hypoplastische Stoffmodelle sind bspw. in Tamagnini, Viggiani, Chambon und Desrues (2000) zum Einsatz gekommen. In der Benutzerroutine `PLX09` ist die Simpsonregel (Runge-Kutta 2/3) implementiert. Halilovič u. a. (2009) und Vrh u. a. (2010) diskutieren eine verbesserte explizite Integration für nichtlineare Stoffmodelle für die Benutzung in Abaqus.

2.4.3 Implizite Verfahren

Bei manchen Differentialgleichungssystemen führt die numerische Integration mit expliziten Verfahren zu Oszillationen, geringen Schrittweiten (bei adaptiver Regelung) und schlichtweg zu einer problematischen oder ineffizienten Berechnung. Diese sog. steifen Probleme können üblicherweise besser mit impliziten Verfahren berechnet werden.[35] Ein Indikator für steife Probleme sind die Eigenwerte aus der Jacobi-Matrix des Differentialgleichungssystems (siehe u. a. Butcher, 2016, S. 28).

Im Rahmen dieser Arbeit werden implizite Integrationsverfahren nicht näher betrachtet. Es sei darauf hingewiesen, dass Heeres und de Borst (2000) das implizite Euler-Verfahren ohne Schrittweiten-Kontrolle für hypoplastische Stoffmodelle thematisieren und das implizite Euler-Verfahren ein wählbares Integrationsverfahren in der Benutzerroutine `ABQS02` ist. Für Hypoplastizität nach Wu (1992) wird eine Implementierung des sog. semi-impliziten Rosenbrock-Wanner-Verfahrens in Batista u. a. (2003) aufgezeigt.[36]

2.4.4 Gleitkommazahlen und Rundungsfehler

Wie bereits angesprochen wird ein Abschätzungsfehler durch eine Approximation der tatsächlichen Lösung erzeugt. Zusätzlich kann sich eine große Anzahl an Iterationen bzw. Rechenoperationen negativ auf den Gesamtfehler auswirken, da Ergebnisse als Gleitkommazahlen mit endlicher Genauigkeit gespeichert werden. Der heute fast universal genutzte Standard IEEE Std 754:2019 beschreibt, wie reelle Zahlen als Gleitkommazahlen dargestellt werden können und welche Anforderungen Rechenoperationen zwischen Gleitkommazahlen erfüllen müssen.

[35] Zur Stabilität impliziter Methoden siehe bspw. Hairer und Wanner (2002), S. 40–49. Für die Stabilitätsregion expliziter Runge-Kutta-Verfahren sei auf Butcher (2016), S. 106 f., verwiesen.

[36] Zum Rosenbrock-Wanner-Verfahrens siehe bspw. Hairer und Wanner (2002), S. 102–116. Die Idee ist ursprünglich in Rosenbrock (1963) publiziert worden (siehe auch Lubich und Roche, 1990).

Gleitkommazahlen werden in der Form $\pm m \cdot \beta^{x-p}$ mit der Basis $\beta = 2$ (oder $\beta = 10$), einer Mantisse m, des Exponenten x und der Präzision p dargestellt. Die Mantisse $0 \leq m \leq \beta^p - 1$ ist normiert, damit eine eindeutige Repräsentation möglich ist.[37]

Da Gleitkommazahlen nur eine endliche Teilmenge der Zahlen abbilden können, ist eine beliebige reelle Zahl im Allgemeinen nicht exakt als Gleitkommazahl darstellbar und muss zur nächsten darstellbaren Gleitkommazahl gerundet werden (siehe auch Goldberg, 1991, S. 6 f., oder Severance und Dowd, 2010, S. 33). Da Gleitkommazahlen nicht in gleichen Abständen voneinander entfernt sind, ist der Rundungsfehler unterschiedlich groß (siehe bspw. Higham, 2002, S. 36). Mit dem Abstand zwischen 1 und der nächstgrößeren Gleitkommazahl als *Maschinengenauigkeit* $\epsilon_M = \beta^{1-p}$, schwankt der Abstand einer normalisierten Gleitkommazahl z und einer angrenzenden zwischen $\beta^{-1}\epsilon_M|z|$ und $\epsilon_M|z|$ (Higham, 2002, S. 38). Der maximale Rundungsfehler durch die Repräsentation einer Zahl als Gleitkommazahl wird als *Rundungseinheit* betrachtet und beträgt $u = 0{,}5\epsilon_M$.

Die Größe von u bzw. ϵ_M hängt von der Genauigkeit der verwendeten Repräsentation von Gleitkommazahlen ab, also der Größe von m und p (d. h. wie viel Speicher jeweils verwendet wird, siehe dazu bspw. Severance und Dowd, 2010, S. 38 f.). Der Zielkonflikt bei der Repräsentation von Gleitkommazahlen besteht in der Abwägung zwischen Genauigkeit, Speicherbedarf und Zugriffs- bzw. Verarbeitungsgeschwindigkeit (siehe Severance und Dowd, 2010, S. 29). Zwei verbreitete Repräsentationen sind Gleitkommazahlen mit *einfacher Genauigkeit* (*„simple precision“*) und *doppelter Genauigkeit* (*„double precision“*). Bei der einfachen Genauigkeit werden Gleitkommazahlen mit 32 Bits dargestellt. Damit ist in der Zehnerbasis eine Repräsentationsgenauigkeit von 6 bis 9 Nachkommastellen möglich ($\epsilon \leq 10^{-6}$). Gleitkommazahlen mit doppelter Genauigkeit nutzen 64 Bits (53 Bits für Mantisse und 11 Bits für die Potenz). Dadurch steigt die Repräsentationsgenauigkeit in der Zehnerbasis auf 15 bis 17 Nachkommastellen ($\epsilon \leq 10^{-15}$). Tabelle 2.3 stellt die Repräsentation mit einfacher und doppelter Genauigkeit gegenüber.

Unabhängig von der gewählten Genauigkeit fordert IEEE Std 754:2019, S. 27, dass das Ergebnis einer Rechenoperation von zwei Gleitkommazahlen dem gerundeten exakten Ergebnis der beiden Gleitkommazahlen bei unendlicher Genauigkeit entsprechen muss. Nach Knuth (1998), S. 229, werden Additionen und Subtraktionen zwar mit einer ausgezeichneten Genauigkeit durchgeführt, können aber vor allem bei der Berechnung der Differenz von zwei fast identischen Werten zu einem substanziellen Genauigkeitsverlust führen (*„catastrophic cancellation“*).

Tabelle 2.3: Parameter zur Repräsentation einer Gleitkommazahl $\pm m \cdot \beta^{x-p}$ mit $x_{\min} \leq x \leq x_{\max}$ und maximalen Rundungsfehler $u = 0{,}5 \cdot \beta^{1-p}$ aus Goldberg (1991), S. 18.

Bezeichnung	β	p	$x_{\min}$	$x_{\max}$	u
Einfache Genauigkeit	2	24	-126	127	$\approx 5{,}96 \cdot 10^{-8}$
Doppelte Genauigkeit	2	53	-1022	1023	$\approx 1{,}11 \cdot 10^{-16}$

[37] Die Darstellung gilt für alle Zahlen außer null und besondere Ausdrücke wie $\pm\infty$ und *NaN*.

Außerdem muss beachtet werden, dass das Assoziativgesetz und das Distributivgesetz bei Gleitkommazahlen nicht mehr allgemeingültig ist (siehe bspw. Goldberg, 1991, S. 30; oder Knuth, 1998, S. 229–231). Mit $\oplus$ und $\odot$ als Notation für die mit einem Rundungsfehler behafteten Versionen von Addition und Multiplikation heißt das

$$(u \oplus v) \oplus w \neq u \oplus (v \oplus w) \qquad \text{und}$$
$$u \odot (v \oplus w) \neq (u \odot v) \oplus (u \odot w)$$

für viele Gleitkommazahlen u, v und w.

3 Ausgangssituation

In diesem Kapitel werden bestehende Benutzerroutinen untersucht und Überlegungen angestellt, wie sich der Aufwand unterschiedlicher Implementierungen vereinfachen lassen könnte. Abschnitt 3.1 gibt einen Überblick über Benutzerroutinen mit dem hypoplastischen Stoffmodell nach von Wolffersdorff (1996), die für numerische Simulationen zum Einsatz kommen und dem Autor zur Verfügung gestanden sind. Der Fokus liegt auf der Beschreibung der Definitionen (Übergabewerte und interne Parameter) sowie die Unterschiede zwischen verschiedenen Implementierungen. Anschließend werden in Abschnitt 3.2 allgemeine Überlegungen für die Neuimplementierung einer Benutzerroutine bzw. Containerstruktur für Benutzerroutinen zusammengefasst.

Wichtige Erkenntnisse aus diesem Kapitel sind:

Übersicht Benutzerroutinen

- Die gleichen Stoffmodelle sind im Allgemeinen für verschiedene Programme in verschiedenen Benutzerroutinen implementiert.
- Um verschiedene Implementierungen miteinander vergleichen zu können, sind mehrere Anpassungen nach intensiver Einarbeitung in die Benutzerroutinen erforderlich.

Implementierung von Benutzerroutinen

- Für Benutzerroutinen mit geotechnischen Stoffmodellen bietet sich ein modernes Fortran als Programmiersprache an.
- Neben der Ausführungsgeschwindigkeit sind auch Lesbarkeit und Struktur wichtige Charakteristiken einer Benutzerroutine.

3.1 Benutzerroutinen

The next time, you write a line of code, remember you are an author, writing for readers who will judge your effort.

McConnell (2004), S. 13

Benutzerroutinen stellen zusätzliche Funktionalitäten für andere Programme bereit (bspw. Materialmodelle für Abaqus oder Plaxis). In dieser Arbeit wird mit einer Benutzerroutine eine Datei bezeichnet, die bei einem Aufruf das Spannungs-Dehnungs-Verhalten eines Materials nach einem implementierten Stoffmodell für das aufrufende Programm berechnet (siehe Bild 3.1). Für ratenabhängige Stoffmodelle, wie Hypoplastizität und Barodesie, muss eine Integration innerhalb der Benutzerroutine durchgeführt werden.

Eine Benutzerroutine benötigt alle relevanten Informationen aus dem aktuellen Zeitschritt, um auf Basis des Stoffmodells und der Übergabewerte eine Reaktion bzw. Antwort des Modells für den nächsten Zeitschritt zu berechnen. Das heißt,

- bei jedem Aufruf müssen selbst für einfache Stoffmodelle Spannung, Dehnung sowie Spannungs- oder Dehnungsänderung und Länge des Zeitinkrements übergeben werden,
- Materialparameter, Statusvariablen und ggf. einstellbare Parameter der Benutzerroutine müssen ebenfalls nach jedem Aufruf vorhanden sein. Auch wenn zumeist eine Übergabe vom aufrufenden Programm erfolgt, könnten diese Daten in der Benutzerroutine definiert bzw. verwaltet werden und
- Deformationsgradient, Starrkörperrotation, aktueller Zeitpunkt und weitere Variablen, wie Temperatur oder Punktkoordinaten, können je nach Stoffmodell erforderlich sein.

Die Antwort eines Stoffmodells besteht zumindest aus der neu berechneten Spannung. Sofern im aufrufenden Programm eine Gleichgewichtsiteration durchgeführt wird – bspw. mit dem Newton-Raphson-Verfahren – werden außerdem die Änderungen der Ausgangsgrößen bzgl. der Eingangsgrößen benötigt. Bei vorgegebener Dehnungsänderung enthält dieser Tensor die partiellen Ableitungen der Spannungsinkremente bezüglich der Dehnungsinkremente und wird im Folgenden als tangentialer Steifigkeitstensor oder *Jacobi-Matrix* $\mathcal{J}$ bezeichnet

$$\mathcal{J} = \frac{\partial \Delta \mathbf{T}}{\partial \Delta \boldsymbol{\varepsilon}} \quad . \tag{3.1}$$

Zusätzlich können Statusvariablen, wie die Porenzahl, und teilweise auch Steuerungsparameter, wie die empfohlene Schrittlänge, für den nächsten Zeitschritt an das aufrufende Programm zurückgegeben werden.

Benutzerroutinen regeln diesen Ablauf, d. h. sie weisen die Übergabewerte internen Variablen zu, lassen die Materialantwort von Stoffmodell und Integrationsverfahren berechnen und geben das Ergebnis (ggf. mit zusätzlichen Statuswerten) an das aufrufende Programm zurück. Für diese Arbeit liegt der Fokus vor allem auf den in Tabelle 3.1 aufgelisteten Benutzerroutinen, die dem Autor für Simulationen zur Verfügung standen.[1]

[1] Für die nicht öffentlichen Benutzerroutinen werden zwar die Eigenschaften diskutiert, aber weder Code noch Zeilennummern angegeben.

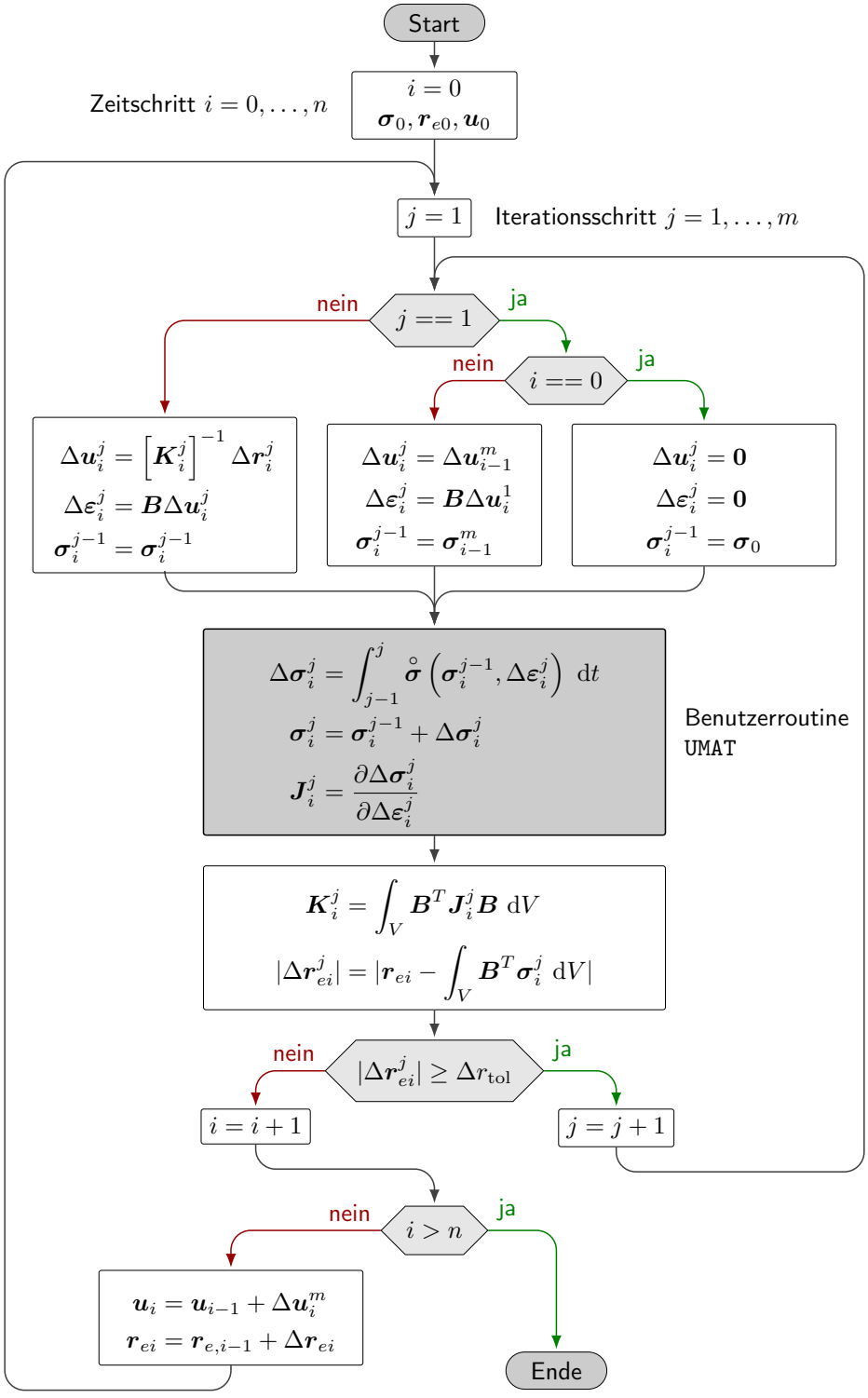


Bild 3.1: Flussdiagramm für die Lösung nichtlinearer Randwertprobleme (in Abaqus) mit dem Newton-Raphson-Verfahren (nach Hügel, 1995, S. 49). Die Benutzerroutine (hier: UMAT) ist dabei nur ein Teilschritt der Gleichgewichtsiteration j in jedem betrachteten Zeitschritt i. Dargestellt sind u. a. die Spannungen $\boldsymbol{\sigma}$, Dehnungsinkremente $\Delta\boldsymbol{\varepsilon}$, Jacobi-Matrix $\boldsymbol{J}$, Residuen $\boldsymbol{r}$ und Verschiebungen $\boldsymbol{u}$.

Tabelle 3.1: Übersicht verschiedener Implementierungen des hypoplastischen Stoffmodells nach von Wolffersdorff (1996), die dem Autor zur Verfügung standen. Die Kürzel basieren auf der Schnittstelle und dem Jahr der Veröffentlichung. Außerdem werden die Abkürzungen `Hypo-VW96` für die Hypoplastizität nach von Wolffersdorff (1996) mit dem Konzept der intergranularen Dehnung nach Niemunis und Herle (1997) und `ViHy-Ni03` für die Viskohypoplastizität nach Niemunis (2003) verwendet. In den Benutzerroutinen `ABQS09` und `ABQE10` weichen einzelne Definitionen des Stoffmodells von der Referenz `Hypo-VW96` ab und sind deshalb mit * gekennzeichnet. Das Ersatzmodell bei `ABQS02` (†) ist nur für geringe Druckspannungen.

Kürzel	`ABQS02`	`PLX09`	`ABQS09`	`ABQE10`	`MAT04`	`ABQS04`
Autor(en)	W. Fellin und A. Ostermann	C. Tamagnini, E. Sellari, D. Mašín, P. A. von Wolffersdorff	A. Niemunis	A. Niemunis (mod.: G. Qiu)	K. Nübel	K. Nübel (mod.: M. Kelm, K.-P. Mahutka)
Jahr	2002	2009	2009	2010	2004	2004
Verfügbarkeit	»https://soilmodels.com/	»https://soilmodels.com/	nicht öffentlich	nicht öffentlich	nicht öffentlich	nicht öffentlich
Schnittstelle	Abaqus/Std.	Plaxis, Abaqus/Std.	Abaqus/Std.	Abaqus/Exp.	MATLAB	Abaqus/Std.
Integrationsverf.	Euler-Explizit, Euler-Implizit und Richardson-Extrapolation	Simpsonregel	Euler-Explizit	Euler-Explizit	Euler-Explizit	Euler-Explizit
Stoffmodell	`Hypo-VW96`	`Hypo-VW96`, Hypo. für Ton	`Hypo-VW96`*, `ViHy-Ni03`	`Hypo-VW96`*, `ViHy-Ni03`	`Hypo-VW96`	`Hypo-VW96`
Intergran. Dehnung	für `statev(5) > 0`	für $m_R > 0{,}5$	für $m_R \geq 2$	für $m_R \geq 2$	für $m_R \geq 2$	ja
Erhöhte Schubst.	nein	nein	ja	ja	nein	nein
Angepasstes f_d	nein	nein	ja	ja	nein	nein
Ersatzmodell (Zugsp.)	nein†	nein	ja	ja	ja	ja
Interne Tensorrepr.	Tensor	Matrix	Tensor	Tensor	Tensor	Tensor

Alle sechs Benutzerroutinen aus Tabelle 3.1 implementieren die Hypoplastizität nach von Wolffersdorff (1996) mit dem Konzept der intergranularen Dehnungen nach Niemunis und Herle (1997). Obwohl diese Benutzerroutinen im Kern das gleiche Stoffmodell enthalten, unterscheiden sich alle in mindestens einem aufgelisteten Merkmal voneinander. Es erfordert einige Einarbeitung bis diese Unterschiede erkannt und beurteilt werden können, da die Implementierungen aufgrund verschiedener Vorgehensweisen und Programmierstile unterschiedlich sind. Vier der sechs Benutzerroutinen werden in Abschnitt 7 für Nachrechnungen verwendet und im Folgenden etwas näher betrachtet.

Die Benutzerroutinen `ABQS02`, `ABQS04` und `ABQS09` sind für die Nutzung aus Abaqus/Standard gedacht und stellen eine `UMAT`-Schnittstelle bereit. Ebenso kann auch die Benutzerroutine `PLX09` über eine `UMAT`-Schnittstelle genutzt werden, ist aber für eine Nutzung aus Plaxis über die `USER_MOD`-Schnittstelle optimiert. `ABQE10` kann über die `VUMAT`-Schnittstelle aus Abaqus/Explizit genutzt werden. Während die bisher aufgelisteten Benutzerroutinen in Fortran programmiert sind, liegt `MAT04` als MATLAB-Skript für die Nutzung aus MATLAB vor.

Beim Aufruf der `UMAT`-Schnittstelle einer Benutzerroutine werden Materialparameter und Konstanten in `props` übergeben und Statusvariablen in `statev`. Für die Initialisierung von Statusvariablen einer Abaqus-Simulation muss entweder in der Input-Datei eine Anfangslösung (`*Initial Conditions, type=SOLUTION`) definiert werden, oder mit der Abaqus-Hilfsfunktion `SDVIni()`.

3.1.1 Verwendete Benutzerroutinen

ABQS02

Die Benutzerroutine von W. Fellin und A. Ostermann enthält eine `UMAT`-Schnittstelle für die Nutzung aus Abaqus.[2]

In `props` wird neben den Materialparametern für das hypoplastische Modell ein hydrostatischer Druck p_c erwartet (Korrektur für kohäsiven Anteil), der als $p_c\delta_{ij}$ vom Spannungstensor in `rateT()` abgezogen wird (Zeile 800–802 in der Benutzerroutine). Die Reihenfolge der tensoriellen Statusvariablen kann aus der Nutzung von `RotSig()` in Zeile 1201 abgeleitet werden. Neben dieser Funktion wird auch die externe Funktion `Xit()` genutzt, die beide von Abaqus bereitgestellt werden (oder für eine Nutzung außerhalb von Abaqus definiert werden müssen, siehe Tabelle 4.3).

Bei der numerischen Integration mit dem expliziten Euler-Verfahren mit Richardson-Extrapolation wird die nächste Schrittlänge nach Gleichung (2.18) mit $p = 1$, $f_{\min} = 0{,}2$, $f_{\max} = 2$ und $S = 0{,}9$ abgeschätzt (siehe Zeilen 601–635). Wie in Abschnitt 2.4 erwähnt, wird die Fehlernorm $\hat{f}_I$ aus Gleichung (2.19) verwendet, wobei in `statev` unterschiedliche Toleranzen für Spannungskomponenten und alle Statusvariablen definiert werden können.

[2] Verfügbar unter »https://soilmodels.com/download/umat-hyposand-uibk/. Im dortigen Archiv sind mehrere Benutzerroutinen enthalten, wobei hier nur die Version mit kleinen Dehnungen und Richardson-Extrapolation betrachtet wird.

Bei Problemen in der Integration wird ein Viertel des aktuellen Zeitschritts als Vorschlag für den nächsten Aufruf in `pnewdt` zurückgegeben.

Die intergranularen Dehnungen werden mit `nstatv(5)` ≥ 1 aktiviert, da sie ansonsten nicht in den Statusvariablen gespeichert werden können (siehe Abfragen in Zeile 932 und 1037).

Für $||\mathbf{D}|| = 0$ wird die Jacobi-Matrix `ddsdde` mit `getDZjac()` abgeschätzt, ansonsten wird sie im jeweiligen Integrationsverfahren als `B` in der Funktion `euler()` und `imeuler()` numerisch bestimmt.

Bei Spannungszuständen nahe null mit $\mathrm{tr}(\mathbf{T}) > \sigma_{\mathrm{el}}$ wird die Spannungsänderung ersatzweise über den folgenden Zusammenhang bestimmt (Zeile 811–813)

$$\mathring{T}_{ii} = \frac{\dot{e}}{e} \cdot \frac{\sigma_{\mathrm{el}}}{3n_H} \left(\frac{h_s}{\sigma_{\mathrm{el}}} \right)^{n_H} \quad . \tag{3.2}$$

Dabei gilt $\sigma_{\mathrm{el}} = 10^{-3}$.

PLX09

Die Routine von D. Mašín mit Code von C. Tamagnini, E. Sellari und P.-A. von Wolffersdorf enthält eine vorkompilierte `.dll`-Bibliothek für die Nutzung aus Plaxis.[3]

Es sind zwei hypoplastische Stoffmodelle verfügbar (Sand und Ton), wobei hier nur die Version für Sand betrachtet wird.[4] Um die `UMAT`-Schnittstelle aus Abaqus zu nutzen, muss eine `testing`-Variable in der `umat.for`-Datei angepasst werden.

```
228       testing=0
229 c     For use in PLAXIS, activate the following line
230       if(kstep.eq.1 .AND. kinc.eq.1) testing=1
231 c     For use in ABAQUS EXPLICIT, activate the following line
232 c     if(kstep.eq.1 .AND. kinc.eq.1) testing=3
233 c     For use in ABAQUS, the two lines above should be inactive
```

Wie bei der Benutzerroutine `ABQS02` wird in `props` ebenfalls ein hydrostatischer Druck p_c erwartet, der als $p_c\delta_{ij}$ vom Spannungstensor vor Übergabe an das Stoffmodell abgezogen wird, sowie die Dichte von Wasser ρ_w, um den Porenwasserüberdruck $u_{n+1} = u_n - \rho_w \cdot \mathrm{tr}(\mathbf{D})$ zu berechnen. Im Vergleich zu den anderen betrachteten Benutzerroutinen sei auf die andere Reihenfolge von m_R und m_T in `props` hingewiesen.

Die Integration wird mit der Simpsonregel durchgeführt, einem expliziten Runge-Kutta-Verfahren der Ordnung zwei mit Fehlerabschätzung. Zum Akzeptieren eines Integrationsschritts muss $||R||/t_A < 1$ mit der folgenden Fehlernorm erfüllt sein (siehe Funktion `norm_res_h()` ab Zeile 1323 in `umat.for`)

$$R_i = |\hat{y}_{n+1,i} - \hat{y}_{p,n+1,i}|/||\hat{\boldsymbol{y}}_{n+1}|| \quad .$$

[3] Verfügbar unter »https://soilmodels.com/download/plaxis-umat-hypoplas-zip/

[4] Die Auswahl der beiden Stoffmodelle ist für Plaxis optimiert, da die `UMAT()`-Funktion selbst nur die Version für Sand enthält.

In `rkf23_update()` wird die nächste Schrittlänge basierend auf Gleichung (2.18) abgeschätzt. Dabei gilt $p = 2$, $f_{\min} = 0{,}25$, $f_{\max} = 4$ und $S = 0{,}9$ sowie (siehe Zeilen 1696–1720)

$$\hat{f} = \begin{cases} \frac{h_{\text{sub}}}{h} \cdot \frac{t_A}{||R||} & \text{für} \quad ||R|| \neq 0 \\ 1 & \text{sonst} \end{cases}$$

mit Zeitinkrement h und Subinkrement h_{sub}. Das Konzept der intergranularen Dehnung wird mit dem Materialparameter $m_R > 0{,}5$ aktiviert (Zeile 1435–1439). Die Jacobi-Matrix wird in `perturbate_h()` als $m_R\mathcal{L}$ definiert für $m_R > 0{,}5$ und ansonsten $\mathcal{L}$ (Zeile 1459–1471). Als einzige der hier betrachteten Benutzerroutinen wird intern eine Matrixrepräsentation der S4-Tensoren verwendet (siehe auch Abschnitt 5.2).

ABQS09

Diese nicht öffentliche Routine von A. Niemunis stellt eine `UMAT`-Schnittstelle bereit. Sie enthält das hypoplastische Stoffmodell nach von Wolffersdorff (1996) mit dem Konzept der intergranularen Dehnung nach Niemunis und Herle (1997) und zusätzlichen Anpassungen, wie ein modifiziertes f_d nach Gleichung (2.10) und eine erhöhte Schubsteifigkeit durch Modifikation von $\mathcal{L}$ und Matrix $\mathbf{N}$, d. h.

$$\begin{aligned} \mathbf{N}_{\text{incr}} &= \mathcal{L}_{\text{incr}} : \left(\mathcal{L}^{-1} : \mathbf{N}\right) \quad \text{mit} \\ \mathcal{L}_{\text{incr}} &= \frac{f_b f_e}{\hat{\mathbf{T}} : \hat{\mathbf{T}}} \cdot \left(\left(F^2 + b^2\right) \mathcal{I}^{\text{sym}} + a^2 \hat{\mathbf{T}} \otimes \hat{\mathbf{T}} - \frac{b^2}{3} \mathcal{I}^{\text{tr}} \right) \quad \text{und} \\ \mathcal{L}^{-1} &= \frac{\hat{\mathbf{T}} : \hat{\mathbf{T}}}{f_b f_e F^2} \left(\mathcal{I}^{\text{sym}} - \frac{\hat{\mathbf{T}} \otimes \hat{\mathbf{T}}}{\left(\frac{F}{a}\right)^2 + \hat{\mathbf{T}} : \hat{\mathbf{T}}} \right) \quad , \end{aligned} \tag{3.3}$$

sowie

$$b^2 = \left(1 + \frac{a^2}{3} + \frac{a}{\sqrt{3}}\right) \frac{1 - 2\nu}{1 + \nu} - 1$$

mit der Querdehnzahl ν. In der Benutzerroutine wird die Jacobi-Matrix $\mathcal{J}$ in `ddsdde` bei deaktivierter intergranularer Dehnung entweder mit $2\mathcal{L}h_{\text{sub}}/h$ oder (standardmäßig) folgendermaßen berechnet

$$\begin{aligned} y_{\text{out}} &= \min\left(0{,}95 \ , \ \frac{0{,}95}{||\mathcal{L}^{-1} : \mathbf{N}||} + 10^{-10}\right) \quad \text{und} \\ \mathcal{J}_{i+1} &= \mathcal{J}_i + \left(\mathcal{L} + y_{\text{out}} \mathbf{N} \otimes ||\mathbf{D}||\right) \frac{h_{\text{sub}}}{h} \quad . \end{aligned} \tag{3.4}$$

Neben den hypoplastischen Materialparametern wird die Querdehnzahl ν zur Berechnung einer erhöhten Schubsteifigkeit erwartet. Die Integration der Spannungen erfolgt nach dem expliziten Euler-Verfahren, während die intergranulare Dehnung $\mathbf{h}$, die mit $m_R \geq 2$ aktiviert wird, explizit oder implizit integriert werden kann (siehe Niemunis, 2003, S. 106). Für die Integration der intergranularen Dehnungen wird der minimale Zeitschritt auf ein Zehntel von h_{sub} festgelegt.

ABQE10

Diese Routine ist eine modifizierte und für die VUMAT-Schnittstelle (Abaqus/Explizit) angepasste Version der Benutzerroutine ABQS09 von G. Qiu. Im Vergleich dazu wird

- ein Porendruck p_p mit $T_{ii} = T_{ii} + p_c - p_p$ wird berücksichtigt. Der Porendruck entwickelt sich über den Verzerrungstensor zu $p_{p,n+1} = p_{p,n} - \rho_w \operatorname{tr}(\mathbf{D})\, h$,
- f_d nach der ursprünglichen Definition aus von Wolffersdorff (1996) berechnet,
- die Porenzahl im ersten Zeitschritt optional mit dem Parameter e_{ini} über Gleichung (2.7) bestimmt und
- keine Jacobi-Matrix ddsdde berechnet.

ABQS04

Diese Benutzerroutine von K. Nübel und A. Niemunis in der Überarbeitung von K.-P. Mahutka stellt eine UMAT-Schnittstelle für die Nutzung in Abaqus/Standard bereit. Das Konzept der intergranularen Dehnung ist in dieser Implementierung immer aktiv.

Diese Benutzerroutine enthält ein lineares Ersatzmodell, das anstelle des hypoplastischen Stoffmodells genutzt wird, wenn $\operatorname{tr}(\mathbf{T}) > \sigma_{\mathrm{el}}$. Für das lineare Ersatzmodell wird $E = \min(E_{\mathrm{min}}, E_{\mathrm{rep}})$ und $\nu = 0$ angenommen.

MAT04

Diese Routine ist die direkte Portierung von ABQS04 nach MATLAB. Im Gegensatz zur UMAT-Schnittstelle werden beim Aufruf nur die folgenden Argumente in dieser Reihenfolge erwartet: stress, statev, ddsdde, stran, dstran, dtime, ndi, ntens, props, drot, kstep, kinc. Die Rückgabewerte sind die berechnete Spannung, Statusvariablen und Jacobi-Matrix. Die wesentlichen Unterschiede gegenüber ABQS04 sind, dass

- vier Statusvariablen nicht genutzt werden (4 bis 6 und 14),
- keine minimale Steifigkeit für das elastische Ersatzmodell als interner Parameter ela vorgegeben werden kann und
- der Schalter imp_intg ungenutzt ist, da die intergranularen Dehnungen immer explizit integriert werden.

3.1.2 Weitere Benutzerroutinen und Schnittstellen

In Bienen u. a. (2014) wird eine Vorlage präsentiert, um Benutzerroutinen mit UMAT-Schnittstelle für Abaqus/Standard über die VUMAT-Schnittstelle in Abaqus/Explizit nutzen zu können.[5] Die Vorlage lässt sich schnell anpassen und einfach einsetzen, aber es muss beachtet werden, dass

[5] Die Vorlage ist verfügbar unter »https://soilmodels.com/download/vumat_umatinterface-zip/

- von identischen Ein- und Ausgangsgrößen (z. B. für Spannung und Dehnungsrate) für beide Schnittstellen ausgegangen wird. Da die Eingangsgrößen von Abaqus/Standard rotiert sind (mitbewegte Koordinaten) und die von Abaqus/Explizit nicht (raumfeste Koordinaten), trifft das nur für geradlinige Belastungen ohne Verzerrung zu (siehe Dassault Systèmes, 2020) und
- die Jacobi-Matrix `ddsdde` ohne Änderungen in der `UMAT`-Schnittstelle berechnet wird, obwohl das für die `VUMAT`-Schnittstelle nicht notwendig ist (siehe Abschnitt 4.3).

Neben der vorgestellten Benutzerroutine `MAT04` gibt es auch eine Implementierung der Hypoplastizität in MATLAB im Rahmen einer Testumgebung für Stoffmodelle.[6] Ebenso gibt es dort Implementierungen der Barodesie mit `UMAT`-Schnittstelle und als Plaxis-Bibliothek. In Kolymbas (2022) und Schranz (2018) sind auch MATLAB-Implementierung zu finden.[7]

Zusammenfassend sind für die Stoffmodelle der Hypoplastizität und Barodesie Implementierungen für die `UMAT`- und `VUMAT`-Schnittstelle (Abaqus), die `USER_MOD`-Schnittstelle (Plaxis) und für MATLAB verfügbar. Für das gleiche Stoffmodell (ggf. mit Modifikationen) liegen also verschiedene Implementierungen vor, die über verschiedene Schnittstellen und Programmiersprachen begründet sind.

3.2 Überlegungen zur Implementierung von Benutzerroutinen

Programming is neither fully an art nor fully a science. As it's typically practiced, it's a "craft" that's somewhere between art and science. At its best, it's an engineering discipline that arises from the synergistic fusion of art and science.

McConnell (2004), S. 848

Bei der Überarbeitung bestehender Benutzerroutinen ist es nicht so einfach, grundlegende Entscheidungen einer Implementierung, wie Programmierstil oder Programmiersprache, zu ändern. Im Gegensatz dazu sind bei einem neuen Projekt alle Optionen verfügbar. Deshalb folgen ein paar Gedanken zu den Entscheidungen, die bei der Entwicklung der xMat getroffen worden sind.

Programmiersprache

Die Wahl der Programmiersprache wird davon beeinflusst, ob bzw. mit welchem Aufwand die Anbindung der Benutzerroutine an die zu verwendenden Programme möglich ist. Alle in Abschnitt 3.1 betrachteten Benutzerroutinen werden aus Abaqus/Standard, Abaqus/Explizit, Plaxis oder MATLAB genutzt. Für die Nutzung in Plaxis muss die Benutzerroutine

[6] Die Testumgebung ist verfügbar unter »https://soilmodels.com/download/matlab-driver/

[7] Medicus, Fellin und Schranz (2018), S. 635, verweisen ebenfalls auf die Verfügbarkeit von Benutzerroutinen zur Barodesie für Abaqus und Plaxis und die grundsätzliche Möglichkeit für eine Kalibrierung mit ExCalibre (»https://soilmodels.com/excalibre/).

als vorkompilierte Bibliothek vorliegen (siehe Bentley Systems, 2016). In Abaqus kann entweder eine vorkompilierte Bibliothek genutzt werden, oder der Quellcode in Fortran, C oder C++ (siehe Dassault Systèmes, 2020). Auch in MATLAB ist die Einbindung vorkompilierter Bibliotheken oder von Quellcode in der Skriptsprache von MATLAB möglich (siehe The MathWorks, Inc., 2021). Die Ausführungsgeschwindigkeit eines Programms einer kompilierten Programmiersprache, wie Fortran, C oder C++, ist i. d. R. höher als bei einem vergleichbaren Skript in einer interpretierten Sprache, wie Python oder in MATLAB.[8]

Im Gegensatz dazu ist es mit interpretierten Sprachen möglich, interaktiv mit einer Benutzerroutine in der jeweiligen Sprache zu interagieren sowie schnell und einfach Änderungen durchzuführen. Für die Anbindung an die Simulationsprogramme ist bei Benutzerroutinen in der Geotechnik die Fortran-Version maßgebend, auch wenn es von den betrachteten Stoffmodellen Benutzerroutinen für MATLAB gibt.

Aufgrund der Vorteile durch Ausführungsgeschwindigkeit und einfache Anbindung an andere Programme ist die xMat in Fortran 2003 und im `free`-Format programmiert.[9] Prinzipiell wären neben Fortran auch C oder C++ geeignete Programmiersprachen, aber die Verfügbarkeit anderer Benutzerroutinen für geotechnische Stoffmodelle ist für Fortran höher. Dadurch ist es auch einfacher, andere bestehende Stoffmodelle in die xMat zu integrieren (siehe Abschnitt 5.3). Nichtsdestotrotz bietet sich für eine breite Nutzungsmöglichkeit zumindest eine Schnittstelle zu C an, da viele Programmiersprachen eine Nutzung von C-Bibliotheken unterstützen.

Fortran unterscheidet zwischen den Programmeinheiten `subroutine` und `function`, wobei im Folgenden beide mit dem Begriff *Funktion* bezeichnet werden. Für eine einfache Unterscheidung zu Variablen wird an den Namen einzelner Funktionen wird ein „`()`“ angehängt, unabhängig von der Existenz und Anzahl an Argumenten. Wenn auf die Unterscheidung zwischen `function` und `subroutine` Wert gelegt werden soll, wird bei `subroutine`-Einheiten wie im Programmcode ein `call` vorangestellt.

Konzept

Die xMat soll als Containerstruktur ein einfaches Arbeiten mit verschiedenen Stoffmodellen, Integrationsverfahren und Schnittstellen ermöglichen. Für eine Austauschbarkeit von einzelnen Funktionalitäten empfiehlt sich daher eine klare Definition und Abgrenzung einzelner Programmteile. Ein Konzept, das diese Kapselung fördert, ist die objektorientierte Programmierung.

Mit einer objektorientierten Programmierung kann eine bessere Strukturierung und Lesbarkeit, sowie eine einfachere Anpassbarkeit und Wartung des Codes erreicht werden. Nachteilig sind mögliche Geschwindigkeitseinbußen bei der Ausführung durch zusätzliche

[8] Die erreichbare Geschwindigkeit hängt auch von weiteren Faktoren, wie Programmierstil und Compiler, ab.

[9] Eine Übersicht zu Fortran findet sich bspw. in Adams u. a. (2009) oder Metcalf u. a. (2011). Ein objektorientierter Programmierstil mit Fokus auf Datenstrukturen wird seit Fortran 2003 vom Sprachstandard unterstützt (siehe bspw. Metcalf u. a., 2011, S. 265–287). Die finalen Konzepte der Fortran-Standards können unter »https://wg5-fortran.org/ frei eingesehen werden.

Datenstrukturen und Abstraktionsschichten im Code, die aber durch eine durchdachte Struktur und einen guten Compiler relativiert werden. Die xMat soll zwar eine gute Leistung zeigen, aber im Zweifelsfall werden Lesbarkeit und Übersichtlichkeit höher bewertet.[10]

Stoffmodelle

Der Hauptfokus soll auf das hypoplastische Stoffmodell nach von Wolffersdorff (1996) gelegt werden, das mit verschiedenen Optionen angepasst werden kann. Zusätzlich soll auch das barodetische Stoffmodell nach Kolymbas (2021) etwas genauer untersucht werden. Für Vergleiche und weitere Untersuchungen sollen andere Stoffmodelle bereitgestellt werden (Hypoplastizität nach Wu, 1992; Barodesie nach Kolymbas, 2015; und Barodesie nach Schranz, 2018).

Integrationsverfahren

Wie in anderen Benutzerroutinen soll das explizite Euler-Verfahren genutzt werden können. Dabei ist zu beachten, dass dieses Verfahren aufgrund der bekannten Defizite nicht für den Produktivbetrieb empfohlen wird (siehe z. B. Press u. a., 1997, S. 704, oder Hairer, Nørsett u. a., 2008, S. 40, sowie Butcher, 2016, S. 55–89, für eine detailierte Untersuchung des expliziten Euler-Verfahrens). Aus Gründen der Vergleichbarkeit sollen ebenfalls das explizite Euler-Verfahren mit Richardson-Extrapolation sowie zwei explizite Runge-Kutta-Verfahren zweiter Ordnung verfügbar sein (Simpsonregel und Bogacki-Shampine). Aufgrund von Untersuchungen mit expliziten Runge-Kutta-Verfahren höherer Ordnung sollen drei weitere Verfahren enthalten sein (Fehlberg, Cash-Karp und Dormand-Prince).

Schnittstellen

Prinzipiell soll die xMat von möglichst vielen Programmen genutzt werden können, in denen das Bodenverhalten geotechnischer Simulationen durch Benutzerroutinen gesteuert werden kann. Konkret soll die Nutzung aus Abaqus/Standard, Abaqus/Explizit, Plaxis und MATLAB möglich sein. Zusätzlich soll eine C-Schnittstelle für die Anbindung an zusätzliche Programmiersprachen vorhanden sein.

Allgemeine Empfehlungen

Beim Programmieren findet auch eine Kommunikation mit dem zukünftigen Leser des Codes statt. Je klarer die Kommunikation ist, desto einfacher ist die Orientierung im Code und desto schneller kann der Blick des Lesers auf das Wesentliche gelenkt werden. Das hilft nicht nur anderen, sondern auch dem Autor selbst, insbesondere wenn der letzte Blick in den Code einige Zeit zurückliegt.

[10] Letztendlich ist die Einarbeitung in Benutzerroutinen anderer Autoren schwierig genug und sollte nicht zusätzlich erschwert werden.

Es folgen einige subjektive Empfehlungen für ein Programmierprojekt, an denen sich die Implementierung der xMat in Kapitel 4 orientiert. Sie basieren auf Anmerkungen aus Martin (2009), McConnell (2004) und eigenen Erfahrungen. Weitere Anregungen und Arbeiten zum Thema finden sich bspw. in Wilson u. a. (2014).

- Bevor es richtig losgeht, bereits das Ziel möglichst konkret formulieren:
  - *Anforderungen mit Teilschritten* und ggf. optionalen Zielen formulieren,
  - *Nutzung vorhandener und getesteter Bibliotheken* für einzelne Funktionalitäten abwägen und
  - *Entwicklungsumgebungen und Debugger* als Unterstützung in Betracht ziehen (z. B. Microsoft Visual Studio, Code::Blocks, Eclipse und gdb).
- Die einzelnen Aufgaben in überschaubare Programmeinheiten unterteilen, d. h.
  - *Kapselung von Funktionalitäten* (z. B. in `function`- oder `subroutine`-Blöcke, siehe auch McConnell, 2004, S. 161–186),
  - *Projektstruktur logisch abbilden*, u. a. durch Klassen und Objekte sowie Auslagerung von Einheiten in einzelne Dateien und
  - *Komplexität vermeiden* und den Code so verständlich und klar wie möglich formulieren.
- Den Code möglichst natürlich und strukturiert ausdrücken. Fortran unterstützt sowohl prozedurale als auch objektorientierte Programmierung und ermöglicht es, den Programmierstil an die Problemstellung anzupassen (siehe Metcalf u. a., 2011; oder Rouson u. a., 2011).[11]
- Eine einheitliche Programmierstruktur innerhalb eines Projekts verwenden und sich um einen „sauberen“ Codestil bemühen (siehe Martin, 2009, S. 285–314, für die Erkennung von Verbesserungsmöglichkeiten in bestehendem Code). Das heißt z. B.
  - *einheitliche Programmierkonventionen verwenden* und (moderne) Standards berücksichtigen. Exotische oder veraltete Funktionalitäten nach Möglichkeit vermeiden. Weitere Anregungen finden sich in Markus (2012), S. 121–134, Rouson u. a. (2011), S. 27–30, oder unter »https://www.fortran90.org/src/best-practices.html,
  - *aussagekräftige Namen für Variablen/Funktionen/Klassen wählen*, so soll die Bedeutung selbsterklärend aus dem Namen folgen, die Namen aussprechbar und eindeutig bei einer Suche auffindbar sein (vor allem Variablennamen) und gleiche Konzepte sollten gleich heißen (siehe Martin, 2009, S. 17–30; oder McConnell, 2004, S. 259–289),
  - *Scheinargumente (*„dummy arguments“*) beim Aufruf von Funktionen nutzen*, um eine eindeutige Zuordnung anzuzeigen und Fehler bei der Reihenfolge von Übergabeargumenten zu vermeiden,

[11] Eine kurze Diskussion zu „modernen“ Fortran-Büchern findet sich unter »https://stevelionel.com/drfortran/2013/12/30/doctor-fortran-in-its-a-modern-fortran-world/ und Beispiele für objektorientierte Konzepte unter »https://gist.github.com/n-s-k/522f2669979ed6d0582b8e80cf6c95fd

  - *Funktionalitäten sinnvoll kapseln*, z. B. sollen Funktionen nicht nur Wiederholungen im Code verhindern, sondern möglichst klein und übersichtlich sein. Idealerweise wird in einer Funktion nur eine Aufgabe einer Abstraktionsebene erledigt, Seiteneffekte vermieden und die Anzahl an Übergabeargumenten möglichst gering gehalten (siehe Martin, 2009, S. 31–52),
  - *Kommentare wohlüberlegt einsetzen*, um Absichten und Konsequenzen mit Kommentaren deutlich zu machen. Schwer verständlichen Code neu schreiben statt zu kommentieren und auskommentierten Code sowie veraltete Kommentare vermeiden bzw. entfernen (siehe Martin, 2009, S. 53–74; oder McConnell, 2004, S. 785–817) und
  - *Präprozessor-Makros sparsam verwenden*, insbesondere `#ifdef`-Anweisungen. Stellenweise kann es vorteilhafter und verständlicher sein, den Programmfluss über Variablen mit `parameter`-Attributen und `if`-Abfragen zu steuern. In beiden Fällen erkennt der Compiler schon beim Kompilieren, welche Zweige von Abfragen erreicht werden können und entfernt nicht erreichbaren Code.

- Den Code im Gesamten dokumentieren. Eine Kurzbeschreibung der Funktionen und Nutzungsbeispiele (ggf. in einem separaten Dokument) für zukünftige Anwender erstellen.[12]
- Es reicht nicht aus, wenn ein Programm „gerade so läuft“. Die folgenden Punkte zeigen Möglichkeiten auf, um Robustheit und Verlässlichkeit des Codes zu steigern:
  - *Refactoring des vorhandenen Quellcodes durchführen*, also kritisches Prüfen und ggf. Umformulierung von bereits vorhandenem Code in sinnvolleren Einheiten (siehe z. B. McConnell, 2004, S. 563–585),
  - *empfohlene Funktionalitäten* und bewährte Lösungen nutzen, aber auch auf veraltete Funktionalitäten der Programmiersprache möglichst verzichten,
  - *verschiedene Compiler nutzen und Compiler-Optionen anpassen*, mit voller Debug-Stufe testen und auf alle Warnungen achten und
  - *mit anderen Implementierungen vergleichen*, Benchmarks nachbilden und Code mit einem Profiler untersuchen.
- Das Programm ausreichend testen, am besten in anderen Umgebungen und auf anderen Maschinen und Betriebssystemen. Es ist empfehlenswert, eigene Tests zu schreiben (*„Unit Tests“*, *„Test driven developement“*) und gezielt möglichst alle Zweige von Abfragen abzudecken (*„Code coverage“*).[13]
- Eine Versionsverwaltung, wie git oder Subversion, für die Entwicklung des eigenen Codes nutzen.

[12] Einige Programme, wie Doxygen (»https://www.doxygen.nl/index.html) oder FORD (»https://github.com/Fortran-FOSS-Programmers/ford), können durch strukturierte Kommentare im Fortran-Code automatisch eine Dokumentation erstellen.

[13] Ein automatisiertes Testen einzelner Programmteile ist z. B. mit dem Fortran Unit Test Framework möglich (»https://sourceforge.net/projects/fortranxunit/). Siehe auch Abschnitt 5.6 für weitere Hinweise.

4 Implementierung der xMat

In diesem Kapitel wird die Implementierung der xMat aufgezeigt, die im Rahmen dieser Arbeit erstellte Benutzerroutine und Containerstruktur. Der Überblick über die hypoplastischen Benutzerroutinen in Kapitel 3 legt die Vermutung nahe, dass die Anbindung an andere Schnittstellen der Hauptgrund einiger Implementierungen ist. Daher ist eines der wesentlichen Ziele der xMat, dass ein implementiertes Stoffmodell von verschiedenen Programmen aus genutzt werden kann, ohne neu implementiert werden zu müssen.

Abschnitt 4.1 gibt einen Überblick über die Struktur der xMat und die Absichten dahinter. Anschließend folgen in Abschnitt 4.2 die Implementierungsdetails der einzelnen Module und in Abschnitt 4.3 die Beschreibung der Schnittstellen an andere Programme.

Wichtige Erkenntnisse aus diesem Kapitel sind:

Struktur

- In der xMat wird unterschieden zwischen Schnittstellen mit anderen Programmen und internen Abläufen, die von allen Schnittstellen aus gleich genutzt werden.

Implementierung

- Alle wichtigen Einstellungen innerhalb der xMat sind im Modul `General_Settings` zu finden.
- Für jedes Stoffmodell und jedes Integrationsverfahren existiert eine eigene Klasse.

Modularität der xMat

- Der modulare Aufbau der xMat ermöglicht es, verschiedene Stoffmodelle, Integrationsverfahren und Schnittstellen zu nutzen und hinzuzufügen (siehe Kapitel 5).

Schnittstellen

- Durch die vorhandenen Schnittstellen und speziell die C-Schnittstelle ist eine Anbindung der xMat an weitere Programmiersprachen vergleichsweise einfach möglich.

4.1 Struktur der xMat

And simplicity is the unavoidable price which we must pay for reliability.

E. Dijkstra zitiert C.A.R. Hoare, Dijkstra (1982), S. 127[1]

4.1.1 Schnittstellen und interne Abläufe

Der Aufbau der xMat lässt sich in die zwei Bereiche Schnittstellen und interne Abläufe unterteilen. *Interne Abläufe* bezeichnen das Zusammenspiel zwischen Stoffmodellen und Integrationsverfahren zur Berechnung einer Spannungsantwort. *Schnittstellen* (*„interfaces“*) übernehmen die Kommunikation zwischen aufrufenden Programmen und internen Abläufen (siehe auch Bild 4.1). Deshalb muss jedes aufrufende Programm über eine entsprechende Schnittstelle angebunden sein. Die verfügbaren Schnittstellen werden in den Abschnitten 4.3.1 bis 4.3.6 thematisiert. Allgemeine Informationen zur Anbindung von Programmen an die Schnittstellen finden sich am Anfang von Abschnitt 4.3.

Mit geeigneter Vorverarbeitung in den Schnittstellen ist es möglich, dass die internen Abläufe unabhängig von den jeweils aufrufenden Programmen sind. So können die vorhandenen Stoffmodelle und Integrationsverfahren ohne zusätzliche Anpassungen von allen Schnittstellen genutzt werden. Einerseits sind damit Anpassungen interner Abläufe (wie bspw. die Modifikation eines Stoffmodells) direkt für alle Schnittstellen verfügbar. Andererseits heißt das auch, dass bisher noch nicht unterstützte Programme durch Bereitstellung einer geeigneten Schnittstelle alle implementierten Funktionalitäten nutzen können.

Die Kommunikation zwischen einer Schnittstelle und den internen Abläufen wird über die drei Funktionen `Xmat_Initialize()`, `Import_State()` und `Calculate_Results()` bestimmt:

- Nach der Vorverarbeitung wird mit dem Aufruf von `subroutine Xmat_Initialize(` $\texttt{xmat_obj}_{\text{out}}$, $\texttt{solver_name}^{\text{in}}$, $\texttt{constitutive_model_name}^{\text{in}}$, $\texttt{material_parameters}^{\text{in}}$, $\texttt{calculate_jacobian}^{\text{in}}$, $\texttt{firstcall}^{\text{in}}$`)` zuerst ein `Xmat`-Objekt `xmat_obj` erstellt und eigener Speicherplatz zugewiesen. Anschließend werden anhand von `solver_name` und `constitutive_model_name` ein Integrationsverfahren über `Select_Solver()` und ein Stoffmodell über `Select_Constitutive_Model()` initialisiert. Dem Stoffmodell werden die Materialparameter (`material_parameters`) zugewiesen, die sich in der gesamten Bearbeitung nicht mehr ändern. Außerdem signalisieren zwei Wahrheitswerte, ob eine Berechnung der Jacobi-Matrix gefordert ist (`calculate_jacobian`) und ob dies der erste Aufruf ist und somit interne Überprüfungen durchgeführt werden sollen (`firstcall`).
- Der Aufruf von `subroutine Import_State(`$\texttt{this}^{\text{in}}_{\text{out}}$, $\texttt{nstates}^{\text{in}}$, $\texttt{statevariables}^{\text{in}}$, $\texttt{stress}^{\text{in}}$, $\texttt{strain}^{\text{in}}$, $\texttt{totaltime}^{\text{in}}$, $\texttt{timeincrement}^{\text{in}}$`)` übergibt die Anzahl und Werte der aktuellen Zustandsvariablen (`statevariables` mit der Länge `nstates`) sowie die aktuellen Cauchy-Spannungen (`stress`) und die dazugehörigen Dehnungen (`strain`), den aktuellen Zeitpunkt (`totaltime`) und das Zeitinkrement (`timeincrement`). Intern

[1] Digital verfügbar unter »https://www.cs.utexas.edu/~EWD/transcriptions/EWD04xx/EWD494.html

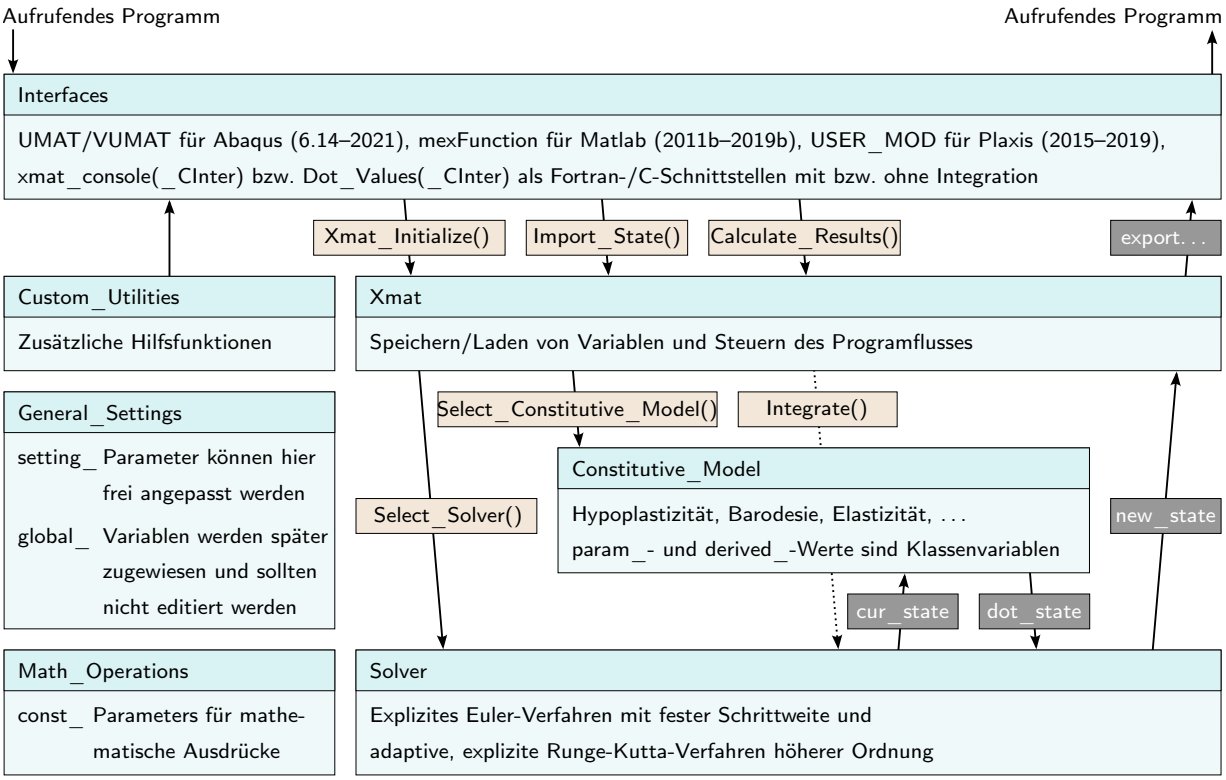


Bild 4.1: Module, zentrale Steuerungsfunktionen und angedeuteter Programmablauf innerhalb der xMat (von links nach rechts). Für jedes aufrufende Programm ist eine Schnittstelle notwendig, um alle Werte für die interne Berechnung vorzubereiten bzw. wieder korrekt an das aufrufende Programm zurückzugeben. Jede Schnittstelle steuert die interne Berechnung über die Funktionsaufrufe `Xmat_Initialize()`, `Import_State()` und `Calculate_Results()`. In `Custom_Utilities` werden zusätzliche Hilfsfunktionen für die Schnittstellen bereitgestellt. Außerdem sind für alle Module/Schnittstellen allgemeine Einstellungen in `General_Settings` und mathematische Operationen in `Math_Operations` definiert.

werden aus den Dehnungen die Dehnungsraten (`assigned_dot_strain`) für dieses Zeitinkrement berechnet, die an die ratenabhängigen Stoffmodelle weitergeleitet werden. Durch die Trennung von `Xmat_Initialize()` und `Import_State()` ist es möglich, dass mit einem Aufruf der Benutzerroutine prinzipiell mehrere Elemente mit den gleichen Materialparametern bearbeitet werden können, ohne `Xmat`-Objekt, `Solver` und `Constitutive_Model` neu zu initialisieren. Dazu müssen alle Elemente dieses Aufrufs das gleiche Stoffmodell und das gleiche Integrationsverfahren verwenden (siehe z. B. die `VUMAT`-Schnittstelle zur Anbindung an Abaqus/Explizit).

- Zu jedem `Import_State()` gehört ein Aufruf von `subroutine Calculate_Results(` $\texttt{this}^{\text{in}}_{\text{out}}$, $\texttt{exportstress}_{\text{out}}$, $\texttt{exportstates}_{\text{out}}$, $\texttt{exportjacobian}_{\text{out}}$, $\texttt{exportdt}_{\text{out}}$), um das Spannungs-Dehnungs-Verhalten mit den vorab übergebenen Werten zu bestimmen. Intern wird zuerst die Länge des ersten Zeitschritts aus dem übergebenen Zeitinkrement und Einstellungen wie `setting_initial_substep_scale` berechnet. Anschließend wird ein Zustandsvektor mit allen relevanten Integrationsgrößen erzeugt

(Spannungen, Jacobi-Matrix, Zustandsvariablen) und von der `Integrate()`-Funktion des Integrationsverfahrens an die `Get_Dot_State()`-Funktion des entsprechenden Stoffmodells weitergeleitet. Das Integrationsverfahren bildet aus den Anfangswerten `cur_state` und den jeweiligen Werten aus `dot_state` die Approximation `new_state` und speichert intern die aktuelle Jacobi-Matrix, sodass am Ende die Spannungsantwort `exportstress`, die neuen Zustände `exportstates` und die Jacobi-Matrix $\mathcal{J}$ in `exportjacobian` zurückgegeben werden.

Für die Werte, die an `Import_State()` übergeben bzw. von `Calculate_Results()` empfangen werden, gilt:

- `stress` sind die aktuellen Cauchy-Spannungen (*„true Cauchy stress“*) – ggf. nach einer Starrkörperrotation – zu Beginn des Zeitinkrements und
- `exportstress` sind die berechneten Cauchy-Spannungen am Ende des Zeitinkrements,
- `strain` entspricht den mechanischen Dehnungsinkrementen des Zeitinkrements und somit dem Unterschied zwischen den Dehnungen zu Beginn und Ende des Zeitinkrements,
- `statevariables` entspricht weiteren Zustandsgrößen zu Beginn des Zeitinkrements, die vor der Übergabe ggf. zusätzlich verarbeitet werden müssen. Für tensorielle Einträge kann eine vorherige Starrkörperrotation notwendig sein und
- `exportstates` sind die berechneten Zustandsgrößen am Ende des Zeitinkrements.

Die interne Tensorrepräsentation der Komponenten wird in der Entwicklungsumgebung der xMat festgelegt (siehe Abschnitt 5.2). Für die beste Unterstützung wird empfohlen, alle Komponenten tensorieller Größen unabhängig von den vorhandenen Symmetrien in `statevariables` anzugeben.

4.1.2 Modularisierung

Die Flexibilität der xMat lässt sich über Module realisieren, die dynamisch während der Laufzeit verknüpft werden können. Code 4.1 zeigt ein Programm mit der verwendeten, objektorientierten Struktur, die für Stoffmodelle und Integrationsverfahren in der xMat zum Einsatz kommt (mit *„type-bound procedures“*, siehe z. B. Markus, 2012, S. 62 f.).

Es werden mehrere Klassen als `type`-Objekte bereitgestellt (hier `ConcreteExample01` und `ConcreteExample02` ab Zeile 26 bzw. 45), die jeweils eine gemeinsame Basisklasse (`Examples` ab Zeile 6) haben. Alle Definitionen sind in eigene Module ausgelagert, was insbesondere bei umfangreicheren Klassen zur Übersichtlichkeit beiträgt.[2] Die abgeleiteten Klassen setzen sich aus den eigenen Variablen und Funktionen sowie denen der Basisklasse zusammen (*„Vererbung“*). Im Hauptprogramm wird ein Element vom Typ der Basisklasse initialisiert (Zeile 98), dem zur Laufzeit nach frei definierbaren Kriterien eine der abgeleiteten Klassen zugeordnet wird (in `Select_Example()` ab Zeile 67).

[2] Mit dem Fortran 2008-Standard oder neuer ist auch eine Lösung mit `submodule` für die einzelnen Komponenten denkbar, was im Rahmen dieser Arbeit nicht weiter verfolgt wird.

Code 4.1: Beispiel der objektorientierten Struktur der xMat in Form eines kleinen Programms.

```
module Example_Baseclass
   implicit none
   private
   public :: Examples
   ! ------------------------------------------------------------- !
   type, abstract :: Examples
   ! ------------------------------------------------------------- !
      character(len=20) :: name = ''
      contains
      procedure(interf_info), deferred :: Info
   end type
   ! ------------------------------------------------------------- !
   abstract interface
   ! ------------------------------------------------------------- !
      subroutine interf_info(this)
         import
         class(Examples), intent(in) :: this
      end subroutine
   end interface
end module

module ConcreteExample01_Class
   use Example_Baseclass
   implicit none
   ! ------------------------------------------------------------- !
   type, extends(Examples) :: ConcreteExample01
   ! ------------------------------------------------------------- !
      private
      contains
      procedure :: Info
   end type
   contains
   ! ------------------------------------------------------------- !
   subroutine Info(this)
   ! ------------------------------------------------------------- !
      class(ConcreteExample01), intent(in) :: this
      write(*, '(a)') 'ConcreteExample01 uses a longer introduction for the name: ' // this%name
   end subroutine
end module

module ConcreteExample02_Class
   use Example_Baseclass
   implicit none
   ! ------------------------------------------------------------- !
   type, extends(Examples) :: ConcreteExample02
   ! ------------------------------------------------------------- !
      private
      contains
      procedure :: Info
   end type
   contains
   ! ------------------------------------------------------------- !
   subroutine Info(this)
   ! ------------------------------------------------------------- !
      class(ConcreteExample02), intent(in) :: this
      write(*, '(a)') 'ConcreteExample02 name: ' // this%name
   end subroutine
end module

```

```
60 module Example_Class
61    use Example_Baseclass
62    implicit none
63    private
64    public :: Select_Example, Examples
65    contains
66    ! ------------------------------------------------------------- !
67    function Select_Example(idx_example) result(myExample)
68    ! ------------------------------------------------------------- !
69       use ConcreteExample01_Class
70       use ConcreteExample02_Class
71       integer, intent(in) :: idx_example
72       class(Examples), allocatable :: myExample
73       if (idx_example == 1) then
74          allocate(ConcreteExample01::myExample)
75          myExample%name = 'Example Class 01'
76       else
77          allocate(ConcreteExample02::myExample)
78          myExample%name = 'Other Example Class'
79       end if
80    end function
81 end module
82
83 program testprog
84    implicit none
85    integer :: idx_example
86    write(*, '(a)')  'Input (1 or 2): '
87    read(*, '(i1)', err=110) idx_example
88    call workenv(idx_example)
89    return
90    110 write(*, '(a)') 'Exiting: Invalid input'
91    contains
92    ! ------------------------------------------------------------- !
93    subroutine workenv(idx_example)
94    ! ------------------------------------------------------------- !
95       use Example_Class
96       implicit none
97       integer, intent(in) :: idx_example
98       class(Examples), allocatable :: active_Example
99       active_Example = Select_Example(idx_example)
100      call active_Example%Info()
101   end subroutine
102 end program
```

Über das zugewiesene Objekt (hier `active_Example` in Zeile 99) können nicht nur Funktionen der Basisklasse aufgerufen werden, sondern auch Funktionen aus der abgeleiteten Klasse. Diese Funktionen müssen bereits in der Basisklasse mit dem `deferred`-Attribut definiert werden (siehe Zeile 10). Die eigentliche Definition befindet sich jeweils in den abgeleiteten Klassen (Zeile 30 und ab Zeile 34). Damit ergeben sich die folgenden Vorteile einer Modularisierung:

- Das Programm wird in sinnvolle Teilmodule mit einzelnen Aufgaben unterteilt,
- in allen Teilmodulen mit einer ähnlichen Aufgabe können gleiche Funktionalitäten in einer gemeinsamen Basisklasse implementiert werden,
- eine Auswahlfunktion (Zeile 67) koordiniert die Initialisierung während der Laufzeit und

- beim Arbeiten mit dem Objekt bzw. Aufruf einer dazugehörigen Funktion (Zeile 100) wird automatisch die Funktion der richtigen Klasse genutzt.

Die Nutzung von Variablen mit der Bezeichnung `allocatable` (Zeile 72 und 98) hat den Vorteil, dass der Speicher während der Laufzeit nur zugewiesen werden muss und beim Verlassen der jeweiligen Umgebung automatisch freigegeben wird. Im Allgemeinen wird für jede `allocate`-Anweisung auch eine `deallocate`-Anweisung benötigt, damit kein Speicher für das Programm verloren geht (*„memory leak“*).[3]

4.2 Implementierung der einzelnen Module

GOD IS REAL, unless declared INTEGER.

über implizite Variablen in Fortran (u. a. zitiert von S. Lionel)[4]

Die xMat nutzt Scheinargumente bei allen Aufrufen außer für intrinsische Funktionen, Debug-Ausgaben mit `Formatval()` und den mathematischen Funktionen. Das Hinzufügen von Scheinargumenten hat zwar einen längeren Ausdruck zur Folge, zeigt aber die tatsächliche Zuordnung von Argumenten auf. Dadurch wird ein unabsichtliches Vertauschen oder Vergessen von Argumenten erschwert, die Reihenfolge der Argumente kann frei gewählt werden und der Compiler hat bei einem Fehler mehr Informationen zur Verfügung.

Konstante Parameter wie $\sqrt{2}$ werden als Rechenausdruck und nicht als Zahlenwert gespeichert. Dadurch ist die Bedeutung bzw. Absicht besser nachvollziehbar, Übertragungsfehler werden reduziert und ein Wechsel der verwendeten Genauigkeit ist relativ einfach möglich. Außerdem ersetzt der Compiler den Rechenausdruck beim Kompilieren mit dem dazugehörigen Zahlenwert, sodass keine Nachteile während der Laufzeit entstehen.[5]

```
1 real(dp), parameter :: const_root2 = sqrt(2.0_dp)
```

4.2.1 Allgemeine Einstellungen (Modul `General_Settings`)

Das erste Modul `General_Settings` in der xMat ist als Schaltpult für zukünftige Anwender gedacht. Alle relevanten internen Parameter, die eine Anpassung der xMat an besondere Anforderungen ermöglichen, sind hier definiert statt an verschiedenen Stellen im Code.

Die Auswahl der einstellbaren Parameter ist auf Grundlage eigener Untersuchungen und Einschätzungen sowie Rückmeldungen aus dem Kollegium am Institut entstanden. Wenn zukünftige Anpassungen oder neue Module die Einstellbarkeit weiterer Parameter erforderlich machen, können diese ebenfalls in diesem Modul definiert und dokumentiert werden.

[3] Zur Speicherfreigabe siehe auch »https://stackoverflow.com/questions/61098780/memory-leak-when-assigning-allocatable-polymorphic-variable-with-return-value

[4] Doctor Fortran Blogeintrag: »https://stevelionel.com/drfortran/2021/09/18/doctor-fortran-in-implicit-dissent/

[5] Variablen ohne `parameter`-Attribut sollten in Fortran immer erst deklariert werden und in einer späteren Anweisung Werte zugewiesen bekommen. Anderenfalls wird ein implizites `save`-Attribut zugewiesen, was zu unerwarteten Ergebnissen führen kann (siehe »https://www.cs.rpi.edu/~szymansk/OOF90/bugs.html).

Idealerweise werden die in `General_Settings` hinterlegten Standardwerte von jedem Anwender vor der ersten Nutzung kritisch geprüft. Abhängig von den gestellten Anforderungen kann dadurch eingeschätzt werden, ob und welche Anpassungen für zukünftige Berechnungen notwendig sind bzw. sein können. Ein tieferer Einstieg in die restliche Routine ist für Anwender im Allgemeinen nicht erforderlich.[6] Als Referenz werden im Folgenden alle einstellbaren Parameter aus `General_Settings` beschrieben.

Grundsätzliche Einstellungen

- `dp` steuert die gewünschte Genauigkeit der Gleitkommazahlen aller intern verwendeten Variablen. Die Voreinstellung ist doppelte Genauigkeit mit `selected_real_kind(15)`. Falls nur eine einfache Genauigkeit benötigt wird, kann `selected_real_kind(7)` genutzt werden.[7]
- `setting_epsilon` dient als Vergleichswert für sehr kleine Gleitkommazahlen. Da Gleitkommazahlen im Allgemeinen nicht exakt repräsentiert werden können, sollte keine exakte Gleichheit von zwei Gleitkommazahlen überprüft werden. Daher bietet es sich an, die absolute Differenz zwischen beiden Zahlen zu betrachten. Solange diese Abweichung voneinander kleiner ist als `setting_epsilon`, werden beide Zahlen als identisch angenommen. In der Voreinstellung wird relativ großzügig mit `500.0_dp*epsilon(1.0_dp)` das 500-fache der Maschinengenauigkeit als Toleranz gewählt, was für doppelte Genauigkeit ungefähr dem Wert 10^{-13} entspricht.
- `setting_len_id` gibt an, mit wie vielen Buchstaben die Bezeichnungen von Stoffmodellen gespeichert werden. In der Voreinstellung werden neun Zeichen erwartet, damit das folgende, frei gewählte Schema erfüllt wird: Die ersten vier Buchstaben stehen für den Namen des Materialmodells, dann folgt ein Bindestrich und je zwei Zeichen für den Autor des Modells bzw. Veröffentlichung und das Jahr. Die folgenden Stoffmodelle sind hinterlegt:
  - `Elas-0000` entspricht der idealen Elastizität (`setting_id_elasticity`),
  - `Hypo-WU92` entspricht der Hypoplastizität nach Wu (1992) mit dem Parameter `setting_id_hypo_wu92`,
  - `Hypo-VW96` entspricht der Hypoplastizität nach von Wolffersdorff (1996) mit zuschaltbaren intergranularen Dehnungen nach Niemunis und Herle (1997) und optionalen Anpassungen nach Niemunis (2003) (`setting_id_hypo_vw96`),
  - `ViHy-Ni03` entspricht der Viskohypoplastizität nach Niemunis (2003) mit dem Parameter `setting_id_viscohypo_ni03`,

[6] Im Gegensatz dazu sollten sich Entwickler tiefer in den Quellcode einarbeiten, um sinnvollere Verbesserungen vornehmen oder zusätzliche Funktionalitäten implementieren zu können.

[7] Prinzipiell kann auch eine höhere Genauigkeit eingestellt werden, sofern das vom entsprechenden Compiler unterstützt wird. Im Kontext von Stoffmodellen und Benutzerroutinen für FE-Analysen ist die Wahrscheinlichkeit sehr hoch, dass Probleme bei der Berechnung eine andere Ursache als eine zu geringe Genauigkeit haben.

- `Baro-Ko15` entspricht der Barodesie nach Kolymbas (2015) mit dem Parameter `setting_id_barodesy_ko15`,
- `Baro-Sc18` entspricht der Barodesie nach Schranz (2018) mit dem Parameter `setting_id_barodesy_sc18` und
- `Baro-Ko21` entspricht der Barodesie nach Kolymbas (2021) mit dem Parameter `setting_id_barodesy_ko21`.

Während des Programmablaufs wird in der xMat anhand des Materialnamens überprüft, welches Stoffmodell für das aktuelle Element genutzt werden soll.[8] Die Bezeichnungen aller implementierten Stoffmodelle müssen an dieser Stelle mit einem eindeutigen Schlüssel der Länge `setting_len_id` hinterlegt werden.

Einstellungen zur Jacobi-Matrix

- `setting_numerical_jacobian` kann `.True.` oder `.False.` sein. Für `.False.` (Standard) wird die Jacobi-Matrix vom Stoffmodell berechnet, was genauer und effizienter sein kann. Falls die Jacobi-Matrix in dem gewählten Stoffmodell nicht berechnet werden kann oder wenn `.True.` für diesen Wert angegeben wird, erfolgt eine numerische Abschätzung der Jacobi-Matrix in der Integrationsschleife und
- `setting_numerical_jacobian_disturbance` gibt die Größe der Störung für die numerische Abschätzung der Jacobi-Matrix vor (Vorgabewert: 10^{-5}).

Einstellungen zur Hypoplastizität

- `setting_very_small_stress` definiert die kleinste gültige Druckspannung für hypoplastische Stoffmodelle (Vorgabewert ist $-0{,}01\,\mathrm{kN/m^2}$). Da die hypoplastischen Stoffmodelle nur für Druckspannungen definiert sind, muss im Rahmen von FE-Analysen eine Lösung für den Umgang mit Druckspannungen nahe null sowie Zugspannungen gefunden werden (siehe Abschnitt 4.2.2). Aktuell werden alle Spannungen größer als `setting_very_small_stress` als ungültig angesehen. Für solche Spannungen wird die Berechnung mit dem hypoplastischen Stoffmodell abgebrochen und das lineare Ersatzmodell genutzt.
- `setting_min_youngs_modulus` definiert das minimale Elastizitätsmodul $E_{\min}$ für das elastische Ersatzmodell. Vorgabewert für das minimale E-Modul ist $5\,\mathrm{kN/m^2}$.

Für das hypoplastische Stoffmodell `Hypo-VW96` sind außerdem die folgenden Anpassungen zuschaltbar:

- `setting_hypo_consistent_f_d` kann `.True.` (Standard) oder `.False.` sein. Für `.True.` wird f_d nach Gleichung (2.10) modifiziert, um die Einhaltung der unteren Grenze $e = e_d$ zu erzwingen.

[8] Die Materialnamen werden intern durch die Funktion `Uppercase()` in Großbuchstaben umgewandelt, sodass die Groß- und Kleinschreibung beim Aufruf nicht relevant ist.

- `setting_hypo_increased_stiffness` kann `.True.` oder `.False.` (Standard) sein. Falls der Wert auf `.True.` gesetzt ist, werden $\mathcal{L}$ und $\mathbf{N}$ nach Gleichung (3.3) für eine erhöhte Schubsteifigkeit modifiziert.

Einstellungen zum Integrationsverfahren

- `setting_solver_default` wird aktuell verwendet, um ein Integrationsverfahren auszuwählen. Die vordefinierten und implementierten Integrationsverfahren sind die folgenden expliziten Runge-Kutta-Verfahren, wobei fünf Runge-Kutta-Verfahren höherer Ordnung jeweils Fehlerabschätzung und adaptive Schrittweiten-Kontrolle unterstützen:
  - `Eul-Exp`: Explizites Euler-Verfahren mit konstanter Schrittweite in einem Durchlauf und ohne Fehlerabschätzung (Standard),
  - `Richard`: Explizites Euler-Verfahren mit einem Gesamtschritt und zwei Teilschritten sowie anschließender Richardson-Extrapolation. Dadurch ergibt sich, wie in Fellin, Mittendorfer u. a. (2009) aufgezeigt, ein relativ einfaches Verfahren, das sowohl eine Fehlerabschätzung als auch eine adaptive Kontrolle der Schrittweite zulässt,
  - `RK-S 23`: Runge-Kutta 2/3 (Simpsonregel),
  - `RK-BS23`: Runge-Kutta 2/3 (Bogacki-Shampine),
  - `RK-F 45`: Runge-Kutta 4/5 (Fehlberg),
  - `RK-DP45`: Runge-Kutta 4/5 (Dormand-Prince) und
  - `RK-CK45`: Runge-Kutta 4/5 (Cash-Karp).
- `setting_max_integration_steps` legt die maximale Anzahl an Iterationen in einem Durchgang fest (Vorgabewert: 50000). Die aktuelle Berechnung wird abgebrochen, wenn in der festgelegten Anzahl an Schritten nicht die Werte des vorgegebenen Zeitschritts in akzeptabler Genauigkeit erreicht werden,
- `setting_restrict_initial_substep` kann `.True.` (Standard) oder `.False.` sein. Bei Aktivierung wird das erste Subinkrement h_{sub} der numerischen Integration mit $h_{\text{sub}} = \min(h, t_s/||\mathbf{D}||)$ begrenzt. Anderenfalls entspricht das erste Subinkrement dem übergebenen Zeitinkrement. In diesem Fall führt das explizite Euler-Verfahren ohne Richardson-Extrapolation die Integration in einem einzigen Schritt durch.
- `setting_initial_substep_scale` ist der Steuerwert t_s für das erste Zeitinkrement der numerischen Integration (Vorgabewert: 10^{-5}). Das erste Zeitinkrement ist immer kleiner oder gleich diesem Wert, sofern `setting_restrict_initial_substep` aktiviert ist. Bei adaptiven Integrationsverfahren wird dieser Wert jeweils nach dem ersten Integrationsschritt ignoriert. Für die Integration wird das minimale Zeitinkrement indirekt über `setting_epsilon` vorgegeben.

Für die adaptiven Integrationsverfahren können folgende Parameter genutzt werden:

- `setting_stepsize_scaling_safety` gibt einen Sicherheitsfaktor S vor, mit dem der Wert des nächsten Zeitschritts bei adaptiven Integrationsverfahren wie in Gleichung (2.18) skaliert wird (Vorgabewert: 0,9),
- `setting_max_rel_error` definiert den maximalen relativen Fehler pro Iterationsschritt für adaptive Verfahren (Vorgabewert: 0,001). Der abgeschätzte Fehler wird komponentenweise skaliert und der größte Absolutwert muss kleiner sein als dieser Wert, ansonsten wird der aktuelle Iterationsschritt mit einem kleineren Zeitschritt wiederholt (siehe auch Abschnitt 4.2.3).
- `setting_max_integration_refinements` gibt die maximale Anzahl an Wiederholungen eines einzelnen Schritts an (Vorgabewert: 20). Falls der Fehler in einem Integrationsschritt zu groß ist, wird ein kleinerer Zeitschritt gewählt und der Schritt wiederholt. Das kann passieren, wenn der initiale Zeitschritt zu groß ist oder eine Unstetigkeit eine deutliche Verkleinerung der Schrittlänge erzwingt. Sollte das Ergebnis für den nächsten Zeitschritt mit einem akzeptablen Fehler nicht innerhalb von `setting_max_integration_refinements` Verfeinerungen berechnet werden können, wird die Berechnung abgebrochen.

Einstellungen zum Speichermanagement

Intern werden die folgenden Größen verwendet, um die maximale Anzahl an Statusvariablen und Materialparameter zu definieren:

- `setting_num_statevariables` reserviert die angegebene Anzahl an Statusvariablen für die Stoffmodelle (Vorgabewert: 20). Falls für einzelne Stoffmodelle mehr Statusvariablen benötigt werden, muss mit diesem Parameter global die Anzahl für alle Stoffmodelle angepasst werden.
- `setting_max_mat_params` reserviert die angegebene Anzahl an Materialparametern für die Stoffmodelle (Vorgabewert: 16). Falls für einzelne Stoffmodelle mehr Materialparameter benötigt werden, muss mit diesem Parameter global die Anzahl für alle Stoffmodelle angepasst werden.
- `setting_max_internal_states` gibt an, wie viele Einträge für den internen Zustandsvektor notwendig sind. Ähnlich wie in der Benutzerroutine von `ABQS02` wird in der xMat ein großer Vektor aus den Spannungsinkrementen, der Jacobi-Matrix und den Änderungen der Statusvariablen gebildet und für die numerische Integration genutzt. Dieser Wert berechnet sich aus der in der xMat-Entwicklungsumgebung gewählten Tensorrepräsentation (siehe Abschnitt 5.2) sowie `setting_num_statevariables` und sollte im Allgemeinen nicht geändert werden. Er entspricht entweder
  - `7*6 + 7*setting_num_statevariables` (Standard) für `MAT66` oder
  - `10*9 + 10*setting_num_statevariables` für `MAT99` und `TENS3333`.

Hilfsfunktionen

Einige Programme stellen eigene Funktionalitäten für Benutzerroutinen bereit, z. B. zur Ausgabe von Nachrichten oder zum Beenden der Berechnung. Um diese Funktionen im Rahmen der xMat zu nutzen und gleichzeitig die Unterstützung beliebiger anderer Schnittstellen zu ermöglichen, werden sie nur in einer einzigen Hilfsfunktion mit Präprozessor-Makros verwendet. Ein Aufruf erfolgt immer indirekt über die bereitgestellten Hilfsfunktionen. Dadurch müssen spezifische Funktionen neuer Schnittstellen oder die Änderung einer vorhandenen Definition nur an einer einzigen Stelle berücksichtigt werden, wie am Beispiel von `Write_Error_And_Exit()` in Code 4.2 zu sehen ist.

Code 4.2: An der dargestellten Hilfsfunktion `Write_Error_And_Exit()` kann gesehen werden, wie mit Präprozessor-Makros die passenden Funktionen abhängig von der verwendeten Schnittstelle aufgerufen werden. An jeder anderen Stelle außerhalb der Hilfsfunktion wird zum Beenden der Berechnung `Write_Error_And_Exit()` aufgerufen.

```
   ! ------------------------------------------------------------ !
   subroutine Write_Error_And_Exit(message)
   ! ------------------------------------------------------------ !
      character(len=*), intent(in) :: message
      ! ------------------------------------------------------------ !
      character(len=len(identifier_log_start) + len(message)) :: ErrorMessage

      ErrorMessage = identifier_log_start // message

      ! For Abaqus calls either stdb_abqerr (Abaqus/Standard) or xplb_abqerr (Abaqus/Explicit) is
      ! used. Besides a printf-ed message string, an int (array), a real (array) and a char (array)
      ! are expected - See Abaqus user routine guide
#ifdef ABQ_STD_CALLING
      call stdb_abqerr(-2, ErrorMessage, 0, 0.0, ' ')
      call xit()
#else
#ifdef ABQ_EXP_CALLING
      call xplb_abqerr(-2, ErrorMessage, 0, 0.0, ' ')
      call xplb_exit()
#else
#ifdef MATLAB_CALLING
      call mexErrMsgTxt(ErrorMessage)
#else
#ifdef PLAXIS_DLL
      write(1, *) ErrorMessage
      stop 'Execution aborted'
#else
#ifdef OCTAVE_CALLING
      write(0, *) ErrorMessage
      call xStopx('Execution aborted')
#else
      write(0, *) ErrorMessage
      stop 'Execution aborted'
#endif
#endif
#endif
#endif
#endif
   end subroutine Write_Error_And_Exit
```

4.2.2 Stoffmodelle (Modul `Constitutive_Models`)

Zu den Klassen der Stoffmodelle

Alle Stoffmodelle werden von der Basisklasse `Constitutive_Model_Baseclass` abgeleitet (siehe Bild 4.2). Damit können sowohl die Variablen und Funktionen der Basisklasse genutzt, als auch eigene Funktionalitäten bereitgestellt werden. Im einfachsten Fall benötigt ein Stoffmodell nur die Funktionen `Initialize()`, für die Zuweisung von Materialparametern und Variablen, und `Calculate_Dot_State()`, um die Materialantwort aus der übergebenen Konfiguration zu berechnen. Bei Bedarf können zusätzliche Funktionen in der jeweiligen Klasse definiert werden, wie bspw. für die Stoffmodelle `Hypo-VW96` und `ViHy-Ni03`. Ein Beispiel für die Struktur einer `Calculate_Dot_State()`-Funktion ist in Code 4.3 dargestellt.

Code 4.3: Die Grundstruktur der `Calculate_Dot_State()`-Funktion eines Stoffmodells. In Zeile 6 muss der tatsächliche Name des Stoffmodells eingetragen werden. Details zu den Makros in Zeile 13 und 14 finden sich in Abschnitt 5.2. Die Berechnung der Spannungsrate sowie Änderungen der Statusvariablen ist in Zeile 26–27 angedeutet. Die aktuellen Spannungen und Statusvariablen werden aus `cur_state` extrahiert (Zeile 22–23) und die Änderungen in `dot_state` gespeichert (Zeile 30–31).

```
1  ! ------------------------------------------------------------ !
2  function Calculate_Dot_State(this, ref_dt, cur_time, cur_state) result(dot_state)
3  ! ------------------------------------------------------------ !
4     use Math_Operations, only: Pack_States, Unpack_States
5     !
6     class(<class of this constitutive model>), intent(inout) :: this
7     real(dp), intent(in) :: ref_dt
8     real(dp), intent(in) :: cur_time
9     real(dp), dimension(setting_max_internal_states), intent(in) :: cur_state
10    real(dp), dimension(setting_max_internal_states) :: dot_state
11    ! ------------------------------------------------------------ !
12    real(dp), dimension(setting_num_statevariables) :: statevariables, dot_statevariables
13    real(dp), dimension(__tensor__) :: jac_stress, dot_jac_stress
14    real(dp), dimension(setting_num_statevariables, __matrix__) :: jac_statevariables, &
15                                                                   dot_jac_statevariables
16    dot_state = 0.0_dp
17    dot_jac_stress = 0.0_dp
18    dot_statevariables = 0.0_dp
19    dot_jac_statevariables = 0.0_dp
20
21    ! --- Unpacking of all states out of a long vector
22    call Unpack_States(input_states=cur_state, stress=cur_stress, jac_stress=jac_stress, &
23       statevariables=statevariables, jac_statevariables=jac_statevariables)
24
25    ! Implementation of constitutive model, i.e. at least
26    ! dot_stress = ...
27    ! dot_statevariables = ...
28
29    ! --- Packing all dot_states in a long vector
30    dot_state = Pack_States(stress=dot_stress, jac_stress=dot_jac_stress, &
31       statevariables=dot_statevariables, jac_statevariables=dot_jac_statevariables)
32 end function Calculate_Dot_State
```

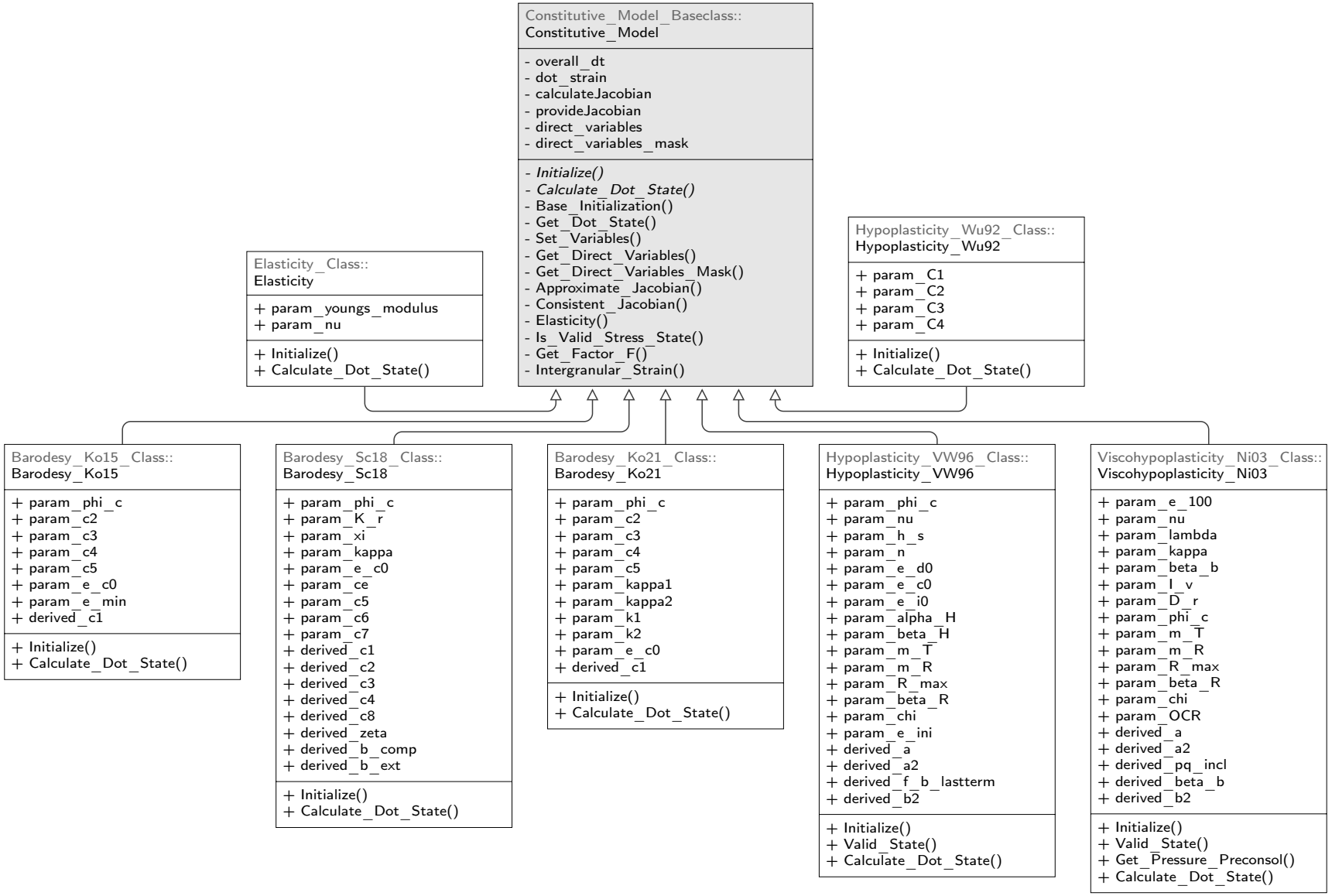


Bild 4.2: Alle Stoffmodelle sind von der Basisklasse `Constitutive_Model_Baseclass` abgeleitet. Sie müssen `Initialize()` und `Calculate_Dot_State()` definieren und können zusätzliche Variablen und Funktionen bereitstellen.

Bei manchen Stoffmodellen kann es bei der Integration aufgrund von sprungartigen Zustandsänderungen zu Problemen kommen, wenn ein Wert ohne zeitliche Abhängigkeit gespeichert werden soll. In diesen Fällen sollte der Wert nicht in `dot_state`, sondern an der gleichen Position in `direct_variables` gespeichert werden. Zusammen mit einer Anpassung der `direct_variables_mask` wird dem Integrationsverfahren mitgeteilt, dass dieser Wert nicht für die Fehlerabschätzung betrachtet werden soll (siehe Beispiel in Abschnitt 5.3).

Elastisches Ersatzmodell

Die hypoplastischen Stoffmodelle sind, wie die meisten Stoffmodelle für Böden, nur für Druckspannungen definiert, da in Böden ohne bindige Anteile, Teilsättigung oder Verkittung keine Zugspannung entstehen kann – die Bodenpartikel trennen sich voneinander und lagern sich um. In Simulationen wie FE-Analysen wird der Boden typischerweise nicht mehr durch einzelne Bodenpartikel, sondern als Kontinuum betrachtet. Dadurch wird das Verhalten durch die miteinander verbundenen Bodenelemente beeinflusst, an denen durch Verformungen Zugspannungen entstehen können. Deshalb muss darüber nachgedacht werden, wie das Verhalten bei ungültigen Spannungszuständen (Zugspannungen oder Spannungen nahe null) in FE-Analysen geregelt werden soll: Abbruch der Simulation oder Fortführung mit einer Ersatzlösung.

Für die beiden Stoffmodelle `Hypo-VW96` und `ViHy-Ni03` ist deshalb ein elastisches Ersatzmodell implementiert worden (ähnlich wie die Implementierung in `ABQS04`). Die Überprüfung, ob das Ersatzmodell genutzt werden soll, erfolgt in der Funktion `Is_Valid_Stress_State()`. Dort wird aktuell die Spur des Spannungstensors betrachtet und mit einem frei definierten Grenzwert `setting_very_small_stress` verglichen. Sofern eine kleinere Druckspannung oder sogar eine Zugspannung vorliegt, kommt das Ersatzmodell zum Einsatz.

Allein mit diesem Unterscheidungskriterium werden viele ungültige Spannungszustände nicht berücksichtigt. Geschickter wäre eine absolute Einschränkung auf den Oktant der Druckspannungen zusammen mit der Überprüfung der jeweils gültigen Grenzbedingung.

Wenn ein Ersatzmodell zum Einsatz kommt, muss auch geklärt werden, was mit den Statusvariablen, wie Porenzahl oder intergranularer Dehnung, geschieht: Eine Option ist, sie als konstant anzunehmen, bis wieder ein gültiger Spannungszustand erreicht wird. Eine andere Option ist, sie beim Erreichen eines gültigen Spannungszustands neu zu initialisieren, z. B. die Porenzahl in der lockersten Lagerung anzunehmen und die intergranularen Dehnungen auf null zu setzen.

Numerische Berechnung der Jacobi-Matrix

Manche Schnittstellen erfordern die Berechnung der Jacobi-Matrix in der Benutzerroutine. Wenn innerhalb eines Stoffmodells explizit ein Materialtensor $\mathcal{C}$ aufgestellt werden kann, der die Gleichung $\mathring{\mathbf{T}} = \mathcal{C} : \mathbf{D}$ erfüllt, entspricht $\mathcal{C}$ der Jacobi-Matrix $\mathcal{J}$. Anderenfalls kann es notwendig sein, eine numerische Approximation der Jacobi-Matrix zu erzeugen. Die Umsetzung in der xMat orientiert sich an Fellin und Ostermann (2002) sowie der Implementierung in der Benutzerroutine `ABQS02` und ist in Bild 4.3 dargestellt.

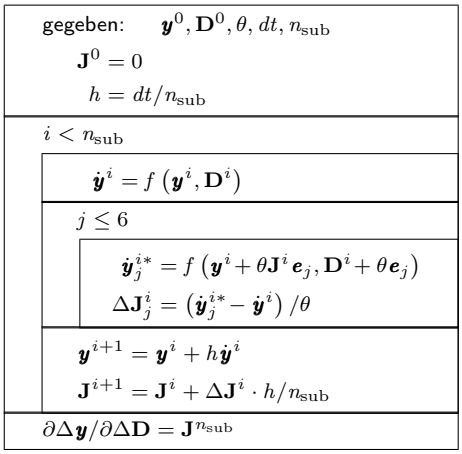


Bild 4.3: Berechnung der Jacobi-Matrix basierend auf der Implementierung in Benutzerroutine `ABQS02`. Durch Störung der Eingangsgrößen mit θ wird die Sensitivität der Ausgangsgrößen (d. h. die partiellen Ableitungen $\partial \Delta \mathbf{y}/\partial \Delta \mathbf{D}$) abgeschätzt. Dadurch sind mindestens sechs zusätzliche Auswertungen von `Calculate_Dot_State()` erforderlich.

4.2.3 Integrationsverfahren (Modul `Solver`)

Prinzipiell ist die Struktur der Integrationsverfahren wie bei den Stoffmodellen: Es gibt eine Basisklasse (hier: `Solver_Baseclass`), von der alle anderen Klassen abgeleitet werden. Die numerische Integration der Stoffmodelle wird in der Funktion `Integrate()` gesteuert und ist in Bild 4.4 dargestellt. Die einzelnen Integrationsverfahren stellen jeweils eine Funktion `Integration_Algorithm()` bereit, die von `Integrate()` aufgerufen wird.

Nach der Initialisierung von Rückgabewerten und internen Variablen sowie der Überprüfung der übergebenen Schrittgröße `stepsize`, wird `idx_integloop = 0` gesetzt und die Integrationsschleife aufgerufen. In jeder Iteration wird der Zähler `idx_integloop` um einen Wert erhöht und am Ende der Schleife mit `setting_max_integration_steps` verglichen. Falls mehr Schritte benötigt werden (`idx_integloop >= setting_max_integration_steps`), wird die Integration abgebrochen.[9]

Zu Beginn der Integrationsschleife wird mit `this%next_dot_state_assigned` überprüft, ob eine Funktionsauswertung für den ersten Zwischenschritt existiert. Die Funktionsauswertung liegt typischerweise für Integrationsverfahren mit FSAL-Eigenschaft nach der ersten Iteration vor. Anderenfalls findet eine explizite Auswertung mit `Get_Dot_State()` statt. Anschließend werden skalierte Referenzwerte zum Abschätzen des Fehlers bestimmt (`state_scaled_err`), der Zähler der Konvergenzschleife `idx_convloop` zurückgesetzt und diese Schleife gestartet.

In der Konvergenzschleife wird mit `this%Integration_Algorithm()` das gewählte Integrationsverfahren aufgerufen und eine Approximation für das aktuelle Integral `new_state` bestimmt. Bei konstanter Schrittgröße ohne Abschätzung des Integrationsfehlers wird die

[9] Bei einer Integration mit konstanter Schrittgröße wird vor der Integrationsschleife geprüft, ob die Berechnung in unter `setting_max_integration_steps` durchgeführt werden kann und die Berechnung ggf. schon dort abgebrochen.

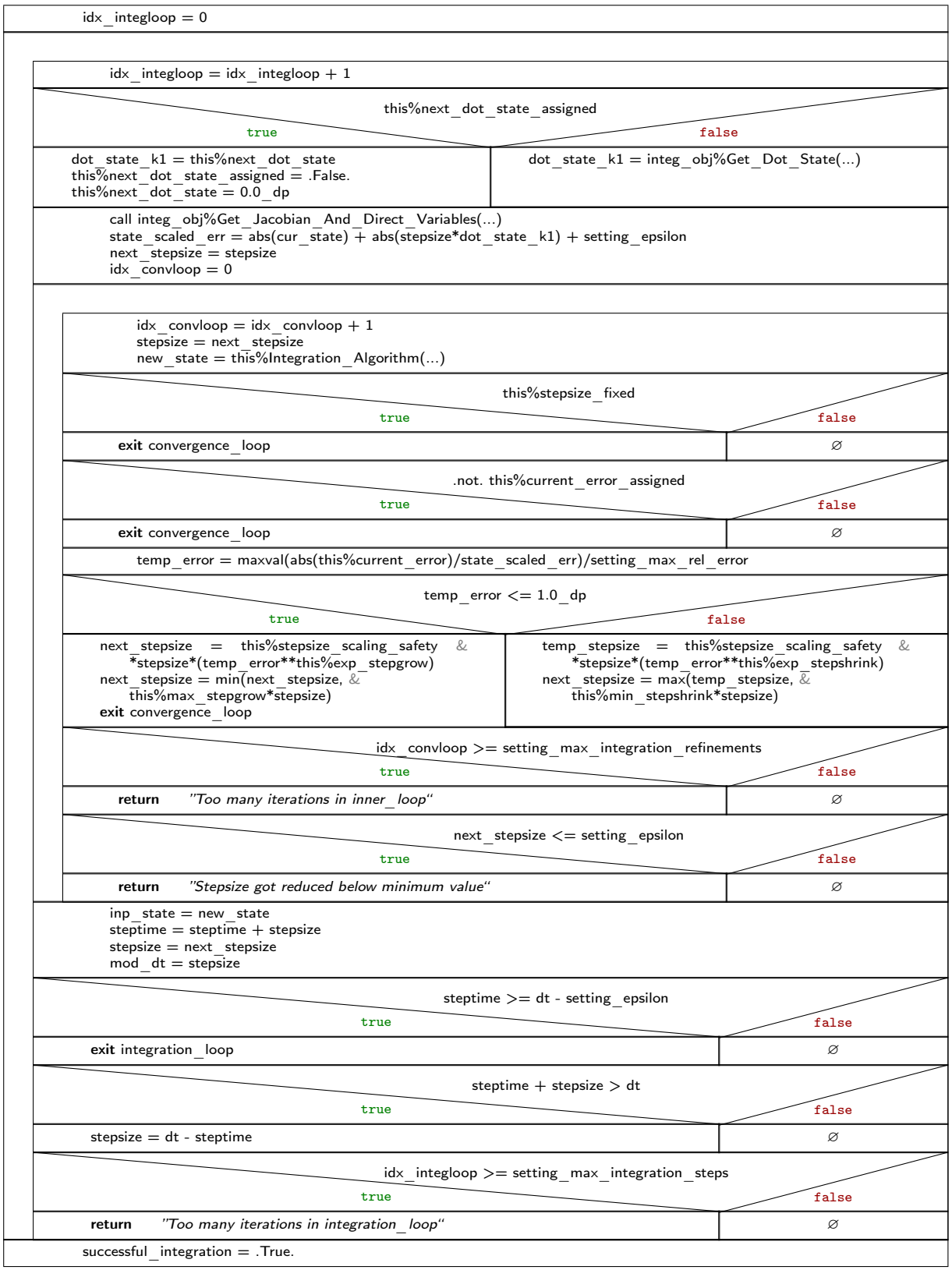


Bild 4.4: Der Hauptteil der Integrationsschleife der xMat in der `Integrate()`-Funktion als Struktogramm nach Nassi-Shneiderman.

Konvergenzschleife verlassen, anderenfalls wird über die abgeschätzten Integrationsfehler ein Faktor `temp_error` bestimmt. Ist der Faktor kleiner als eins (Konvergenz), so wird der Integrationsschritt akzeptiert, über den Faktor `temp_error` eine neue Schrittgröße `next_stepsize` bestimmt und die Konvergenzschleife verlassen. Anderenfalls wird die Schrittgröße verkleinert und eine weitere Iteration der Konvergenzschleife beginnt. Wie in der Integrationsschleife wird auch in der inneren Schleife ein Zähler verwendet (hier: `idx_convloop`). Falls keine Konvergenz nach einer vorgegebenen Anzahl direkt aufeinander folgender Verkleinerungen (`setting_max_integration_refinements`) erreicht wird oder die nächste Schrittgröße unter eine vorgegebene Grenze sinkt (aktuell `setting_epsilon`), wird die Integration abgebrochen.

Zu den Klassen der Integrationsverfahren

Dieses Modul verbindet die verfügbaren Integrationsverfahren mit einer gemeinsamen Steuerungsfunktion für die numerische Integration der Stoffmodelle. Anders als bei den Stoffmodellen muss das aktive Integrationsverfahren in der vorliegenden Implementierung im Modul `General_Settings` festgelegt werden (mit `setting_solver_default`, siehe auch Abschnitt 4.2.1) und kann nicht durch ein Übergabeargument beim Aufruf verändert werden. Die implementierten Integrationsverfahren sind die in Tabelle 4.1 aufgelisteten expliziten Runge-Kutta-Verfahren mit den in Tabelle 4.2 aufgelisteten Faktoren für die adaptive Regelung der Schrittweite.

Tabelle 4.1: Auflistung der expliziten Runge-Kutta-Integrationsverfahren, die in der xMat implementiert sind. Abgesehen vom expliziten Euler-Verfahren wird bei allen Verfahren auch eine Lösung höherer Ordnung bestimmt, sodass der Integrationsfehler abgeschätzt werden kann. Manche der Integrationsverfahren haben ein eingebettetes Verfahren anderer Ordnung zur Fehlerschätzung. Verfahren mit der FSAL-Eigenschaft können die letzte Auswertung als ersten Schritt im neuen Zeitinkrement wiederverwenden und somit einen Rechenschritt sparen (siehe auch Abschnitt 2.4).

Integrationsverfahren	Kürzel	Eingebettet	FSAL
Euler-Explizit	`Eul-Exp`	nein	nein
Euler-Explizit mit Richardson Extrapolation	`Richard`	nein	nein
Simpson-Verfahren (Runge-Kutta 2/3)	`RK-S 23`	nein	nein
Bogacki-Shampine (Runge-Kutta 2/3)	`RK-BS23`	ja	ja
Runge-Kutta-Fehlberg (Runge-Kutta 4/5)	`RK-F 45`	ja	nein
Cash-Karp (Runge-Kutta 4/5)	`RK-CK45`	ja	nein
Dormand-Prince (Runge-Kutta 4/5)	`RK-DP45`	ja	ja

Tabelle 4.2: Faktoren für die adaptive Regelung der Schrittweite der verschiedenen Integrationsverfahren. Die Einschätzung der Werte basiert auf Gleichung (2.18) und den Empfehlungen aus Press u. a. (1997), S. 710–712.

Kürzel	exp_stepgrow	exp_stepshrink	max_stepgrow	min_stepshrink
Eul-Exp	—	—	—	—
Richard	−0,5	−0,5	2	0,2
RK-S 23	−1/3	−0,5	4	0,2
RK-BS23	−1/3	−0,5	3	0,2
RK-F 45	−0,2	−0,25	5	0,1
RK-CK45	−0,2	−0,25	5	0,1
RK-DP45	−0,2	−0,25	5	0,1

4.3 Anbindung an andere Programme

Um die implementierten Stoffmodelle möglichst gut nutzen zu können, gibt es bei der xMat zwei Arten von Schnittstellen:

- Schnittstellen mit Integration der Stoffgleichungen über den angegebenen Zeitraum berechnen den neuen Spannungszustand und die dazugehörigen Statusvariablen, was typischerweise von FE-Programmen gefordert wird. Die vorgestellten Schnittstellen `UMAT()` und `VUMAT()` für Abaqus, `xmat_console()` für Fortran, `mexFunction()` für MATLAB, `USER_MOD()` für Plaxis und `xmat_console_CInter()` für C gehören zu dieser Gruppe.[10]
- Schnittstellen ohne Integration berechnen lediglich die Antwort des Stoffmodells aufgrund der übergebenen Werte (Spannungsrate, Änderung der Statusvariablen und Jacobi-Matrix). Diese Schnittstellen eignen sich vor allem, wenn die Integration vom aufrufenden Programm durchgeführt wird, wie z. B. in Simulink. Die dazugehörige Schnittstelle für Fortran ist `Dot_Values()` und für C `Dot_Values_CInter()`.

Die Schnittstellen mit Integration sind nach dem folgenden Muster aufgebaut:

- Zuweisung und ggf. Anpassung der Übergabewerte an Variablen der Schnittstelle in der mit `dp` im Modul `General_Settings` definierten Genauigkeit,
- Vorverarbeitung der Statusvariablen durch `Preprocess_State_Matrices()`
- Durchführung der eigentlichen Berechnung. Wie in Abschnitt 4.1 beschrieben, werden die internen Abläufe durch die Funktionen `Xmat_Initialize()`, `Import_State()` und `Calculate_Results()` gesteuert,
- Nachbereitung der Statusvariablen in `Postprocess_State_Matrices()` und
- Zuweisung und ggf. Anpassung der Rückgabewerte an die passenden Übergabewerte der Schnittstelle.

Im Gegensatz dazu wird bei Schnittstellen ohne Integration die Funktion `Dot_Results()`

[10] Manche Schnittstellen sind auch von anderen Programmen nutzbar (bspw. `UMAT` mit Tochnog), wobei der Hauptfokus auf der Funktionalität mit den erwähnten Programmen liegt.

anstelle von `Calculate_Results()` aufgerufen werden. Ein konkretes Beispiel einer Schnittstelle findet sich in Abschnitt 5.5 mit der Beschreibung, wie zusätzliche Schnittstellen zur xMat hinzugefügt werden können.

4.3.1 Schnittstellen zu Abaqus

Das FE-Programm Abaqus biete viele Möglichkeiten, eigene Anpassungen in Form von Benutzerroutinen einzubringen (siehe Dassault Systèmes, 2015; oder Dassault Systèmes, 2020, für eine Übersicht der verfügbaren Schnittstellen). Im Rahmen dieser Arbeit sind vor allem die beiden Schnittstellen von Interesse, die das mechanische Materialverhalten steuern: die `UMAT`-Schnittstelle für Abaqus/Standard und die `VUMAT`-Schnittstelle für Abaqus/Explizit.

UMAT-Schnittstelle

Für Benutzerroutinen über die `UMAT`-Schnittstelle werden u. a. die folgenden Größen am Anfang des aktuellen Zeitschritts erwartet (siehe auch die Signatur in Code 4.4):

Code 4.4: `UMAT`-Schnittstelle für Abaqus 6.14 bis (mindestens) 2021 basierend auf Dassault Systèmes (2015) und Dassault Systèmes (2020).

```
subroutine UMAT( &
   ! ==================== user coding to define ==================== !
      stress, statev, ddsdde, sse, spd, scd, &
   ! ====================== and if necessary ====================== !
      rpl, ddsddt, drplde, drpldt, &
   ! ============= variables passed in for information ============= !
      stran, dstran, time, dtime, temp, dtemp, predef, dpred, cmname, &
      ndi, nshr, ntens, nstatv, props, nprops, &
      coords, drot, &
   ! ================ variables that can be updated ================ !
      pnewdt, &
   ! ========== (more) variables passed in for information ========= !
      celent, dfgrd0, dfgrd1, &
      noel, npt, layer, kspt, jstep, kinc)

   include 'aba_param.inc'

   dimension :: statev(nstatv)
   dimension :: stress(ntens), ddsddt(ntens), drplde(ntens), stran(ntens), dstran(ntens)
   dimension :: ddsdde(ntens, ntens)
   dimension :: time(2)
   dimension :: predef(1), dpred(1)
   dimension :: props(nprops)
   dimension :: coords(3)
   dimension :: drot(3, 3), dfgrd0(3, 3), dfgrd1(3, 3)
   dimension :: jstep(4)

   character(len=80) :: cmname

   ! user coding
end subroutine
```

- Cauchy-Spannungen `stress` im mitgedrehten Koordinatensystem in der Reihenfolge σ_{11}, σ_{22}, σ_{33}, σ_{12}, σ_{13} und σ_{23},
- mechanische Dehnungsinkremente `dstran` in der Reihenfolge ε_{11}, ε_{22}, ε_{33}, γ_{12}, γ_{13}, γ_{23} (mit $\gamma_{xy} = \varepsilon_{xy} + \varepsilon_{yx}$),
- die Zeit seit Beginn des aktuellen Simulationsschritts `time(1)` sowie der gesamten Simulation `time(2)` und das aktuelle Zeitinkrement `dtime`,
- die Statusvariablen `statev`, wobei tensorielle Größen in den Statusvariablen manuell in der Benutzerroutine mit `drot` gedreht werden müssen,
- die Materialparameter `props`,
- der Materialname `cmname` und
- eine inkrementelle Rotationsmatrix `drot`. Die übergebenen Dehnungs- und Spannungskomponenten sind bereits mit `drot` gedreht worden.

Aus den übergebenen Werten werden über die `UMAT`-Schnittstelle insbesondere die folgenden Größen am Ende des aktuellen Zeitinkrements berechnet:

- die Cauchy-Spannungen am Ende des Zeitschritts in `stress`,
- die Statusvariablen am Ende des Zeitschritts in `statev` und
- die Jacobi-Matrix $\mathcal{J}$ des Stoffmodells in `ddsdde`.

In der Benutzerroutine muss für finite Dehnungen die Jacobi-Matrix

$$\mathcal{J} = \frac{1}{\det(\mathbf{F})} \frac{\partial \Delta \mathbf{T}^K}{\partial \Delta \boldsymbol{\varepsilon}}$$

mit den Kirchhoff-Spannungsinkrementen $\Delta \mathbf{T}^K$, dem Deformationsgradienten $\mathbf{F}$ und Dehnungsinkrementen $\Delta \boldsymbol{\varepsilon}$ berechnet werden. Für kleine Volumenänderungen kann wegen $\det(\mathbf{F}) \approx 1$ die Jacobi-Matrix nach Gleichung (3.1) mit den Cauchy-Spannungsinkrementen $\Delta \boldsymbol{\sigma}$ angenähert werden (siehe Dassault Systèmes, 2020).

VUMAT-Schnittstelle

Über die `VUMAT`-Schnittstelle werden etwas andere Größen erwartet, u. a. (siehe auch Code 4.5):

- Cauchy-Spannungen `stressOld` im mitgedrehten Koordinatensystem in der Reihenfolge σ_{11}, σ_{22}, σ_{33}, σ_{12}, σ_{23} und σ_{13},
- Tensoreinträge der Dehnungsinkremente `strainInc` in der Reihenfolge ε_{11}, ε_{22}, ε_{33}, ε_{12}, ε_{23}, ε_{13},
- die Zeit seit Beginn des aktuellen Simulationsschritts `stepTime` sowie der gesamten Simulation `totalTime` und das aktuelle Zeitinkrement `dt`,
- die Statusvariablen in `stateOld` in den gleichen Koordinaten wie die Spannungen,

- die Materialparameter `props`,
- der Materialname `cmname` und
- eine inkrementelle Rotationsmatrix `relSpinInc` als $\Delta\mathbf{W} - \Delta\boldsymbol{\Omega}$.

Dabei sind alle tensoriellen Größen, wie Spannungen, Dehnungsinkremente und Matrizen in den Statusvariablen, in Koordinaten des lokalen Koordinatensystems des Materials und müssen ggf. in der `VUMAT` gedreht werden. In der aktuellen Implementierung der `VUMAT`-Schnittstelle werden keine Drehungen der genannten Größen durchgeführt.

Aus den übergebenen Werten werden über die `VUMAT`-Schnittstelle insbesondere die folgenden Größen am Ende des aktuellen Zeitinkrements berechnet:

- die Cauchy-Spannungen `stressNew` und
- die Statusvariablen `stateNew`.

Code 4.5: `VUMAT`-Schnittstelle für Abaqus 6.11 bis 2019 basierend auf Dassault Systèmes (2015) und Dassault Systèmes (2020).

```
subroutine VUMAT( &
   ! ========= (scalar) variables passed in for information ======== !
      nblock, ndir, nshr, nstatev, nfieldv, nprops, lanneal, &       ! integer
      stepTime, totalTime, dt, &                                     ! real
   ! ========= (other) variables passed in for information ========= !
      cmname, &                                                      ! character
      coordMp, charLength, props, density, strainInc, relSpinInc, &
      tempOld, stretchOld, defgradOld, fieldOld, stressOld, stateOld, enerInternOld, enerInelasOld, &
      tempNew, stretchNew, defgradNew, fieldNew, &
   ! ==================== user coding to define ==================== !
      stressNew, stateNew, &
   ! ================ variables that can be updated ================ !
      enerInternNew, enerInelasNew)

   include 'vaba_param.inc'

   dimension :: props(nprops)
   dimension :: density(nblock), charLength(nblock), tempOld(nblock), tempNew(nblock), &
                enerInternOld(nblock), enerInternNew(nblock), &
                enerInelasOld(nblock), enerInelasNew(nblock)
   dimension :: coordMp(nblock, *)
   dimension :: relSpinInc(nblock, nshr)
   dimension :: strainInc(nblock, ndir+nshr), &
                stretchOld(nblock, ndir+nshr), stretchNew(nblock, ndir+nshr), &
                stressOld(nblock, ndir+nshr), stressNew(nblock, ndir+nshr)
   dimension :: defgradOld(nblock, ndir+nshr+nshr), defgradNew(nblock, ndir+nshr+nshr)
   dimension :: fieldOld(nblock, nfieldv), fieldNew(nblock, nfieldv)
   dimension :: stateOld(nblock, nstatev), stateNew(nblock, nstatev)

   character(len=80) :: cmname

   do km = 1, nblock
      ! user coding
   end do
end subroutine
```

Konsistenz der Schnittstellendefinition

Es sollte beachtet werden, dass sich die Schnittstellendefinitionen in Abaqus ändern können. So ist die angegebene UMAT-Schnittstelle seit Abaqus 6.14 gültig, da die vektorielle Größe `jstep(4)` vorher ein Einzelwert namens `kstep` gewesen ist.[11]

Die VUMAT-Schnittstelle hat sich zwischen Version 6.6 (2006) und 2019 nicht verändert. Seit Abaqus 2020 wird statt des skalaren Werts `lanneal`, der anzeigt, ob ein Glühprozess vorliegt, eine vektorielle Größe `jInfoArray(*)` übergeben. Außerdem wird statt des Einzelwerts (real) `dt`, der die Länge des aktuellen Zeitschritts angibt, nun ein real-Array `dtArray(2*(nblock)+1)` bereitgestellt. In beiden Fällen entspricht der erste Eintrag weiterhin dem Wert aus der alten Schnittstellendefinition.

Durch die Abwärtskompatibilität der neuen Schnittstellendefinitionen können auch Benutzerroutinen mit einer älteren UMAT- bzw. VUMAT-Schnittstelle in aktuellen Abaqus-Versionen genutzt werden. In der xMat sind die Schnittstellen wie in Code 4.4 und 4.5 dargestellt implementiert, sodass eine Kompatibilität bis mindestens Abaqus 6.14 besteht.

Nutzung der xMat in Abaqus

In Abaqus können Benutzerroutinen entweder vorkompiliert und als Bibliotheken eingebunden, oder für die Analyse zum Kompilieren übergeben werden. Zum Vorkompilieren kann `abaqus make library=xmat.f` genutzt werden. Für das direkte Einbinden bei der Analyse muss `user=xmat.f` in den Abaqus-Aufruf integriert werden (siehe Dassault Systèmes, 2020). In beiden Fällen muss Abaqus mit einem Fortran-Compiler verknüpft sein, was z. B. mit `abaqus verify -all` überprüft werden kann.

Standardmäßig erwartet Abaqus Fortran-Dateien im sog. `fixed`-Format (siehe Abschnitt 3.2), was über einen passenden Schalter in einer Umgebungsdatei namens `abaqus_v6.env` angepasst werden kann. Ebenso muss für die xMat das Hilfsmakro `ABAQUS_CALLING` für die Nutzung aus Abaqus definiert sein, was ebenfalls in der Umgebungsdatei eingetragen werden kann. Am einfachsten erscheint es, für die Nutzung der xMat im aktuellen Arbeitsverzeichnis eine Datei namens `abaqus_v6.env` mit einem Inhalt ähnlich zu Code 4.6 zu erstellen.[12]

Seit Abaqus 2020 werden automatisch die Präprozessor-Makros `ABQ_FORTRAN` – und abhängig von der Architektur entweder `ABQ_LNX86_64` oder `ABQ_WIN86_64` – definiert, die für den gleichen Zweck wie das eigens definierte `ABAQUS_CALLING` genutzt werden können. Wenn keine Kompatibilität zu älteren Abaqus-Versionen erforderlich sein sollte, könnte in der xMat `ABQ_FORTRAN` statt `ABAQUS_CALLING` zum Erkennen und Verarbeiten eines Abaqus-Aufrufs genutzt werden.

[11] `jstep(1)` entspricht aus Kompatibilitätsgründen weiterhin der aktuellen Schrittnummer `kstep`, aber die anderen drei Einträge enthalten noch weitere Informationen zum aktuellen Berechnungsschritt.

[12] Die tatsächlichen Argumente unterscheiden sich abhängig von Abaqus-Version und Compiler(-version) und sind in der Systemdatei definiert (`<Abaqus-install-dir>/linux_a64/SMA/site/lnx86_64.env` oder `<Abaqus-install-dir>/win_b64/SMA/site/win86_64.env`). Wichtig ist die lokale Anpassung der `compile_fortran`-Anweisung.

Code 4.6: Verwendete Umgebungsdatei `abaqus_v6.env` im aktuellen Arbeitsverzeichnis für Abaqus 2018 mit dem Intel-Compiler. Damit die xMat genutzt werden kann, muss in der `compile_fortran`-Anweisung `-DABAQUS_CALLING` und entweder `-free`, `-fpp` (für den Intel-Compiler) oder `-ffree-form`, `-cpp` (für den GNU Fortran-Compiler) definiert werden. Unter Windows werden die Argumente mit einem Schrägstrich eingeleitet (z. B. `/free`).

```
import os
abaHomeInc = os.path.abspath(os.path.join(os.environ.get('ABA_HOME', ''), os.pardir))

compile_fortran = ['ifort',
                   '-V',
                   '-free', '-DABAQUS_CALLING', # <-- this line is important for use with xMat
                   '-c', '-fPIC', '-auto', '-mP2OPT_hpo_vec_divbyzero=F', '-extend_source',
                   '-fpp', '-WB', '-I%I',
                   '-I'+abaHomeInc, '%P',
                   ]
del abaHomeInc
```

Im Gegensatz dazu scheint es nicht möglich zu sein, in der Benutzerroutine an dem von Abaqus initiierten Kompiliervorgang automatisch zu erkennen, ob die Benutzerroutine für Abaqus/Standard (`UMAT`-Schnittstelle) oder Abaqus/Explizit (`VUMAT`-Schnittstelle) kompiliert werden soll. Deshalb befindet sich, wie in Code 4.7 dargestellt, in der xMat ein Schalter, um je nach verwendeter Schnittstelle die Funktionen für Abaqus/Standard mit `ABQ_STD_CALLING` oder Abaqus/Explizit mit `ABQ_EXP_CALLING` zu nutzen. Ist keines der beiden Makros definiert (oder das falsche), kommt es i. d. R. zu Abbrüchen beim Kompilieren und Verlinken oder beim Start einer Simulation (siehe auch Anhang D).

Die xMat respektiert die von Abaqus erwartete Präzision und bindet dazu entweder die Datei `aba_param.inc` (Abaqus/Standard) oder `vaba_param.inc` (Abaqus/Explizit) ein, um implizite Variablendefinitionen in der `UMAT`- bzw. `VUMAT`-Schnittstelle zu erlauben (siehe Dassault Systèmes, 2020). Seit Abaqus 2018 unterscheidet Abaqus intern zwischen `vaba_param_sp.inc` und `vaba_param_dp.inc`. Es muss beachtet werden, dass beide Dateien in Abaqus 2018 und 2019 Kommentare im `fixed`-Format enthalten, die ohne Anpassung nicht in eine Benutzerroutine im `free`-Format eingebunden werden können.[13]

Code 4.7: Aktuell muss in der xMat durch `ABAQUS_CALLING` entweder, wie dargestellt, `ABQ_STD_CALLING` (für `UMAT`-Schnittstelle mit Abaqus/Standard) oder `ABQ_EXP_CALLING` (für `VUMAT`-Schnittstelle mit Abaqus/Explizit) definiert werden.

```
#ifdef ABAQUS_CALLING
#define ABQ_STD_CALLING
#endif
```

[13] In/bis Abaqus 6.14-2 gibt es dort noch keine Kommentare und seit Abaqus 2020 werden Kommentare mit Ausrufezeichen eingeleitet, die sowohl im `fixed`- als auch im `free`-Format unterstützt werden. Für Abaqus 2018 und 2019 müssen diese Dateien angepasst werden (Kommentare mit `!` statt `c`).

Es sei noch angemerkt, dass es für Simulationen in Abaqus mindestens zwei Möglichkeiten zur Initialisierung von Statusvariablen gibt: Anfangslösungen können im Modell mit `*INITIAL CONDITION, TYPE=SOLUTION` definiert oder über eine Benutzerroutine `SDVIni.f` initialisiert werden.

Verfügbare Hilfsfunktionen in Abaqus

Abaqus bietet mehrere Hilfsfunktionen an, die von Benutzerroutinen genutzt werden können. Manche sind nur für die `UMAT`-Schnittstelle nutzbar, manche nur für die `VUMAT`-Schnittstelle. Tabelle 4.3 zeigt eine Auswahl an Hilfsfunktionen für die Programmierung einer `UMAT` bzw. `VUMAT`.

Wie bereits in Abschnitt 4.2.1 beschrieben, werden schnittstellenabhängige Funktionen in der xMat nur indirekt aufgerufen (siehe auch Code 4.2). Andere Benutzerroutinen mit hypoplastischen Stoffmodellen und `UMAT`-Schnittstelle nutzen die von Abaqus bereitgestellten Hilfsfunktionen, z. B. `SPrinc()` zum Ermitteln der Eigenwerte einer Matrix und `RotSig()` zum Drehen von Matrizen. Diese Funktionalitäten stellt die xMat durch eigene Funktion bereit: `Eigendecomposition()` zum Ermitteln der Eigenwerte und Eigenvektoren sowie `Matrix_Rotation()` zum Drehen von Matrizen.

4.3.2 Schnittstelle zu Tochnog

Tochnog Professional[14] bietet ein Paket für Linux an, mit dem auch eigene Benutzerroutinen mit der beschriebenen `UMAT`-Schnittstelle genutzt werden können.[15] Für die Bereitstellung einer Benutzerroutine in Tochnog ist ein Vorkompilieren nötig. Im `Makefile` von Tochnog Professional Linux User Supplied muss eine Zeile für die xMat angepasst werden (siehe Code 4.8). Nach dem Kompilieren und der Anpassung des Modells kann die Datei `tochnog_user_supplied` ausgeführt werden.

Es wird die Schnittstellendefinition der `UMAT` aus Abaqus vor Version 6.14-2 erwartet, d. h. `integer kstep` statt `dimension(4) jstep` (vgl. Code 4.4). Außerdem kann das Stoffmodell

Code 4.8: Im Vergleich zur beiliegenden `Makefile`-Datei muss Zeile 4 geändert werden, sodass die xMat erfolgreich kompiliert werden kann (`-ffree-form` und `-cpp`) sowie Tochnog-spezifische Einstellungen vornimmt (`-DTOCHNOG_CALLING`). Zeile 5 ist optional.

```
1 default:
2       g++ -fPIC -c user.cpp;
3       gcc -shared -Wl,-soname,user.so -o user.so user.o;
4       gfortran -ffree-form -cpp -DTOCHNOG_CALLING -finit-local-zero -fPIC -c xmat.f -o umat.o;
5       -rm *.mod;
6       gcc -shared -Wl,-soname,umat.so -o umat.so umat.o;
```

[14] Tochnog Professional Linux User Supplied »https://www.tochnogprofessional.nl/download/page.php

[15] Falls eine Unterscheidung zwischen dem Aufruf der `UMAT`-Schnittstelle aus Abaqus/Standard oder Tochnog erforderlich sein sollte, kann die Variable `internname` mit einem Präprozessor-Makro für Abaqus/Standard weiterhin als `umat_abq_std` und für Tochnog als `umat_tochnog` definiert werden.

Tabelle 4.3: Auswahl einiger von Abaqus bereitgestellter Hilfsfunktionen, die aus einer Benutzerroutine mit UMAT-Schnittstelle (Abaqus/Standard) oder VUMAT-Schnittstelle (Abaqus/Explizit) aufgerufen werden können (nach Dassault Systèmes, 2020). Einige der aufgelisteten Hilfsfunktionen können von mehr Abaqus-Schnittstellen als nur UMAT und VUMAT genutzt werden. Die mit * markierten Funktionsaufrufe sind in Abaqus/Standard und Abaqus/Explizit gleich.

Aufgabe	Abaqus/Standard	Abaqus/Explizit
Umgebungsvariablen ermittlen	`call GetEnvVar(EnvVarName,EnvVar,LEnvVar)`	`call VGetEnvVar(EnvVarName,EnvVar,LEnvVar)`
Jobname ermitteln	`call GetJobName(JobName,LenJobName)`	`call VGetJobName(JobName,LenJobName)`
Ausgabeordner ermitteln	`call GetOutDir(OutDir,LenOutDir)`	`call VGetOutDir(OutDir,LenOutDir)`
Anzahl an Prozessen	`call GetNumCPUs(NumProcesses)`	`call VGetNumCPUs(NumProcesses)`
Prozessnummer ermitteln	`call GetRank(KProcessNum)`	`call VGetRank(KProcessNum)`
*Anzahl an Threads ermitteln	`GetNumThreads()`	`GetNumThreads()`
*Thread-ID ermitteln	`Get_Thread_Id()`	`Get_Thread_Id()`
*Kommunikator für Arbeiter	`GetCommunicator()`	`GetCommunicator()`
Part-Informationen ermitteln	`call GetPartInfo(IntNum,JTyp,CPName,LocNum,JRCD)`	`call VGetPartInfo(IntNum,JTyp,CPName,LocNum,JRCD)`
Node-/Elementnummer ermitteln	`call GetInternal(CPName,LocNum,JTyp,IntNum,JRCD)`	`call VGetInternal(CPName,LocNum,JTyp,IntNum,JRCD)`
Phys. Konstanten ermitteln	`call GetPhysicalConstant(jKey,Value,jError)`	—
Spannungsinvarianten ermitteln	`call SInv(Stress,SInv1,SInv2,nDi,nShr)`	—
Hauptspannungen ermitteln	`call SPrinc(S,PS,LStr,nDi,nShr)`	`call VSPrinc(nBlock,S,EigVal,nDir,nShr)`
Hauptspg. u. Richtung ermitteln	`call SPrinD(S,PS,AN,LStr,nDi,nShr)`	`call VSPrinD(nBlock,S,EigVal,EigVec,nDir,nShr)`
Rotieren eines Tensors	`call RotSig(S,R,SPrime,LStr,nDi,nShr)`	—
Nachrichtenausgabe	`call Stdb_ABQerr(LOP,String,IntV,RealV,CharV)`	`call Xplb_ABQerr(LOP,String,IntV,RealV,CharV)`
Analyse beenden	`call Xit`	`call Xplb_Exit`

nicht durch Übergabe eines Materialnamens `cmname` ausgewählt werden. Deshalb wird in der xMat im Modul `General_Settings` ein Präprozessor-Makro `TOCHNOG_MATERIALNAME` mit dem Materialnamen definiert, wenn beim Kompilieren `TOCHNOG_CALLING` genutzt wird. Tochnog übergibt beim Aufruf immer einen dreidimensionalen Spannungszustand mit sechs Komponenten (`ntens=6`).

4.3.3 Fortran-Schnittstelle

Die `UMAT`-Schnittstelle kann auch von anderen Fortran-Programmen verwendet werden, sodass eine eigene Fortran-Schnittstelle nicht zwingend erforderlich ist. Gleichzeitig ist eine zweite, kompaktere Schnittstelle nützlich zum Testen und Entwickeln einzelner Komponenten. Auf diesem Weg sind `xmat_console` als Schnittstelle mit Integration und `Dot_Values` als Schnittstelle ohne Integration entstanden.

Schnittstelle mit Integration

Die Schnittstellendefinition von `xmat_console` ist in Code 4.9 dargestellt. Während die Definition der `UMAT`-Schnittstelle für die Anbindung an Abaqus vorgegeben ist, sind die Übergabeargumente der `xmat_console`-Schnittstelle frei gewählt worden. In der aktuellen Form ist die Schnittstelle für einfache Tests mit geradlinigen Belastungen gedacht, sodass weder Rotationsmatrix noch Deformationsgradient übergeben werden. Für die Schnittstelle werden die folgenden Größen erwartet:

- der Materialname `materialname`,
- die Anzahl an Materialparametern `nparams`,
- die Materialparameter `materialparameters`,
- die Anzahl an Statusvariablen `nstatevar`,
- die Statusvariablen `statevariables`,
- die Anzahl an Spannungs- und Dehnungskomponenten `ncomponents`,
- Cauchy-Spannungen `oldstress` in der Reihenfolge σ_{11}, σ_{22}, σ_{33}, σ_{12}, σ_{23} und σ_{13},
- mechanische Dehnungsinkremente `oldstrain` in der Reihenfolge ε_{11}, ε_{22}, ε_{33}, ε_{12}, ε_{23}, ε_{13},
- das aktuelle Zeitinkrement `timeincrement` und
- die Zeit seit Beginn der Simulation `totaltime`.

Zurückgegeben werden

- die Cauchy-Spannungen `newstress`,
- die Statusvariablen `newstate` und
- die Jacobi-Matrix `jacobian`.

Schnittstelle ohne Integration

Die Schnittstelle ohne Integration `Dot_Values()` orientiert sich an der Definition von `xmat_console()`. Es werden die gleichen Übergabeargumente erwartet und der Aufbau ist analog zu Code 4.9, wobei `Dot_Results()` anstatt `Calculate_Results()` aufgerufen wird. Die Rückgabewerte der `Dot_Values`-Schnittstelle sind:

- die Änderung der Spannungen `dotstress`,
- die Änderung der Statusvariablen `dotstate` und
- die Jacobi-Matrix `jacobian`.

Code 4.9: Auszug aus der Fortran-Schnittstelle mit Integration über die Funktion `xmat_console()`. Nach einer Vorverarbeitung werden die internen Abläufe mit den Aufrufen `Xmat_Initialize()`, `Import_State()` und `Calculate_Results()` gesteuert. Anschließend folgt eine Zuweisung der Rückgabewerte an das aufrufende Programm

```
subroutine xmat_console( &
   ! ============= variables passed in for information ============= !
      materialname, nparams, materialparameters, nstatevar, statevariables, &
      ncomponents, oldstress, oldstrain, timeincrement, totaltime, &
   ! ==================== User coding to define ==================== !
      newstress, newstate, jacobian)
   ! ...
   character(len=80), intent(in) :: materialname
   integer, intent(in) :: nparams, nstatevar
   real(dp), dimension(nparams), intent(in) :: materialparameters
   real(dp), dimension(nstatevar), intent(in) :: statevariables
   integer, intent(in) :: ncomponents
   real(dp), dimension(ncomponents), intent(in) :: oldstress, oldstrain
   real(dp), intent(in) :: timeincrement, totaltime
   real(dp), dimension(ncomponents), intent(out) :: newstress
   real(dp), dimension(nstatevar), intent(out) :: newstate
   real(dp), dimension(ncomponents, ncomponents), intent(out) :: jacobian
   ! ---------------------------------------------------------------- !
   ! ...

   inp_stress = Import_Matrix(mat=real(oldstress, dp), num_dimensions=num_dimensions, num_shear=num_shear)
   tmp_dot_strain = Nonzero_Division(val=real(oldstrain, dp), fac=timeincrement)
   inp_dot_strain = Import_Matrix(mat=tmp_dot_strain, num_dimensions=num_dimensions, num_shear=num_shear)

   ! ...

   call Xmat_Initialize(xmat_obj=xmat_obj, solver_name=setting_solver_default, &
      constitutive_model_name=identifier, material_parameters=materialparameters, &
      calculate_jacobian=.True., firstcall=firstcall)
   call xmat_obj%Import_State(nstates=nstatevar, statevariables=statevariables, stress=inp_stress, &
      dot_strain=inp_dot_strain, totaltime=totaltime, timeincrement=timeincrement)
   call xmat_obj%Calculate_Results(exportstress=inp_stress, exportstates=newstate, &
      exportjacobian=jacobian_internal, exportdt=exportdt)

   ! --- Reassign to calling program variables
   newstress = Export_Matrix(mat=inp_stress, num_dimensions=num_dimensions, num_shear=num_shear)
   jacobian = Export_Tensor(tens=jacobian_internal, num_dimensions=num_dimensions, num_shear=num_shear)
end subroutine xmat_console
```

4.3.4 Schnittstelle zu MATLAB

MATLAB bietet mehrere Möglichkeiten zum Einbinden von externen Bibliotheken an. Ein relativ einfacher und aus MATLAB-Skripten heraus komfortabler Weg ist die Nutzung von `MEX`-Funktionen, die wie MATLAB-Funktionen aus MATLAB heraus aufgerufen werden können (siehe The MathWorks, Inc., 2021). Sofern ein Fortran-Compiler in MATLAB verlinkt ist, kann aus Fortran-Quellcode mit einer `mexFunction`-Schnittstelle (siehe Code 4.10) eine `MEX`-Funktion erzeugt werden. Zum Kompilieren der `MEX`-Datei kann ein Befehl wie der folgende genutzt werden:[16]

```
mex 'COMPFLAGS="$COMPFLAGS /free /fpp"' -largeArrayDims -DMATLAB_CALLING xmat.f
```

Die Schnittstelle zu MATLAB kann prinzipiell frei definiert werden. Wie für die Fortran-Schnittstelle `xmat_console` ist auch diese Schnittstelle hauptsächlich zum Testen genutzt worden. Für eine vergleichbare Vorgehensweise hat es sich somit angeboten, die Übergabeargumente der `mexFunction`-Schnittstelle an `xmat_console` zu orientieren. Aktuell hat der Aufruf aus MATLAB die folgende Signatur:

```
[newstress, newstate, jacobian] = xmat(materialname, materialparameters, statevariables, ...
                                       oldstress, oldstrain, timeincrement, totaltime);
```

Code 4.10: MATLAB-Schnittstelle über die Funktion `mexFunction()` nach The MathWorks, Inc. (2021). Anders als bei den vorab beschriebenen Schnittstellen werden die Ein- und Ausgabewerte mit bereitgestellten `mx...`-Funktionen aus dem Speicher der übergebenen Zeiger geladen und geschrieben. Dazu muss vorab `fintrf.h` eingebunden worden sein. Für jede übergebene vektorielle Größe in `plhs` wird zuerst die Anzahl der Elemente bestimmt und anschließend eine Zuweisung auf eine Fortran-Variable durchgeführt (Zeile 10–11). Nach der Berechnung müssen die Rückgabewerte in `prhs` gespeichert werden (Zeile 16–17).

```
1  #include fintrf.h
2  ! ...
3  subroutine mexFunction(nlhs, plhs, nrhs, prhs)
4     ! ...
5     integer(kind = 4) :: nlhs, nrhs
6     mwpointer :: plhs(nlhs), prhs(nrhs)
7     mwpointer :: mxGetNumberOfElements, mxGetData, mxCreateDoubleMatrix, mxGetPr
8     ! ...
9
10    nel_stress = mxGetNumberOfElements(prhs(4))
11    call mxCopyPtrToReal8(mxGetData(prhs(4)), stress, nel_stress)
12    ! ...
13
14    ! user coding
15
16    plhs(1) = mxCreateDoubleMatrix(nel_stress, 1, 0)
17    call mxCopyReal8ToPtr(stress, mxGetPr(plhs(1)), nel_stress)
18    ! ...
19 end subroutine mexFunction
```

[16] Dieser Befehl ist für MATLAB 2019b mit Intel Fortran-Compiler unter Windows zum Einsatz gekommen. Die Anpassungen für andere Compiler oder Betriebssysteme sollten einfach möglich sein.

Übertragen auf die maxFunction-Schnittstelle werden die folgenden Größen am Anfang des aktuellen Zeitschritts erwartet:

- der Materialname materialname,
- die Materialparameter materialparameters,
- die Statusvariablen statevariables,
- Cauchy-Spannungen oldstress in der Reihenfolge σ_{11}, σ_{22}, σ_{33}, σ_{12}, σ_{23} und σ_{13},
- mechanische Dehnungsinkremente oldstrain in der Reihenfolge ε_{11}, ε_{22}, ε_{33}, ε_{12}, ε_{23}, ε_{13},
- das aktuelle Zeitinkrement timeincrement und
- die Zeit seit Beginn der Simulation totaltime.

Zurückgegeben werden

- die Cauchy-Spannungen newstress,
- die Statusvariablen newstate und
- die Jacobi-Matrix jacobian.

Funktionen zur Ausgabe von Nachrichten und zum Beenden der Berechnung, die von MATLAB bereitgestellt worden sind, werden wie für die anderen Programme in den Hilfsfunktionen Write_Message(), Write_Warning() und Write_Error_And_Exit() definiert. Die Steuerung erfolgt mit dem Präprozessor-Makro MATLAB_CALLING.

Für die Nutzung aus GNU Octave wird aus der xMat und einem zusätzlichen Skript eine OCT-Datei kompiliert (siehe Anhang F). Intern wird anstatt der mexFunction-Schnittstelle die C-Schnittstelle aus Abschnitt 4.3.6 genutzt.

4.3.5 Schnittstelle zu Plaxis

Plaxis benötigt eine Schnittstelle namens USER_MOD und weiteren Hilfsfunktionen, die in einer Bibliothek bereitgestellt werden. Dazu kann die xMat in einer geeignet vorbereiteten Eingabeaufforderung mit dem Befehl

```
1 ifort /free /fpp /DPLAXIS_DLL /dll xmat.f /recursive /MT /exe:xmat64.dll
```

kompiliert werden.[17] Anschließend muss die Datei xmat64.dll in den udsm-Ordner im Installationsverzeichnis von Plaxis kopiert werden.

Die Struktur der USER_MOD-Schnittstelle mit den dazugehörigen Übergabeargumenten ist in Code 4.11 dargestellt. Bei einem Aufruf werden u. a. die folgenden Übergabewerte erwartet:

- Cauchy-Spannungen als die ersten sechs Einträge von Sig0 in der Reihenfolge σ_{11}, σ_{22}, σ_{33}, σ_{12}, σ_{23} und σ_{13},

[17] Der dargestellte Befehl ist für den Intel-Compiler auf einem 64 bit-System.

- mechanische Dehnungsinkremente `dEps` in der Reihenfolge ε_{11}, ε_{22}, ε_{33}, γ_{12}, γ_{23}, γ_{13} (mit $\gamma_{xy} = \varepsilon_{xy} + \varepsilon_{yx}$),
- die Zeit seit Beginn des aktuellen Simulationsschritts `Time0` und das aktuelle Zeitinkrement `dTime`,
- die Statusvariablen `StVar0`,
- die Materialparameter `Props`,
- der Zahlenwert `IDTask`, der festlegt, welche Aufgabe zu erledigen bzw. welche Rückgabewerte bereitzustellen sind und
- der Zahlenwert `iMod`, der bei mehreren Stoffmodellen bestimmt, welches Modell ausgewählt werden soll.

Code 4.11: Schnittstelle zu Plaxis nach Bentley Systems (2016). Für die Erkennung der Plaxis-Funktionen in einer Bibliothek werden Anweisungen wie in Zeile 25 benötigt. Der Aufruf von `USER_MOD()` erfordert je nach Wert von `IDTask` andere Aufgaben und Rückgabewerte.

```
1  subroutine USER_MOD( &
2     ! ============= variables passed in for information ============= !
3        IDTask, iMod, IsUndr, iStep, iTer, Iel, Jnt, X, Y, Z, &
4        Time0, dTime, Props, Sig0, Swp0, StVar0, dEps, &
5     ! ==================== user coding to define ==================== !
6        D, Bulk_W, Sig, Swp, StVar, &
7        ipl, nStat, NonSym, iStrsDep, iTimeDep, iTang, &
8     ! ========== variables passed in for debug information ========== !
9        iPrjDir, iPrjLen, &
10    ! ==================== user coding to define ==================== !
11       iAbort)
12
13    integer :: IDTask, iMod, IsUndr, iStep, iTer, Iel, Jnt
14    double precision :: X, Y, Z, Time0, dTime
15    double precision :: Props(50)
16    double precision :: Sig0(20)
17    double precision :: Swp0, Bulk_W, Swp
18    double precision :: StVar0(nStat), StVar(nStat)
19    double precision :: dEps(12)
20    double precision :: D(6, 6)
21    double precision :: Sig(6)
22    integer :: ipl, nStat, NonSym, iStrsDep, iTimeDep, iTang, iPrjDir, iPrjLen, iAbort
23
24 #ifndef NOBIB
25 !DEC$ ATTRIBUTES DLLExport,StdCall,Reference :: User_Mod
26 #endif
27
28    ! user coding of tasks depending on values of IDTask
29    !  - 1: Initialise state variables
30    !  - 2: Calculate constitutive stresses
31    !  - 3: Create effective material stiffness matrix
32    !  - 4: Return number of state variables
33    !  - 5: Return matrix attributes
34    !  - 6: Create elastic material stiffness matrix
35 end subroutine
```

Abhängig von IDTask werden u. a. die folgenden Rückgabewerte erwartet:

- Setzen der initialen Statusvariablen in StVar0 (IDTask = 1) oder Rückgabe der berechneten Statusvariablen am Ende des Zeitschritts in StVar (IDTask = 2),
- die Cauchy-Spannungen am Ende des Zeitschritts in Sig (IDTask = 2) und
- die Jacobi-Matrix D (IDTask = 3 oder 6).

Neben der Funktion USER_MOD() werden für die Nutzung aus Plaxis die folgenden Hilfsfunktionen bereitgestellt:

- subroutine GetModelCount(numModels$_{\text{out}}$) ermittelt die Anzahl der verfügbaren Stoffmodelle,
- subroutine GetModelName(iModel$^{\text{in}}$, ModelName$_{\text{out}}$) gibt den Namen des ausgewählten Stoffmodells zurück,
- subroutine GetParamCount(iModel$^{\text{in}}$, nParameters$_{\text{out}}$) ermittelt die Anzahl benötigter Materialparameter in Props für das ausgewählte Stoffmodell,
- subroutine GetParamName(iModel$^{\text{in}}$, iParameter$^{\text{in}}$, ParameterName$_{\text{out}}$) gibt den Namen des Materialparameters zurück, der sich beim ausgewählten Stoffmodell an der angegebenen Position in Props befindet,
- subroutine GetParamUnit(iModel$^{\text{in}}$, iParameter$^{\text{in}}$, ParameterUnit$_{\text{out}}$) gibt die Einheit des Materialparameters zurück, der sich beim ausgewählten Stoffmodell an der angegebenen Position im Props befindet,
- subroutine GetStateVarCount(iModel$^{\text{in}}$, nStatevars$_{\text{out}}$) gibt die Anzahl benötigter Statusvariablen für das ausgewählte Stoffmodell zurück,
- subroutine GetStateVarName(iModel$^{\text{in}}$, iStatevar$^{\text{in}}$, StatevarName$_{\text{out}}$) ermittelt den Namen der Statusvariable, die sich an der angegebenen Position im ausgewählten Stoffmodell befindet und
- subroutine GetStateVarUnit(iModel$^{\text{in}}$, iStatevar$^{\text{in}}$, StatevarUnit$_{\text{out}}$) ermittelt die Einheit der Statusvariable, die sich an der angegebenen Position im ausgewählten Stoffmodell befindet.

In Plaxis können zwar die Materialparameter in Props in der Programmoberfläche vorgegeben werden, aber nicht die Statusvariablen in StVar bzw. StVar0. Ein Aufruf der USER_MOD-Schnittstelle mit IDTask = 1 überlässt es der Benutzerroutine, die Statusvariablen zu initialisieren.

Angelehnt an die Vorgehensweise in der Benutzerroutine PLX09 werden deshalb die Anfangswerte der Statusvariablen ebenfalls in Props hinter den eigentlichen Materialparametern übergeben. Dadurch können die Statusvariablen in der USER_MOD-Schnittstelle mit IDTask = 1 dem Vektor StVar0 zugewiesen werden. Aus dem Grund betrachten die drei Hilfsfunktionen GetParamCount(), GetParamName() und GetParamUnit() nicht nur die Materialparameter, sondern auch alle Statusvariablen eines Stoffmodells.

Alle aufgelisteten Hilfsroutinen interagieren mit dem Modul PlaxisInformationPool, in dem alle Informationen bezüglich der implementierten Stoffmodelle zusammengestellt sind.

Dazu werden dort die folgenden Variablen genutzt:

- `ModelList` listet die Bezeichner der implementierten Stoffmodelle auf. Aus dem Index des Bezeichners wird mit `iMod` das Stoffmodell ausgewählt, das verwendet werden soll. Die hier festgelegte Reihenfolge bzw. Position muss auch bei den vier folgenden Variablen eingehalten werden,
- `numParameters` enthält die Anzahl der Materialparameter für jedes Stoffmodell,
- `numStateVariables` enthält die Anzahl der Statusvariablen für jedes Stoffmodell,
- `ModelParameterNamesUnits` beinhaltet sowohl Name als auch Einheit der Materialparameter für jedes Stoffmodell und
- `ModelStatevarNamesUnits` beinhaltet sowohl Name als auch Einheit der Statusvariablen für jedes Stoffmodell.

In der aktuellen Implementierung der xMat müssen die Informationen für jedes Stoffmodell dort manuell eingetragen und auf dem aktuellen Stand gehalten werden.

4.3.6 C-Schnittstelle

Unter gewissen Bedingungen kann ein Programm aus Dateien mit Fortran-Quellcode und C-Quellcode erzeugt werden (*„Mixed language programming“*, *„Interoperability“*). Die Austauschbarkeit erfordert eine Schnittstelle mit der gleichen Repräsentation von Daten und Definitionen in beiden Sprachen (siehe auch Adams u. a., 2009, S. 561). In Fortran wird mit dem `iso_c_binding`-Modul die Möglichkeit geschaffen, Fortran-Variablen, wie `integer`, `real` und `double precision`, mit dazu passenden C-Repräsentationen (`int`, `float` und `double`) zu verknüpfen. Details finden sich u. a. in Adams u. a. (2009), S. 561–579, oder Metcalf u. a. (2011), S. 243–254.

Für die Nutzung der xMat-Funktionen werden jeweils Übergabefunktionen benötigt, damit der Aufruf aus C angepasst und verarbeitet werden kann. Die Funktion `xmat_console()` wird über die C-Schnittstelle `xmat_console_CInter()` aufgerufen. Analog kann die Funktion `Dot_Values()` über die C-Schnittstelle `Dot_Values_CInter()` genutzt werden. Da beide Übergabefunktionen sehr ähnlich sind, wird in Code 4.12 nur die Definition der C-Schnittstelle `Dot_Values_CInter()` dargestellt. In den Übergabefunktionen wird die Kompatibilität zu C durch folgende Anpassungen ermöglicht:

- In Zeile 6 wird der Name der Funktion mit `bind(C, name=...)` verfügbar gemacht,
- die passenden Variablentypen werden aus `iso_c_binding` geladen (Zeile 8),
- Übergabe- und Rückgabewerte werden mit den passenden Variablentypen deklariert (Zeile 29–37),
- vor dem Aufruf von `Dot_Values()` werden die Übergabewerte in Zeile 51–60 explizit den entsprechenden Fortran-Variablen zugewiesen und
- die Rückgabewerte von `Dot_Values()` werden am Ende der Funktion in die Rückgabewerte der C-Schnittstelle geschrieben (Zeile 67–73).

Code 4.12: Die C-Schnittstelle `Dot_Values_CInter()`, über die von C-Programmen die Funktion `Dot_Values()` genutzt werden kann.

```
subroutine Dot_Values_CInter( &
   ! ============= variables passed in for information ============= !
      c_materialname, c_nparams, c_materialparameters, c_nstatevar, c_statevariables, &
      c_ncomponents, c_oldstress, c_oldstrain, c_timeincrement, c_totaltime, &
   ! ==================== user coding to define ==================== !
      c_dotstress, c_dotstate, c_jacobian) bind(C, name='Dot_Values_CInter')
! ---------------------------------------------------------------- !
   use iso_c_binding, only: c_int, c_double, c_char
   use General_Settings, only: dp, C_To_Fortran_String
   implicit none

   interface
      subroutine Dot_Values(materialname, nparams, materialparameters, nstatevar, statevariables, &
         ncomponents, oldstress, oldstrain, timeincrement, totaltime, dotstress, dotstate, jacobian)
         use General_Settings, only: dp
         character(len=80), intent(in) :: materialname
         integer, intent(in) :: nparams, nstatevar
         real(dp), dimension(nparams), intent(in) :: materialparameters
         real(dp), dimension(nstatevar), intent(in) :: statevariables
         integer, intent(in) :: ncomponents
         real(dp), dimension(ncomponents), intent(in) :: oldstress, oldstrain
         real(dp), intent(in) :: timeincrement, totaltime
         real(dp), dimension(ncomponents), intent(out) :: dotstress
         real(dp), dimension(nstatevar), intent(out) :: dotstate
         real(dp), dimension(ncomponents, ncomponents), intent(out) :: jacobian
      end subroutine Dot_Values
   end interface

   character(kind=c_char, len=1), dimension(80), intent(in) :: c_materialname
   integer(c_int), intent(in) :: c_nparams, c_nstatevar, c_ncomponents
   real(c_double), dimension(c_nparams), intent(in) :: c_materialparameters
   real(c_double), dimension(c_nstatevar), intent(in) ::c_statevariables
   real(c_double), dimension(c_ncomponents), intent(in) :: c_oldstress, c_oldstrain
   real(c_double), intent(in) :: c_timeincrement, c_totaltime
   real(c_double), dimension(c_ncomponents), intent(out) :: c_dotstress
   real(c_double), dimension(c_nstatevar), intent(out) :: c_dotstate
   real(c_double), dimension(c_ncomponents*c_ncomponents), intent(out) :: c_jacobian
   ! ---------------------------------------------------------------- !
   character(len=80) :: materialname
   real(dp), dimension(c_nparams) :: materialparameters
   real(dp), dimension(c_nstatevar) :: statevariables, dotstate
   real(dp), dimension(c_ncomponents) :: oldstress, oldstrain, dotstress
   real(dp), dimension(c_ncomponents, c_ncomponents) :: jacobian
   real(dp) :: timeincrement, totaltime
   integer :: nparams, nstatevar, ncomponents, idx, jdx

#ifndef NOBIB
!DEC$ ATTRIBUTES DLLExport,StdCall :: Dot_Values_CInter
#endif

   materialname       = C_To_Fortran_String(length=80, c_string=c_materialname)
   nparams            = c_nparams
   materialparameters = real(c_materialparameters, dp)
   nstatevar          = c_nstatevar
   statevariables     = real(c_statevariables, dp)
   ncomponents        = c_ncomponents
   oldstress          = real(c_oldstress, dp)
   oldstrain          = real(c_oldstrain, dp)
   timeincrement      = real(c_timeincrement, dp)
```

```
60    totaltime          = real(c_totaltime, dp)
61
62    call Dot_Values(materialname=materialname, nparams=nparams, materialparameters=materialparameters, &
63       nstatevar=nstatevar, statevariables=statevariables, ncomponents=ncomponents, oldstress=oldstress, &
64       oldstrain=oldstrain, timeincrement=timeincrement, totaltime=totaltime, dotstress=dotstress, &
65       dotstate=dotstate, jacobian=jacobian)
66
67    c_dotstress = real(dotstress, c_double)
68    c_dotstate = real(dotstate, c_double)
69    do idx = 1, ncomponents
70       do jdx = 1, ncomponents
71          c_jacobian((jdx-1)*ncomponents+idx) = real(jacobian(jdx, idx), c_double)
72       end do
73    end do
74 end subroutine Dot_Values_CInter
```

Wenn die xMat über die C-Schnittstelle in einem einzigen Programm genutzt werden soll, ist es empfehlenswert, die Dateien schrittweise zu kompilieren. Zuerst sollte die xMat mit einem Fortran-Compiler und dem Makro `CINTER` in Objektcode kompiliert werden, dann kann C-Quellcode zusammen mit dem Objektcode der xMat mit einem C-Compiler kompiliert und passend verlinkt werden. Auf dem Linux-System mit den GNU C- und GNU Fortran-Compilern sind die beiden folgenden Befehle genutzt worden:

```
gfortran -ffree-form -cpp -DCINTER xmat.f -c
gcc testprogram.c xmat.o -o testprogram -lgfortran -lm
```

Die Signaturen der zugrunde liegenden C-Funktionen sind in Code 4.13 dargestellt. Dabei muss beachtet werden, dass

- Fortran zusammenhängende Speicherblöcke für Vektoren, Matrizen und Tensoren erwartet. Das Speichermanagement dafür muss in C oder einer nachfolgenden Schnittstelle sichergestellt werden, z. B. mit `malloc` und anschließender Zuweisung für die Eingabewerte und `calloc` zum Reservieren des Speichers für die Rückgabewerte und
- Fortran Übergabewerte als Referenzen erwartet, was über die C-Schnittstelle explizit angegeben werden muss (siehe Anhang F für den Quellcode eines C-Testprogramms).

Es ist auch möglich, die xMat als Bibliothek bereitzustellen und die C-Schnittstelle von einer beliebigen anderen Sprache zu nutzen. Im Folgenden wird die Nutzung dieser Bibliothek aus Python und Simulink beschrieben.

Code 4.13: Die Signatur von C-Funktionen zum Aufruf der entsprechenden Funktionen der xMat. Der Speicher für alle Variablen muss auf der C-Seite bereitgestellt werden.

```
extern void Dot_Values_CInter(const char *materialname, const int *nparams,
   double materialparameters[], const int *nstatevar, double statevariables[],
   const int *ncomponents, double oldstress[], double oldstrain[], double timeincrement[],
   double totaltime[], double dotstress[], double dotstate[], double jacobian[]);

extern void xmat_console_CInter(const char *materialname, const int *nparams,
   double materialparameters[], const int *nstatevar, double statevariables[],
   const int *ncomponents, double oldstress[], double oldstrain[], double timeincrement[],
   double totaltime[], double newstress[], double newstate[], double jacobian[]);
```

Python

Um die C-Schnittstellen der xMat aus Python zu nutzen, muss zuerst eine Bibliothek aus der xMat erstellt werden. Unter Windows entspricht das dem Befehl aus Abschnitt 4.3.5, unter Linux kann der folgende Befehl mit dem GNU Fortran-Compiler benutzt werden:

```
gfortran -ffree-form -cpp -shared -fPIC -DCINTER xmat.f -o xmat.so
```

Die Verbindung von Python zur C-Schnittstelle wird über das Paket `ctypes` aus der Standardbibliothek von Python ermöglicht.[18] Da Gleitkommazahlen in Python 3 standardmäßig einem `double` in C entsprechen und die C-Schnittstelle der xMat Gleitkommazahlen mit doppelter Genauigkeit erwartet, müssen die Werte von `ctypes` nur noch explizit als `c_double` übergeben werden.

Bei der Textausgabe in Python kann es zu einem Versatz zwischen den Nachrichten aus Python und der xMat kommen. Während eine Ausgabe in der Konsole wie erwartet dargestellt werden sollte, kann sich die Reihenfolge der Nachrichten bei Umleitung der Ausgabe in eine Datei ändern, da Pufferung und Ausgabe von Nachrichten in C und Python unterschiedlich gehandhabt werden. Ein Beispiel zur Nutzung der xMat aus Python findet sich in Anhang F.

Simulink

Während die C-Schnittstelle relativ einfach von Python aus genutzt werden kann, ist für die Anbindung an Simulink etwas mehr Aufwand erforderlich. Es wird eine zusätzliche Datei (`xmat_sl.c`) mit vorgegebenen Funktionen benötigt, die eine Verbindung zwischen der C-Schnittstelle der xMat und Simulink herstellt (siehe auch Anhang F).

Sowohl die xMat als auch die Datei `xmat_sl.c` können in MATLAB folgendermaßen als C MEX S-Funktion kompiliert werden.[19]

```
mex 'COMPFLAGS="$COMPFLAGS /free /DCINTER"' xmat.f -c
mex xmat.obj xmat_sl.c
```

Anschließend kann in Simulink ein benutzerdefinierter Block mit der C MEX S-Funktion erzeugt und in eine Untersuchung eingebettet werden. Bild 4.5 zeigt eine beispielhafte Einbettung und Vernetzung der Blöcke um einen Block, der die Funktionalität von `Dot_Values_CInter()` nutzt.

[18] Die Dokumentation zu `ctypes` findet sich unter »https://docs.python.org/3/library/ctypes.html. Alternativ könnte eine C-Schnittstelle mit Cython (»http://cython.org/) realisiert werden, oder eine Anbindung über die Fortran-Schnittstelle mit F2PY (»https://numpy.org/doc/stable/f2py/index.html), was aber beides nicht weiter verfolgt worden ist.

[19] Sofern MATLAB mit einem C- und einem Fortran-Compiler verknüpft ist.

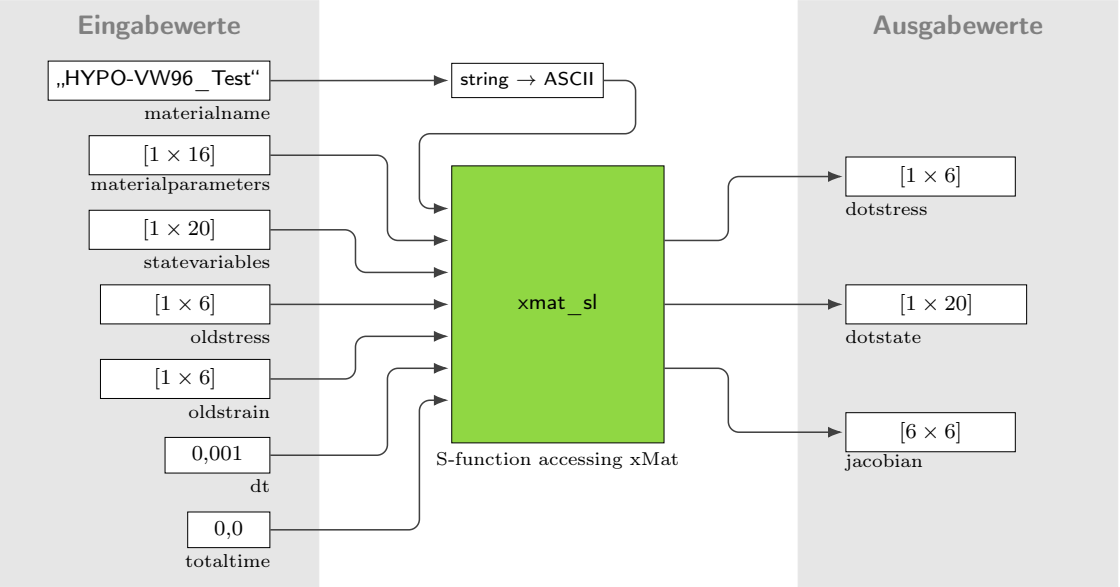


Bild 4.5: Mögliche Nutzung und Vernetzung von Blöcken in Simulink. Der Block mit der zentralen S-Funktion wird über `xmat_sl.c` bereitgestellt, in der die Kommunikation von Simulink über C-Code zur Funktion `Dot_Values_CInter()` in der xMat definiert ist.

4.4 Reflexionen

Die xMat ist mit dem Ziel programmiert worden, dass der Code modular, gut zu lesen und einfach zu erweitern ist. Nichtsdestotrotz gibt es noch Verbesserungspotentiale und implizite Annahmen im Code. Im Folgenden werden ein paar Beispiele aufgezeigt.

4.4.1 Bereiche im Code mit Verbesserungspotential

Informationen zu Stoffmodellen in PlaxisInformationPool

Aktuell nutzen die Hilfsroutinen von Plaxis die Informationen zu den Stoffmodellen aus `PlaxisInformationPool`. Aus konzeptioneller Sicht ist es besser, die dort gespeicherten Informationen über Anzahl, Name und Einheit von Materialparametern und Statusvariablen in den einzelnen Stoffmodellen zu integrieren. Dadurch sind zentrale Änderungen möglich.

Etwas näher betrachtet wird damit nur ein Teil des Problems gelöst. Eigentlich sollten die Informationen zu Materialparametern und Statusvariablen möglichst automatisch ermittelt werden können, damit die Eigenschaften nicht doppelt definiert werden müssen. Das führt zu einer weiteren Abstraktionsebene, in der die einzelnen Werte über einen Mechanismus geladen und gespeichert werden, der Zustand und Zuordnung der Werte kennt. Und dieser Mechanismus müsste wiederum außerhalb der Stoffmodellklasse einen Zugriff auf alle Stoffmodelle erlauben, um Informationen zu extrahieren.

Es sind genauere Betrachtungen notwendig, um zu entscheiden, ob sich die Vorteile dieser Herangehensweise lohnen.

Modifikation tensorieller Größen in den Statusvariablen

In der xMat können Schnittstellen bspw. die Funktion `Preprocess_State_Matrices()` aus dem Modul `Custom_Utilities` nutzen, um tensorielle Größen wie die intergranulare Dehnung in den Statusvariablen zu modifizieren. Auf den ersten Blick wäre auch eine Anpassung innerhalb der Stoffmodellklasse denkbar gewesen. Dazu müsste das Stoffmodell auf die Rotationsmatrix zugreifen können und bei jedem Aufruf von `Calculate_Dot_State()` die übergebenen Statusvariablen modifizieren. Um diesen Mehraufwand zu sparen, könnte die Anpassung in der Klasse des Integrationsverfahrens durchgeführt werden. Vor allem gegen die zweite Variante spricht, dass dadurch unnötige Abhängigkeiten in der jeweiligen Klasse entstünden, während ein Ziel die möglichst gute Kapselung der drei Bereiche Stoffmodelle, Integrationsverfahren und Schnittstellen ist.

In gewisser Weise ist die Abhängigkeit der Funktionen in der `Custom_Utilities`-Klasse von Stoffmodell und Schnittstelle der Preis, der gezahlt worden ist, um die Schnittstellen und die Stoffmodellklassen möglichst gut voneinander zu trennen und effizient zu halten. Gleichzeitig muss aufgrund der Abhängigkeit von `Custom_Utilities` zu Stoffmodellen und Schnittstellen darauf geachtet werden, dass bei Codeänderungen die Definitionen in dieser Klasse noch richtig sind.

4.4.2 Abhängigkeit von den Einheiten

Die xMat und die darin enthaltenen Stoffmodelle sind auf Basis der in Tabelle 4.4 dargestellten geotechnischen Einheiten programmiert worden. Prinzipiell ist sie auch mit anderen Einheitensystemen nutzbar, aber es müssen die folgenden Überprüfungen bzw. Anpassungen durchgeführt werden:

- Sofern ein Vergleich mit `setting_very_small_stress` stattfindet, muss der Parameter ggf. angepasst werden. Ein typischer Anwendungsfall ist der Aufruf von `Is_Valid_Stress_State()` zur Erkennung ungültiger Spannungszustände. Es wird implizit angenommen, dass der Wert mit geotechnischen Einheiten in kPa angegeben ist. Dieser Wert würde mit SI-Einheiten in Pa interpretiert werden.
- Die gleichen Überlegungen sind ebenfalls für `setting_min_youngs_modulus` gültig. Dieser Wert definiert die minimale Steifigkeit für das elastische Ersatzmodell (in kPa für geotechnische Einheiten), sodass auch hier ggf. eine Anpassung erforderlich sein kann.
- Sofern ein Stoffmodell dimensionsbehaftete Materialparameter besitzt (u. a. h_s in `Hypo-VW96`), müssen diese Werte ebenfalls vor Aufruf der xMat entsprechend angepasst werden. Beim Initialisieren des Stoffmodells wird für manche Stoffmodelle in der Funktion `Initialize()` (mit `firstcall = .True.`) eine Überprüfung der Materialparameter mit einprogrammierten Grenzen durchgeführt. Für dimensionsbehaftete Größen sind die angegebenen Grenzen in geotechnischen Einheiten angegeben und müssen für andere Einheitensysteme ggf. angepasst werden. Es ist auch möglich (aber nicht empfohlen), die Überprüfung durch `firstcall = .False.` in der Schnittstelle komplett zu deaktivieren.

Tabelle 4.4: Die Grundgrößen in zwei konsistenten Einheitensystemen. In dieser Arbeit wird das dargestellte geotechnische System verwendet.

Größe	Länge	Gewicht	Zeit	Kraft	Spannung	Energie	Dichte
Dimension	L	M	T	$\frac{\mathsf{ML}}{\mathsf{T}^2}$	$\frac{\mathsf{M}}{\mathsf{LT}^2}$	$\frac{\mathsf{ML}^2}{\mathsf{T}^2}$	$\frac{\mathsf{M}}{\mathsf{L}^3}$
SI	m	kg	s	N	Pa ($\mathrm{N/m^2}$)	J	$\mathrm{kg/m^3}$
Geotechnik	m	t	s	kN	kPa ($\mathrm{kN/m^2}$)	kJ	$\mathrm{t/m^3}$

5 Erweiterung der xMat

Leave the campground cleaner than you found it.

Übertragung der Pfadfinderregel aufs Programmieren, Martin (2009), S. 14

In diesem Kapitel werden grundlegende Änderungen an der xMat betrachtet, die teilweise über die Benutzerroutine selbst hinausgehen. Die in Kapitel 4 dargestellte Implementierung ist nicht als finale Version, sondern als Ausgangspunkt für eigene Anpassungen zu sehen.

Deshalb wird in Abschnitt 5.1 die Entwicklungsumgebung der xMat vorgestellt, in die die Benutzerroutine eingebettet ist. In der Entwicklungsumgebung kann bspw. die interne Repräsentation tensorieller Größen in der xMat geändert werden (siehe Abschnitt 5.2).

Um das Potenzial der xMat für eigene Untersuchungen zu nutzen, wird anschließend beschrieben, wie neue bzw. eigene Stoffmodelle (Abschnitt 5.3), Integrationsverfahren (Abschnitt 5.4) oder Schnittstellen zu anderen Programmen (Abschnitt 5.5) hinzugefügt werden können.[1]

Abschnitt 5.6 beschreibt, wie die Resultate einzelner Funktionen oder größere Codebereiche mithilfe von Unit-Tests überprüft werden können.

Wichtige Erkenntnisse aus diesem Kapitel sind:

Anpassbarkeit

- Grundlegende Änderungen an der xMat können in der Entwicklungsumgebung durchgeführt werden und
- durch die Struktur der xMat ist das Hinzufügen weiterer Stoffmodelle, Integrationsverfahren oder Schnittstellen relativ einfach möglich.

Verlässlichkeit

- Das korrekte Funktionieren von neuem Code kann durch Unit-Tests überprüft werden.

[1] Es ist auch die Anbindung externer Bibliotheken wie bspw. LAPACK möglich, was in Anhang D aufgezeigt wird.

5.1 Entwicklungsumgebung der xMat

Für die Arbeit an der xMat ist eine Entwicklungsumgebung erstellt worden, die als Ergebnis die in Kapitel 4 vorgestellte Benutzerroutine xMat produziert. In erster Linie ist die Entwicklungsumgebung ein Projektordner mit verschiedenen Werkzeugen, Hilfsmitteln und Quelldateien.

In der Entwicklungsumgebung werden alle Funktionen der xMat als einzelne Dateien in passenden Unterordnern unter `src/` gespeichert. Die jeweiligen Module und eine Datei mit der Hauptstruktur nutzen Anweisungen, wie `#adddirectory` und `#addfile`, um entsprechende Module und Dateien an dieser Stelle einzubinden. Die Vorgehensweise ist vergleichbar mit einem `#include`-Makro für den C-Präprozessor. Ein Hilfsprogramm lädt ausgehend von der Hauptstruktur alle so definierten Dateien, ersetzt rekursiv die Anweisungen durch den Dateiinhalt und erstellt daraus die fertige Benutzerroutine. Diese Herangehensweise hat die folgenden Vorteile:

- Funktionseinheiten, wie Stoffmodelle, Integrationsverfahren oder Schnittstellen, können einfach zu einer Benutzerroutine hinzugefügt oder entfernt werden. So können in einer Entwicklungsumgebung ohne großen Aufwand verschiedene Versionen der Benutzerroutine mit unterschiedlichem Funktionsumfang erstellt werden.
- Dadurch können Funktionen oder Module in mehreren Varianten vorliegen, was eine einfache Anpassung der internen Repräsentation tensorieller Größen ermöglicht.

5.2 Repräsentation von Matrizen und S4-Tensoren

Im Allgemeinen werden in Stoffmodellen die Zusammenhänge zwischen Spannungen und Dehnungen über tensorielle Abhängigkeiten beschrieben (siehe Abschnitt 2.3.2). Eine naheliegende Vorgehensweise ist, direkt mit S4-Tensoren und den erforderlichen Rechenoperationen zu arbeiten. Wie bereits in Tabelle 3.1 angedeutet ist, gibt es sowohl Benutzerroutinen mit Stoffmodellen die so eine vollständige Repräsentation der Tensoren nutzen, als auch Implementierungen mit einer projizierten bzw. reduzieren Repräsentation. In der Entwicklungsumgebung der xMat sind neben der tensoriellen Repräsentation `TENS3333` die zwei reduzierten Repräsentationen `MAT99` und `MAT66` auswählbar.

Die vollständige Repräsentation eines S4-Tensors definiert die 81 Einträge im Raum $\mathbb{R}^{3\times3\times3\times3}$. In der reduzierten Repräsentationen `MAT99` wird der S4-Tensor auf eine 9×9-Matrix projiziert. Analog werden Matrizen aus $\mathbb{R}^{3\times3}$ auf Vektoren mit neun Einträgen abgebildet. Dadurch vereinfachen sich mehrere Rechenoperationen – so entspricht eine doppelte Kontraktion von zwei S4-Tensoren der Multiplikation der projizierten 9×9-Matrizen und die doppelte Kontraktion von einem S4-Tensor mit einer Matrix einer Matrix-Vektor-Multiplikation.

Für symmetrische Matrizen ($M_{ij} = M_{ji}$) und S4-Tensoren mit der Symmetrie $T_{ijkl} = T_{jikl} = T_{ijlk} = T_{jilk}$ können S4-Tensoren in der reduzierten Repräsentation `MAT66` durch 6×6-Matrizen und symmetrische Matrizen durch Vektoren mit sechs Einträgen dargestellt

werden.[2] Dadurch wird weniger Speicher benötigt und Rechenoperationen können schneller durchgeführt werden. Neben der Symmetrie der Eingabematrizen muss sichergestellt werden, dass das Stoffmodell die geforderten Symmetrien erhält, da sonst systematische Fehler bei der Berechnung auftreten.

In der tensoriellen Darstellung und den beiden reduzierten Darstellungen entspricht eine Matrix $\mathbf{M}$ den folgenden Größen

$$\mathbf{M} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \quad , \quad \mathbf{M}_{\mathrm{MAT99}} = \begin{bmatrix} M_{11} \\ M_{22} \\ M_{33} \\ M_{12} \\ M_{21} \\ M_{23} \\ M_{32} \\ M_{13} \\ M_{31} \end{bmatrix} \quad \text{und} \quad \mathbf{M}_{\mathrm{MAT66}} = \begin{bmatrix} M_{11} \\ M_{22} \\ M_{33} \\ \frac{M_{12}+M_{21}}{2} \\ \frac{M_{23}+M_{32}}{2} \\ \frac{M_{13}+M_{31}}{2} \end{bmatrix} \quad .$$

Die beiden reduzierten Darstellungen eines S4-Tensors sind auf der nächsten Seite zu sehen. Eine mögliche Visualisierung der tensoriellen Darstellung findet sich in Anhang A.

Außerdem ist zu beachten, dass die Darstellung von S4-Tensoren und Rechenoperationen in der symmetrischen Projektion nicht mehr eindeutig ist (siehe Helnwein, 2001, S. 2760 f.) und zusätzliche Vereinbarungen getroffen werden müssen. Oftmals wird für Spannungen eine kontravariante und für Dehnungen eine kovariante Darstellung gewählt (siehe bspw. Simo und Hughes, 1998, S. 168–170). In der xMat wird durchweg eine tensorielle Darstellung verwendet, auch für die `MAT66`-Repräsentation. Das erfordert u. a. die Anpassung einiger Rechenoperationen selbst, damit die korrekten Ergebnisse berechnet werden. Insbesondere erzeugen die folgenden Rechenoperationen mit symmetrischen Eingabewerten Skalare, symmetrische Matrizen oder S4-Tensoren[3] mit der gleichen Symmetrie $\mathbb{S}$:

$$\begin{aligned} &\mathcal{T} : \mathcal{K} \in \mathbb{S}^{3\times3\times3\times3} \quad , \qquad \mathcal{T} + \mathcal{K} \in \mathbb{S}^{3\times3\times3\times3} \quad , \qquad \mathbf{A} + \mathbf{B} \in \mathbb{S}^{3\times3} \quad , \\ &\mathbf{A} : \mathbf{B} \in \mathbb{S} \quad , \qquad \mathcal{T} : \mathbf{A} \in \mathbb{S}^{3\times3} \quad \text{und} \qquad \mathbf{A} \otimes \mathbf{B} \in \mathbb{S}^{3\times3\times3\times3} \quad . \end{aligned}$$

Im Allgemeinen ist das Produkt von zwei symmetrischen Matrizen nicht symmetrisch, aber das Produkt von zwei gleichen symmetrischen Matrizen erhält die Symmetrie. Ähnlichkeitstransformationen $\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q}$ mit einer symmetrischen Matrix $\mathbf{A}$ und einer beliebigen,

[2] Ein Raum der Dimension n^2 hat einen symmetrischen Unterraum der Dimension $n(n+1)/2$ (siehe u. a. Truesdell und Noll, 2004, S. 20). Damit ergibt sich für $n = 3$ eine Dimension von 9 für `MAT99` und 6 für `MAT66`. S4-Tensoren mit der geforderten *minor symmetry* haben höchstens 36 unterschiedliche Einträge und total symmetrische Tensoren höchstens 21 (siehe Backus, 1970, S. 641) und somit können beide als 6×6-Matrizen dargestellt werden.

[3] Für S4-Tensoren muss mindestens $T_{ijkl} = T_{jikl}$ erfüllt sein.

$$\mathcal{T}_{\text{MAT99}} = \left(\begin{array}{ccc:ccc:ccc} T_{1111} & T_{1122} & T_{1133} & T_{1112} & T_{1121} & T_{1123} & T_{1132} & T_{1113} & T_{1131} \\ T_{2211} & T_{2222} & T_{2233} & T_{2212} & T_{2221} & T_{2223} & T_{2232} & T_{2213} & T_{2231} \\ T_{3311} & T_{3322} & T_{3333} & T_{3312} & T_{3321} & T_{3323} & T_{3332} & T_{3313} & T_{3331} \\ \hdashline T_{1211} & T_{1222} & T_{1233} & T_{1212} & T_{1221} & T_{1223} & T_{1232} & T_{1213} & T_{1231} \\ T_{2111} & T_{2122} & T_{2133} & T_{2112} & T_{2121} & T_{2123} & T_{2132} & T_{2113} & T_{2131} \\ \hdashline T_{2311} & T_{2322} & T_{2333} & T_{2312} & T_{2321} & T_{2323} & T_{2332} & T_{2313} & T_{2331} \\ T_{3211} & T_{3222} & T_{3233} & T_{3212} & T_{3221} & T_{3223} & T_{3232} & T_{3213} & T_{3231} \\ \hdashline T_{1311} & T_{1322} & T_{1333} & T_{1312} & T_{1321} & T_{1323} & T_{1332} & T_{1313} & T_{1331} \\ T_{3111} & T_{3122} & T_{3133} & T_{3112} & T_{3121} & T_{3123} & T_{3132} & T_{3113} & T_{3131} \end{array}\right) \quad \text{und}$$

$$\mathcal{T}_{\text{MAT66}} = \left(\begin{array}{ccc:ccc} T_{1111} & T_{1122} & T_{1133} & \frac{T_{1112}+T_{1121}}{2} & \frac{T_{1123}+T_{1132}}{2} & \frac{T_{1113}+T_{1131}}{2} \\ T_{2211} & T_{2222} & T_{2233} & \frac{T_{2212}+T_{2221}}{2} & \frac{T_{2223}+T_{2232}}{2} & \frac{T_{2213}+T_{2231}}{2} \\ T_{3311} & T_{3322} & T_{3333} & \frac{T_{3312}+T_{3321}}{2} & \frac{T_{3323}+T_{3332}}{2} & \frac{T_{3313}+T_{3331}}{2} \\ \hdashline \frac{T_{1211}+T_{2111}}{2} & \frac{T_{1222}+T_{2122}}{2} & \frac{T_{1233}+T_{2133}}{2} & \frac{T_{1212}+T_{1221}+T_{2112}+T_{2121}}{4} & \frac{T_{1223}+T_{1232}+T_{2123}+T_{2132}}{4} & \frac{T_{1213}+T_{1231}+T_{2113}+T_{2131}}{4} \\ \frac{T_{2311}+T_{3211}}{2} & \frac{T_{2322}+T_{3222}}{2} & \frac{T_{2333}+T_{3233}}{2} & \frac{T_{2312}+T_{2321}+T_{3212}+T_{3221}}{4} & \frac{T_{2323}+T_{2332}+T_{3223}+T_{3232}}{4} & \frac{T_{2313}+T_{2331}+T_{3213}+T_{3231}}{4} \\ \frac{T_{1311}+T_{3111}}{2} & \frac{T_{1322}+T_{3122}}{2} & \frac{T_{1333}+T_{3133}}{2} & \frac{T_{1312}+T_{1321}+T_{3112}+T_{3121}}{4} & \frac{T_{1323}+T_{1332}+T_{3123}+T_{3132}}{4} & \frac{T_{1313}+T_{1331}+T_{3113}+T_{3131}}{4} \end{array}\right) \quad \text{bzw.}$$

$$= \left(\begin{array}{ccc:ccc} T_{1111} & T_{1122} & T_{1133} & T_{1112} & T_{1123} & T_{1113} \\ T_{2211} & T_{2222} & T_{2233} & T_{2212} & T_{2223} & T_{2213} \\ T_{3311} & T_{3322} & T_{3333} & T_{3312} & T_{3323} & T_{3313} \\ \hdashline T_{1211} & T_{1222} & T_{1233} & T_{1212} & T_{1223} & T_{1213} \\ \hdashline T_{2311} & T_{2322} & T_{2333} & T_{2312} & T_{2323} & T_{2313} \\ \hdashline T_{1311} & T_{1322} & T_{1333} & T_{1312} & T_{1323} & T_{1313} \end{array}\right) \quad \text{aufgrund der vorausgesetzten Symmetrien.}$$

regulären Matrix **Q** erhalten die Symmetrie der Matrix. Bei der Berechnung von tr(**TD**) wird die Spur unabhängig von der Symmetrie der resultierenden Matrix auch in der reduzierten Repräsentation richtig bestimmt.

Für jedes Stoffmodell kann durch eine Analyse der Gleichungen bestimmt werden, welche Repräsentation aufgrund von Symmetrien möglich und sinnvoll ist. Die im Rahmen dieser Arbeit implementierten Stoffmodelle sind so formuliert, dass sie mit allen drei Darstellungen genutzt werden können. Dazu wird die Dimension der tensoriellen Größen intern nicht mit Zahlenwerten, sondern mit Hilfsgrößen definiert (siehe Tabelle 5.1).

Welche Repräsentation auch gewählt wird, solange die Implementierung richtig und konsistent durchgeführt wird, ist eine eindeutige Transformation zwischen allen Repräsentationen möglich (siehe Bild 5.1). Aufgrund der schnelleren Berechnungszeit ist in der Entwicklungsumgebung der xMat standardmäßig die Repräsentation `MAT66` vorausgewählt. Falls zusätzliche Stoffmodelle eine spezielle Formulierung erfordern, kann die Darstellung in der durch `MAT99` oder die tensorielle Repräsentation `TENS3333` ersetzt werden.

Tabelle 5.1: Interne Hilfsgrößen, die vor dem Kompilieren durch die angegebenen Zahlenwerte ersetzt werden. Durch die Verwendung der allgemeinen Hilfsgrößen im Code kann automatisch für jede gewählte Repräsentation die passende Benutzerroutine erstellt werden.

Hilfsgröße	`TENS3333`	`MAT99`	`MAT66`
`__tensor__`	3, 3, 3, 3	9, 9	6, 6
`__matrix__`	3, 3	9	6
`__nelmat__`	9	9	6

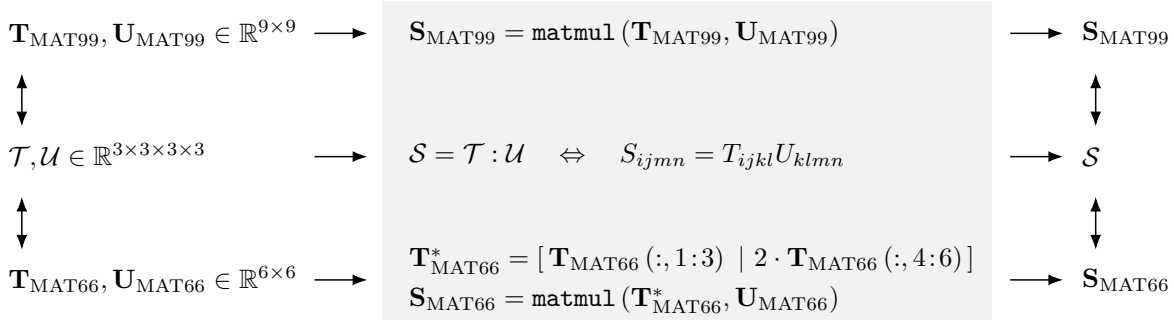


Bild 5.1: Projektion von Tensoren und deren Rechenoperationen auf Matrizen am Beispiel von `Double_Contraction44()`. Links sind die Eingangsgrößen der drei verschiedenen Repräsentationen dargestellt und rechts die Ausgangsgrößen. Für jede Repräsentation wird ggf. die Rechenoperation angepasst, sodass jeweils Eingangsgrößen und Ausgangsgrößen eindeutig von einer Repräsentation in eine andere transformiert werden können und alle drei Repräsentationen in dem Sinne gleichwertig sind.

5.3 Hinzufügen weiterer Stoffmodelle

Grundsätzlich sind vier Schritte notwendig, um ein neues Stoffmodell zur xMat hinzuzufügen:

1. Eine neue Klasse mit dem eigentlichen Stoffmodell zu erstellen, unterteilt in
   - die Adaption der vorgegebenen Klassenstruktur und
   - die Implementierung der Stoffgleichungen,
2. die Bezeichnung in `General_Settings` zu definieren,
3. das Stoffmodell in `Constitutive_Model` einzubinden und
4. die Informationen in `PlaxisInformationPool` zu aktualisieren.

Neue Klasse erstellen (Klassenstruktur)

Stoffmodelle werden in der xMat als eigene Klassen hinzugefügt, die die gemeinsame Basisklasse `Constitutive_Model_Baseclass` erweitern. In der Klasse müssen Variablen für Materialparameter definiert werden. Zusätzliche Variablen, z. B. für vorberechnete Werte, können ebenfalls definiert und genutzt werden. Außerdem müssen die beiden Funktionen `Initialize()` und `Calculate_Dot_State()` für das Stoffmodell definiert werden. Falls mehrere Stoffmodelle gleiche oder sehr ähnliche Funktionalitäten oder Hilfsfunktionen haben, kann eine Auslagerung von Funktionen in die Basisklasse sinnvoll sein.

Um das Vorgehen zu verdeutlichen, wird im Folgenden aufgezeigt, wie das Stoffmodell `Baro-Ko21` in die xMat integriert werden kann bzw. worden ist. Als Grundlage dient die Definition des Stoffmodells aus Abschnitt 2.3.4 und die Implementierung als MATLAB-Skript aus Kolymbas (2022), die in Code 5.1 dargestellt ist.

Code 5.1: Die Referenzimplementierung von `Baro-Ko21` als MATLAB-Skript nach Kolymbas (2022), S. 127 f.

```
function [Tp, ecp] = stg(T, D, e, ec)
   c = [-1.025 .5 -0.2 5000 .5];
   p = -trace(T)/3;
   E = 1.0 - 0.02*log(p);
   ecp = 30*(E-ec);

   nT = norm(T, 'fro');
   nD = norm(D, 'fro');
   T0 = T/nT;
   D0 = D/nD;
   delta = trace(D0);

   D0S = D0-delta/3*eye(3);
   R = -expm(c(1)*D0S)+c(2)*delta*eye(3);
   R0 = R/norm(R, 'fro');

   h = c(4)*nT^c(5);
   f = ec+c(3)*delta;
   g = -e+c(3)*delta;
   Tp = h*(f*R0+g*T0)*nD;
end
```

```
22
23 % ...
24
25 nD = norm(D, 'fro');
26 D0 = D/nD;
27 L = D0-D0alt;
28 lambda = norm(L, 'fro');
29 D0alt = D0;
30 ec = ec+ecp*dt+lambda*(ec0-ec)/2;
```

Neben den angegebenen bzw. ersichtlichen Parametern c_1, c_2, c_3, c_4, c_5 und dem initialen Übergabewert e_{c0} sind die restlichen vier Parameter des Stoffmodells κ_1, κ_2, k_1, und k_2 als Zahlenwerte einprogrammiert (siehe Stoffmodellgleichungen in Abschnitt 2.3.4). Für die neue Klasse ergeben sich also zehn Parameter, die in der Klasse definiert und in der `Initialize()`-Funktion zugewiesen werden müssen. Die notwendige Struktur dazu ist in Code 5.2 dargestellt.

Sofern die Jacobi-Matrix bzw. deren Berechnung für das Stoffmodell bekannt ist, sollte die Bereitstellung implementiert werden. Anderenfalls muss bei der Initialisierung, wie in Zeile 31 von Code 5.2 dargestellt, `provide_jacobian=.False.` an `Base_Initialization()` übergeben werden. Die Bereitstellung ist im Allgemeinen deutlich effizienter, als eine numerische Berechnung der Jacobi-Matrix.

Code 5.2: Beispielhafte Struktur einer Stoffmodellklasse am Beispiel von `Baro-Ko21`.

```
1  module Barodesy_Ko21_Class
2     use General_Settings, only: dp, setting_num_statevariables, setting_max_internal_states
3     use Constitutive_Model_Baseclass, only: Constitutive_Model
4     implicit none
5     ! ---------------------------------------------------------------- !
6     type, extends(Constitutive_Model) :: Barodesy_Ko21
7     ! ---------------------------------------------------------------- !
8        private
9        real(dp) :: param_phi_c, param_c2, param_c3, param_c4, param_c5, param_kappa1, param_kappa2, &
10                   param_k1, param_k2, param_e_c0
11       real(dp) :: derived_c1
12       contains
13       procedure :: Initialize
14       procedure :: Calculate_Dot_State
15    end type
16
17    contains
18
19    ! ---------------------------------------------------------------- !
20    subroutine Initialize(this, params, calculate_jacobian, firstcall)
21    ! ---------------------------------------------------------------- !
22       use Math_Operations, only: Abort_If_Not_In_Interval
23       !
24       class(Barodesy_Ko21), intent(inout) :: this
25       real(dp), dimension(:), intent(in) :: params
26       logical, intent(in) :: calculate_jacobian, firstcall
27       ! ---------------------------------------------------------- !
28       real(dp) :: K_c
29
30       ! --- Initialize base class variables
31       call this%Base_Initialization(calculate_jacobian=calculate_jacobian, provide_jacobian=.False.)
32       this%direct_variables_mask(3:12) = .True.
33
```

```
34        ! --- Parameters
35        this%param_phi_c  = params(1)
36        this%param_c2     = params(2)
37        this%param_c3     = params(3)
38        this%param_c4     = params(4)
39        this%param_c5     = params(5)
40        this%param_kappa1 = params(6)
41        this%param_kappa2 = params(7)
42        this%param_k1     = params(8)
43        this%param_k2     = params(9)
44        this%param_e_c0   = params(10)
45
46        ! --- Check valid parameter range
47        if (firstcall) then
48           call Abort_If_Not_In_Interval('phi_c', this%param_phi_c, [0.1745_dp, 1.0472_dp])
49        end if
50
51        K_c = (1.0_dp - sin(this%param_phi_c)) / (1.0_dp + sin(this%param_phi_c))
52        this%derived_c1 = sqrt(2.0_dp/3.0_dp)*log(K_c)
53     end subroutine Initialize
54
55     ! ------------------------------------------------------------------ !
56     function Calculate_Dot_State(this, ref_dt, cur_time, cur_state) result(dot_state)
57     ! ------------------------------------------------------------------ !
58        ! ...
59     end function Calculate_Dot_State
60  end module Barodesy_Ko21_Class
```

Neue Klasse erstellen (Stoffgleichungen)

Die Implementierung im Rahmen der `Calculate_Dot_State()`-Funktion ist in Code 5.3 dargestellt. Im Vergleich mit der Funktion `stg()` in Code 5.1 finden sich die zentralen Gleichungen des Stoffmodells in Zeile 36 bis 62 von Code 5.3 dargestellt wieder. Für die Funktion sind mehrere Punkte zu beachten:

- Bei der Deklaration der Variablen werden anstelle konkreter Angaben die Makros `__matrix__` für Matrizen und `__tensor__` wird für S4-Tensoren verwendet. Falls die Anzahl an Elementen benötigt wird, kann `__nelmat__` genutzt werden, was bei diesem Stoffmodell nicht erforderlich ist. Mit den Makros kann das Stoffmodell, wie dargestellt, unabhängig von einer konkreten Darstellung implementiert werden.
- Wie das MATLAB-Beispiel und Gleichung (2.12) zeigen, setzt sich die Integration von e_c aus zwei Teilen zusammen, wobei nur der erste Teil vom aktuellen Zeitschritt abhängt. Um Probleme mit adaptiven Integrationsverfahren zu vermeiden, wird die Variable in zwei Feldern gespeichert und der zeitunabhängige Teil gesondert behandelt (siehe Zeilen 33, 59, 62, 64 und 69).
    - Für den zweiten Teil sind Informationen zur Dehnungsrate aus dem vorherigen Zeitschritt erforderlich, die mit den Statusvariablen gespeichert werden (siehe Zeilen 34 und 65).[4]

[4] Matrizen sollten immer als Vektoren mit neun Elementen gespeichert werden. Für die Transformation sollten `Vec9_To_Mat()` und `Mat_To_Vec9()` genutzt werden.

- Da weder der zeitunabhängige Teil noch die Dehnungsraten integriert werden sollen, werden sie `this%direct_variables` zugewiesen (Zeile 64 und 65) und die entsprechenden Stellen in `dot_state` bleiben null. Damit das im Integrationsverfahren richtig berücksichtigt wird, muss eine passende Maske in `Initialize()` definiert werden (Zeile 32 in Code 5.2).

- Die Entwicklung der Porenzahl wird mit $\dot{e} = (1 + e)\,\mathrm{tr}(\mathbf{D})$ berechnet und ebenfalls in den Statusvariablen gespeichert (Zeile 31, 56 und 68). Insgesamt sind also 12 Statusvariablen für dieses Stoffmodell notwendig.

Code 5.3: Die `Calculate_Dot_State()`-Funktion von `Baro-Ko21`.

```
1   ! ------------------------------------------------------------ !
2   function Calculate_Dot_State(this, ref_dt, cur_time, cur_state, dot_strain) result(dot_state)
3   ! ------------------------------------------------------------ !
4      use Math_Operations, only: const_identity2d, Nonzero_Division, Norm, Trace, Deviatoric_Part, &
5                                 Matrix_Exponential, Pack_States, Unpack_States, Vec9_To_Mat, Mat_To_Vec9
6      !
7      class(Barodesy_Ko21), intent(inout) :: this
8      real(dp), intent(in) :: ref_dt
9      real(dp), intent(in) :: cur_time
10     real(dp), dimension(setting_max_internal_states), intent(in) :: cur_state
11     real(dp), dimension(__matrix__), intent(in) :: dot_strain
12     real(dp), dimension(setting_max_internal_states) :: dot_state
13     ! ------------------------------------------------------------ !
14     real(dp), dimension(__matrix__) :: cur_stress, dot_stress, stress_dir, R_mat, R_dir
15     real(dp), dimension(__matrix__) :: D_dir, D_dirold, D_dirdev
16     real(dp), dimension(setting_num_statevariables) :: statevariables, dot_statevariables
17     real(dp), dimension(__tensor__) :: jac_stress, dot_jac_stress
18     real(dp), dimension(setting_num_statevariables, __matrix__) :: jac_statevariables
19     real(dp), dimension(setting_num_statevariables, __matrix__) :: dot_jac_statevariables
20     real(dp) :: cur_voidratio, dot_voidratio, p_mean, norm_D, norm_T, delta, e_c, lambda_D, &
21                 dot_e_c1, delta_e_c2, eps1, eps2, h_fac, f_fac, g_fac
22
23     dot_state = 0.0_dp
24     dot_jac_stress = 0.0_dp
25     dot_statevariables = 0.0_dp
26     dot_jac_statevariables = 0.0_dp
27
28     ! --- Unpacking of all states out of a long vector
29     call Unpack_States(input_states=cur_state, stress=cur_stress, jac_stress=jac_stress, &
30        statevariables=statevariables, jac_statevariables=jac_statevariables)
31     cur_voidratio = statevariables(1)
32     ! Currently e_c is saved in two fields due to different integration of both components
33     e_c = statevariables(2) + statevariables(3)
34     D_dirold = Vec9_To_Mat(vec9=statevariables(4:12))
35
36     norm_D = Norm(dot_strain)
37     D_dir = Nonzero_Division(val=dot_strain, fac=norm_D)
38     delta = Trace(D_dir)
39
40     norm_T = Norm(cur_stress)
41     stress_dir = Nonzero_Division(val=cur_stress, fac=norm_T)
42     p_mean = -Trace(cur_stress)/3.0_dp
43
44     eps1 = this%param_k1 - this%param_k2*log(p_mean)
45     eps2 = eps1*(1.0_dp + this%param_kappa2*delta)
46
47     D_dirdev = Deviatoric_Part(D_dir)
48     R_mat = -Matrix_Exponential(mat=D_dirdev * this%derived_c1) + this%param_c2*delta*const_identity2d
```

```
      R_dir = Nonzero_Division(val=R_mat, fac=Norm(R_mat))

      h_fac = this%param_c4 * norm_T**this%param_c5
      f_fac = e_c + this%param_c3*delta
      g_fac = -cur_voidratio + this%param_c3*delta

      dot_stress = h_fac * (f_fac*R_dir + g_fac*stress_dir) * norm_D
      dot_voidratio = (1.0_dp + cur_voidratio)*Trace(dot_strain)

      ! The integration of dot_e_c relies on two different parts as hinted in Kolymbas (2021)
      dot_e_c1 = this%param_kappa1*(eps2 - e_c) * norm_D

      lambda_D = Norm(D_dir - D_dirold)
      delta_e_c2 = lambda_D * 0.5_dp * (this%param_e_c0 - e_c)

      this%direct_variables(3) = statevariables(3) + delta_e_c2
      this%direct_variables(4:12) = Mat_To_Vec9(mat=D_dir)

      ! --- Packing all dot_states in a long vector
      dot_statevariables(1) = dot_voidratio
      dot_statevariables(2) = dot_e_c1
      dot_state = Pack_States(stress=dot_stress, jac_stress=dot_jac_stress, &
         statevariables=dot_statevariables, jac_statevariables=dot_jac_statevariables)
   end function Calculate_Dot_State
```

Bezeichnung definieren

Im nächsten Schritt muss eine Bezeichnung für das neue Stoffmodell gewählt und in eine neue `setting_id_...`-Variable im Modul `General_Settings` eingetragen und verfügbar gemacht werden (bspw. `setting_id_barodesy_ko21`). Die Voreinstellung erfordert eine Zeichenlänge von `setting_len_id` und es empfiehlt sich, die Namensstruktur der anderen Stoffmodelle beizubehalten.

```
character(len=setting_len_id), parameter :: setting_id_barodesy_ko21 = 'Baro-Ko21'
! ...
public ... setting_id_barodesy_ko21
```

Stoffmodell einbinden

Für das neue Stoffmodell muss die Abfrage in der Funktion `Select_Constitutive_Model()` im Model `Constitutive_Model` angepasst werden. Dazu muss zuerst der Bezeichnung und die Klasse in `Constitutive_Model` wie folgt verfügbar gemacht werden:

```
use General_Settings, only: ... setting_id_barodesy_ko21
! ...
use Barodesy_Kolymbas21_Class
```

Anschließend wird in `Select_Constitutive_Model()` eine zusätzliche Abfrage für das Stoffmodell hinzugefügt. Dort wird der Speicher für die neue Klasse reserviert, sofern sie ausgewählt worden ist.

```
   else if (identifier == setting_id_barodesy_ko21) then
      allocate(Barodesy_Kolymbas21::new_constitutive_model)
```

Informationspool aktualisieren

Zuletzt müssen noch alle Variablen im Modul `PlaxisInformationPool` für die Plaxis-Unterstützung angepasst werden. Dort werden die Informationen zu allen implementierten Stoffmodellen bereitgestellt, mit denen Plaxis eine Berechnung durchführen kann.[5] Zuerst muss die Bezeichnung des Stoffmodells verfügbar gemacht werden.

```
use General_Settings, only: ... setting_id_barodesy_ko21
```

Anschließend müssen die Angaben zu Materialparametern und Statusvariablen in diesem Modul hinterlegt werden. Dazu sind die in Code 5.4 dargestellten Anpassungen an den fünf Variablen der Klasse nötig.

Code 5.4: Anpassungen im Modul `PlaxisInformationPool` für die Unterstützung des Stoffmodells `Baro-Ko21` aus Plaxis. Es sind nur die Ergänzungen dargestellt und auf fehlenden Code wird mit ... hingewiesen.

```
   character(len=setting_len_id), parameter, dimension(7) :: ModelList = [... setting_id_barodesy_ko21]
   integer, parameter, dimension(7) :: numParameters =     [... 10]
   integer, parameter, dimension(7) :: numStateVariables = [... 12]

   ...

   character(len=16), parameter, dimension(16, 7) :: ModelParameterNamesUnits = reshape([ &
      ...
      ! Baro-Ko21
      '@j#_c#   :rad   ', 'c_2#     :-     ', 'c_3#     :-     ', 'c_4#     :-     ', &
      'c_5#     :-     ', '@k_1#    :-     ', '@k_2#    :-     ', 'k_1#     :-     ', &
      'k_2#     :-     ', 'e_c0#    :-     ', '         :-     ', '         :-     ', &
      '         :-     ', '         :-     ', '         :-     ', '         :-     ' &
      ], [16, 7])

   character(len=16), parameter, dimension(12, 7) :: ModelStatevarNamesUnits = reshape([ &
      ...
      ! Baro-Ko21
      'voidratio:-     ', 'e_c1#    :-     ', 'e_c2#    :-     ', 'D_old#11 :-     ', &
      'D_old#22 :-     ', 'D_old#33 :-     ', 'D_old#12 :-     ', 'D_old#21 :-     ', &
      'D_old#23 :-     ', 'D_old#32 :-     ', 'D_old#13 :-     ', 'D_old#31 :-     ' &
   ], [12, 7])
```

An diesem Punkt ist das neue Stoffmodell in die xMat integriert. Insgesamt ist die Implementierung zwar etwas aufwändiger und umfangreicher als für das MATLAB-Skript, aber ein so in die xMat implementiertes Stoffmodell kann mit allen bereitgestellten Integrationsverfahren von allen unterstützten Schnittstellen aus genutzt werden.

[5] Mehr Informationen zum Modul `PlaxisInformationPool` findet sich in Abschnitt 4.3.5.

5.4 Hinzufügen weiterer Integrationsverfahren

Das Hinzufügen weiterer Integrationsverfahren hat viele Gemeinsamkeiten mit der eben beschriebenen Vorgehensweise für neue Stoffmodelle. Im Rahmen der Arbeit sind nur explizite Runge-Kutta-Verfahren eingesetzt worden, aber die Implementierung anderer Integrationsverfahren sollte vergleichbar sein.

Zuerst muss eine neue Klasse erstellt werden, die `Solver_Baseclass` erweitert. Die bereits implementierten Integrationsverfahren nutzen keine zusätzlichen Variablen in der abgeleiteten Klasse. Die Klasse für das Integrationsverfahren muss nur die Funktion `Integration_Algorithm()` definieren, wobei auch weitere Hilfsfunktionen möglich sind. Am Beispiel des Bogacki-Shampine-Verfahrens ergibt sich die in Code 5.5 dargestellte Struktur für ein Integrationsverfahren.

Code 5.5: Beispielhafte Klassenstruktur eines Integrationsverfahrens basierend auf dem implementierten `RK-BS23`.

```
module RK23_Bogacki_Shampine_Class
   use General_Settings, only: dp, setting_max_internal_states
   use Solver_Baseclass, only: Solver
   use Constitutive_Model_Baseclass, only: Constitutive_Model
   implicit none

   ! ---------------------------------------------------------------- !
   type, extends(Solver) :: RK23_Bogacki_Shampine
   ! ---------------------------------------------------------------- !
      private

      contains

      procedure :: Integration_Algorithm
   end type

   contains

   ! ---------------------------------------------------------------- !
   function Integration_Algorithm(this, integ_obj, cur_state, dot_state_k1, cur_time, dt) &
      result(new_state)
   ! ---------------------------------------------------------------- !
      class(RK23_Bogacki_Shampine), intent(inout) :: this
      class(Constitutive_Model), intent(inout) :: integ_obj
      real(dp), dimension(setting_max_internal_states), intent(in) :: cur_state, dot_state_k1
      real(dp), intent(in) :: cur_time, dt
      real(dp), dimension(setting_max_internal_states) :: new_state
      ! ------------------------------------------------------------ !
      real(dp), dimension(setting_max_internal_states) :: dot_state_k2, dot_state_k3, &
                                                          dot_state_k4, extra_state

      dot_state_k2 = integ_obj%Get_Dot_State(ref_dt=dt, &
         cur_time=cur_time + 0.5_dp*dt, cur_state=cur_state + dt*0.5_dp*dot_state_k1)
      dot_state_k3 = integ_obj%Get_Dot_State(ref_dt=dt, &
         cur_time=cur_time + 0.75_dp*dt, cur_state=cur_state + dt*0.75_dp*dot_state_k2)

      new_state = cur_state + dt*(2.0_dp/9.0_dp*dot_state_k1 &
                  + 1.0_dp/3.0_dp*dot_state_k2 + 4.0_dp/9.0_dp*dot_state_k3)

      dot_state_k4 = integ_obj%Get_Dot_State(ref_dt=dt, &
         cur_time=cur_time + dt, cur_state=new_state)
```

```
42        this%next_dot_state_assigned = .True.
43        this%next_dot_state = dot_state_k4
44
45        extra_state = cur_state + dt*(7.0_dp/24.0_dp*dot_state_k1 + 1.0_dp/4.0_dp*dot_state_k2 &
46                    + 1.0_dp/3.0_dp*dot_state_k3 + 1.0_dp/8.0_dp*dot_state_k4)
47        call this%Set_Error_Estimate(estimation=abs(new_state - extra_state))
48     end function Integration_Algorithm
49  end module RK23_Bogacki_Shampine_Class
```

Im nächsten Schritt muss das Modul `Solver` angepasst werden. Dazu wird die Klasse mit einer `use ...`-Anweisung (hier: `use RK23_Bogacki_Shampine_Class`) verfügbar gemacht. Für diese neue Variable muss in der Funktion `Select_Solver` ein neuer Zweig zur bestehenden Reihe an `if`-Abfragen hinzugefügt werden, um den Speicher für die neue Klasse zu reservieren. Für eine neue Klasse `RK23_Bogacki_Shampine` mit der Bezeichnung `RK-BS23` werden die folgenden Zeilen vor der finalen `else`-Anweisung in `Select_Solver()` ergänzt:

```
1     else if (identifier == 'RK-BS23') then
2        allocate(RK23_Bogacki_Shampine::new_solver)
3        call new_solver%Initialize(name='RK2/3 (Bogacki-Shampine)', exp_stepgrow=-1.0_dp/3.0_dp, &
4           exp_stepshrink=-0.5_dp, max_stepgrow=3.0_dp, min_stepshrink=0.2_dp)
```

Die Argumente `exp_stepgrow`, `exp_stepshrink`, `max_stepgrow` und `min_stepshrink` sind optional, aber sinnvoll für eine zum Integrationsverfahren passende adaptive Schrittweiten-Kontrolle. Weitere Informationen dazu finden sich in Abschnitt 4.2.3 sowie in Press u. a. (1997), S. 712. Außerdem ist es empfehlenswert, die Bezeichnung des neuen Integrationsverfahrens als Kommentar zum Parameter `setting_solver_default` im Modul `General_Settings` hinzuzufügen, damit ein schneller Überblick über die verfügbaren Integrationsverfahren möglich ist. Um das neue Integrationsverfahren zu nutzen, muss die Bezeichnung des Integrationsverfahrens `setting_solver_default` zugewiesen werden.

5.5 Hinzufügen weiterer Schnittstellen

Für jede Schnittstelle wird typischerweise eine eigene Funktion in Form einer `subroutine` benötigt. Der Aufbau ist dabei für vorhandene und neue Schnittstellen identisch und wird beispielhaft in Code 5.6 aufgezeigt:

1. Importieren von `Xmat` und `Xmat_Initialize` aus der Klasse `Xmat_Class` (Zeile 13),
2. konvertieren aller Übergabewerte aus dem aufrufenden Programm zu internen Variablen mit passender Größe und Genauigkeit, ggf. auch eine Vorverarbeitung in der Schnittstelle oder mit anderen Hilfsfunktionen (Zeile 37 bis 54),
3. erstellen eines Objekts der Klasse `xmat_obj` durch den Aufruf von `Xmat_Initialize()` (Zeile 56 bis 58). Mit diesem Objekt können die internen Abläufe über `Import_State()` und `Calculate_Results()` gesteuert werden (Zeile 59 bis 62). Für Schnittstellen ohne Integration kommt `Dot_Results()` statt `Calculate_Results()` zum Einsatz und
4. konvertieren der berechneten Werte in das Format, das vom aufrufenden Programm erwartet wird (Zeile 65 und 66).

In den internen Abläufen wird angenommen, dass sich die übergebene Cauchy-Spannung, die Dehnungsinkremente und andere tensorielle Größen im selben Bezugssystem befinden. Gegebenenfalls notwendige Transformationen sollten in der Schnittstelle oder einer Hilfsfunktion aus `Custom_Utilities` durchgeführt werden, die von der Schnittstelle aufgerufen wird. Insbesondere wird aktuell angenommen, dass alle ratenabhängigen Stoffmodelle ohne zusätzliche Korrekturterme im verwendeten Bezugssystem integriert werden können.

Teilweise stellen aufrufende Programme wie Abaqus zusätzliche Funktionalitäten in Form von Hilfsroutinen bereit (siehe bspw. Abschnitt 4.3.1). Es wird empfohlen, stattdessen entweder vorhandene Hilfsfunktionen, wie `Matrix_Rotation()` anstatt `RotSig()`, zu verwenden, oder den Aufruf in Hilfsfunktionen zu verarbeiten, wie `Write_Error_And_Exit()` zum Beenden der Berechnung.

Code 5.6: Beispielhafte Struktur von Schnittstellen basierend auf Implementierung der Fortran-Schnittstelle `xmat_console()`.

```
! ------------------------------------------------------------------ !
subroutine xmat_console( &
   ! ============= variables passed in for information ============= !
      materialname, nparams, materialparameters, nstatevar, statevariables, &
      ncomponents, oldstress, oldstrain, timeincrement, totaltime, &
   ! ==================== user coding to define ==================== !
      newstress, newstate, jacobian)
! ------------------------------------------------------------------ !
   use General_Settings, only: dp, setting_len_id, setting_solver_default, Estimate_Components, &
                               Uppercase, Check_Input_Dimensions, Write_Error_And_Exit
   use Debug, only: Formatval
   use Math_Operations, only: Nonzero_Division, Import_Matrix, Export_Matrix, Export_Tensor
   use Xmat_Class, only: Xmat, Xmat_Initialize
   implicit none
   !
   character(len=80), intent(in) :: materialname
   integer, intent(in) :: nparams, nstatevar
   real(dp), dimension(nparams), intent(in) :: materialparameters
   real(dp), dimension(nstatevar), intent(in) :: statevariables
   integer, intent(in) :: ncomponents
   real(dp), dimension(ncomponents), intent(in) :: oldstress, oldstrain
   real(dp), intent(in) :: timeincrement, totaltime
   real(dp), dimension(ncomponents), intent(out) :: newstress
   real(dp), dimension(nstatevar), intent(out) :: newstate
   real(dp), dimension(ncomponents, ncomponents), intent(out) :: jacobian
   ! ------------------------------------------------------------------ !
   class(Xmat), allocatable :: xmat_obj
   real(dp), dimension(__tensor__) :: jacobian_internal
   real(dp), dimension(__matrix__) :: inp_stress, inp_dot_strain
   real(dp), dimension(ncomponents) :: tmp_dot_strain
   character(len=setting_len_id) :: identifier
   character(len=13), parameter :: internname = 'xmat_console '
   real(dp) :: exportdt
   integer :: num_dimensions, num_shear
   logical :: firstcall

   ! Determine a supported set of direct and shear components out of ncomponents
   call Estimate_Components(nel=ncomponents, num_dimensions=num_dimensions, num_shear=num_shear)
   if (.not. Check_Input_Dimensions(num_dimensions=num_dimensions, num_shear=num_shear)) then
      call Write_Error_And_Exit('xmat_console: ' // Formatval('ncomponents', ncomponents) &
         // ' can not be converted to a suitable set of direct and shear components')
   end if
```

```
43
44    ! --- Assign to custom precision variables (consider element positioning)
45    identifier = Uppercase(text=materialname(1:setting_len_id))
46
47    inp_stress = Import_Matrix(mat=real(oldstress, dp), num_dimensions=num_dimensions, num_shear=num_shear)
48    tmp_dot_strain = Nonzero_Division(val=real(oldstrain, dp), fac=timeincrement)
49    inp_dot_strain = Import_Matrix(mat=tmp_dot_strain, num_dimensions=num_dimensions, num_shear=num_shear)
50
51    firstcall = .False.
52    if (totaltime <= timeincrement) then
53       firstcall = .True.
54    end if
55
56    call Xmat_Initialize(xmat_obj=xmat_obj, solver_name=setting_solver_default, &
57       constitutive_model_name=identifier, material_parameters=materialparameters, &
58       calculate_jacobian=.True., firstcall=firstcall)
59    call xmat_obj%Import_State(nstates=nstatevar, statevariables=statevariables, stress=inp_stress, &
60       dot_strain=inp_dot_strain, totaltime=totaltime, timeincrement=timeincrement)
61    call xmat_obj%Calculate_Results(exportstress=inp_stress, exportstates=newstate, &
62       exportjacobian=jacobian_internal, exportdt=exportdt)
63
64    ! --- Reassign to calling program variables
65    newstress = Export_Matrix(mat=inp_stress, num_dimensions=num_dimensions, num_shear=num_shear)
66    jacobian = Export_Tensor(tens=jacobian_internal, num_dimensions=num_dimensions, num_shear=num_shear)
67 end subroutine xmat_console
```

5.6 Unit-Tests

The higher your test coverage, the less your fear.
You can make changes with near impunity . . .

Martin (2009), S. 124

Unit-Tests dienen dazu, die geforderte Funktionalität einzelner Komponenten durch einfache Tests zu überprüfen.[6] Nach Martin (2009), S. 132, sollen Unit-Tests dem *FIRST*-Prinzip genügen, d. h.

- ***F**ast*: einfach und schnell ausführbar und überprüfbar sein,
- ***I**ndependent*: keine Abhängigkeiten zu anderen Testabfragen haben,
- ***R**epeatable*: jederzeit und in verschiedenen Umgebungen wiederholbar sein,
- ***S**elf-Validating*: durch eine Rückgabe von `.True.` oder `.False.` einfach überprüfbar sein und
- ***T**imely*: zeitlich mit (bzw. direkt vor) dem eigentlichen Code geschrieben werden.

[6] Eine detaillierte Darstellung findet sich in Martin (2009), S. 121–133.

Im Rahmen dieser Arbeit ist das Fortran Unit Test Framework FRUIT[7] zum Einsatz gekommen. Mit der Datei `fruit.f90` ist das Modul `fruit` verfügbar, das die Überprüfungsmechanismen `assert_equals()` und `assert_not_equals()` bereitstellt.[8] Diese Hauptfunktionen unterstützen bereits die Variablentypen `logical`, `character`, `integer`, `real`, `double precision` und `complex`. Für die Überprüfung der letzten drei Typen können Toleranzen angegeben werden, um Gleitkommazahlen nicht auf exakte Gleichheit prüfen zu müssen (siehe auch Abschnitt 2.4.4). Tabelle 5.2 listet die Unterschiede bei den Aufrufen für Skalare, Vektoren und Matrizen auf.

Code 5.7 zeigt die zentralen Teile des Testprogramms, in dem die relevanten Module geladen und in einzelnen Testszenarien überprüft werden. Zur Durchführung der Unit-Tests muss das Modul mit `use fruit` geladen und mit `init_fruit()` initialisiert werden (Zeile 2 und 6). Optional können die Überprüfungen mit `run_test_case()` in einzelne Testszenarien gegliedert werden. Die eigentlichen Überprüfungen werden mit `assert_equals()` und `assert_not_equals()` durchgeführt.

Tabelle 5.2: Die Funktionen `assert_equals()` und `assert_not_equals()` können für verschiedene Variablentypen und Dimensionen (Skalar, Vektor, Matrix) genutzt werden. S4-Tensoren werden für Vergleiche im Rahmen dieser Arbeit auf Matrizen projiziert. Für die Typen `real`, `double precision` und `complex` kann die Toleranz mit dem Argument `delta` angegeben werden (`message` ist optional, aber hilfreich für aussagekräftige Fehlermeldungen).

Vergleichswerte	Funktionsaufruf
Skalare	`call assert_equals(var1`$^{\text{in}}$`, var2`$^{\text{in}}$`, delta`$^{\text{in}}$`, message`$^{\text{in}}$`)`
Vektoren in $\mathbb{R}^n$	`call assert_equals(var1`$^{\text{in}}$`, var2`$^{\text{in}}$`, n`$^{\text{in}}$`, delta`$^{\text{in}}$`, message`$^{\text{in}}$`)`
Matrizen in $\mathbb{R}^{n\times m}$	`call assert_equals(var1`$^{\text{in}}$`, var2`$^{\text{in}}$`, n`$^{\text{in}}$`, m`$^{\text{in}}$`, delta`$^{\text{in}}$`, message`$^{\text{in}}$`)`

Code 5.7: Struktur des Testprogramms zur Durchführung der Unit-Tests. Die eigentlichen Überprüfungen sind in einzelne Module gekapselt und werden in Form von Testszenarien geprüft, wie für `Math_Operations` in Zeile 3 und 8 dargestellt. Ähnliche Einträge sind für alle relevanten Module notwendig.

```
1  program xmat_fruit
2     use fruit
3     use math_operations_test
4     ! ...
5
6     call init_fruit()
7     ! ...
8     call run_test_case(tc=Math_Operations_Testing, tc_name='Testing "Math Operations"')
9     write(*, '(a)') ' <- Math Operations'
10    ! ...
11    call fruit_summary()
12 end program xmat_fruit
```

[7] FRUIT ©A. H. Chen: »https://sourceforge.net/projects/fortranxunit/

[8] Alternativ stehen die gleichwertigen Funktionen `assertEquals()` und `assertNotEquals()` zur Verfügung.

Die übergebenen Funktionsnamen verweisen auf die Hauptfunktion der jeweiligen Module, in der die Koordination der Tests geregelt ist (siehe Code 5.8). Im Rahmen dieser Arbeit werden die Namen so gewählt, dass die dazugehörigen Module um `_Test` und die Hauptfunktion um `_Testing` ergänzt werden, wie bspw. `Math_Operations_Testing`. Die Subroutine, in denen eine Funktion getestet wird, hat den Namen der Funktion mit einem abschließendem `_test`. Wie dargestellt ist das Ziel von Unit-Tests die Überprüfung aller Funktionalitäten, insbesondere der zahlreichen Hilfsfunktionen. Die Aufrufe der Benutzerroutine über verschiedene Schnittstellen werden in Abschnitt 7 untersucht.

In Code 5.9 ist ein Ausschnitt der Testfunktion `Value_In_Interval_Test` dargestellt. Die einzelnen Tests von `Value_In_Interval` werden, wie dargestellt, über Aufrufe von `assert_equals()` durchgeführt.

Einige interne Funktionen der xMat sind in Modulen als `private` deklariert und bei normaler Verwendung nicht von außerhalb nutzbar. Um Überprüfungen an allen Funktionen durchführen zu können, wird für Unit-Tests eine modifizierte Version der xMat verwendet, die freien Zugriff auf alle Funktionalitäten erlaubt.

Code 5.8: Aufbau von `Math_Operations_Test`. In Zeile 2 und 3 sind alle notwendigen Importe aus der modifizierten xMat zum Testen der Funktionen aus `Math_Operations`. Die Testfunktionen werden in `Math_Operations_Testing()` aufgerufen.

```
1  module Math_Operations_Test
2     use General_Settings, only: dp, setting_epsilon
3     use Math_Operations
4     use fruit
5     implicit none
6
7     private
8     public :: Math_Operations_Testing
9
10    contains
11    ! ...
12    subroutine Math_Operations_Testing()
13    ! call to various test functions
14    end subroutine Math_Operations_Testing
15    ! ...
16 end module Math_Operations_Test
```

Code 5.9: Ausschnitt der Testfunktion für `Value_In_Interval`. Eindeutige `message`-Argumente erlauben bei Misserfolg eine schnelle Zuordnung.

```
1  subroutine Value_In_Interval_Test()
2     call assert_equals(var1=.True., &                    ! Support upper bound before lower bound
3        var2=Value_In_Interval(value=5.0_dp, limits=[7.0_dp, 4.0_dp]), &
4        message='Value_In_Interval(value=5.0_dp, limits=[7.0_dp, 4.0_dp])')
5
6     call assert_equals(var1=.True., &                    ! Support negative numbers
7        var2=Value_In_Interval(value=-6.002_dp, limits=[-7.0_dp, -6.0_dp]), &
8        message='Value_In_Interval(value=-6.002_dp, limits=[-7.0_dp, -6.0_dp])')
9     ! ...
10 end subroutine Value_In_Interval_Test
```

6 Untersuchte Materialien und deren Parameter

In diesem Kapitel werden die in dieser Arbeit verwendeten Materialien vorgestellt. Zum einen werden ein Mittelsand (Hamburger Sand, siehe Abschnitt 6.1) und eine Glaskugelmischung verwendet (siehe Abschnitt 6.2). Für diese beiden Materialien sind Versuche von Autor und Laborteam am Institut für Geotechnik und Baubetrieb durchgeführt worden. Zum anderen werden für die Vergleiche in Kapitel 7 Parametersätze für Karlsruher Feinsand und Hostun-Sand verwendet (siehe Abschnitt 6.4), für die alle Daten und Ergebnisse aus Literaturquellen entnommen sind (vor allem aus Wichtmann und Triantafyllidis, 2016; und Schranz, 2018). Die Korngrößenverteilungen der Materialien sind in Bild 6.1 zu sehen. Abschnitt 6.5 thematisiert die nachträgliche Anpassung der Parametersätze.

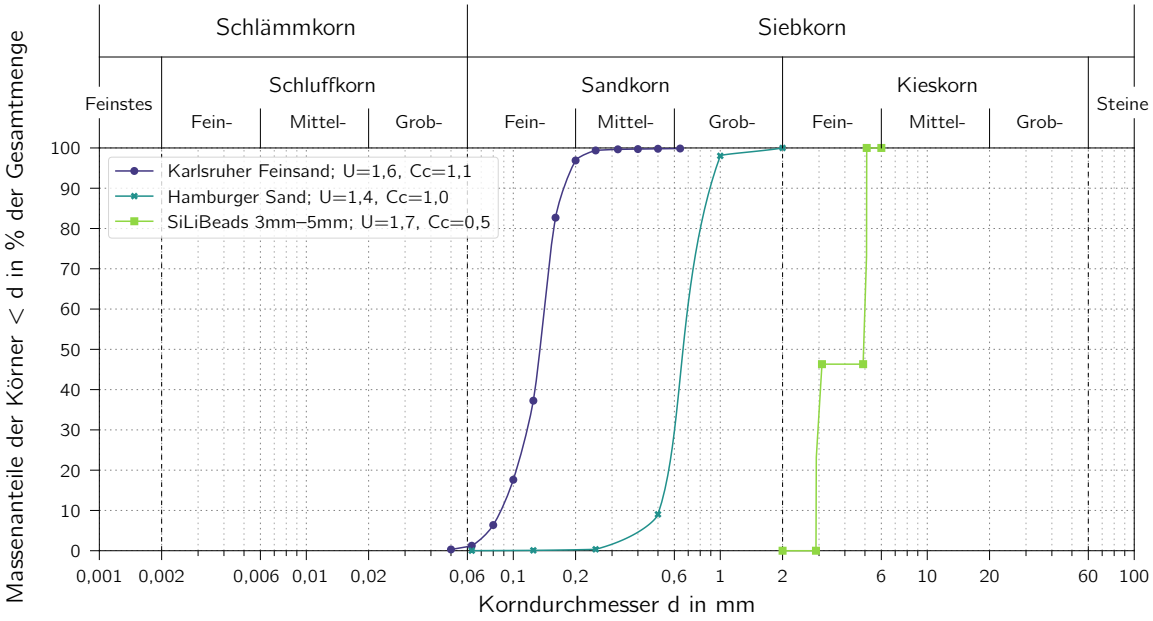


Bild 6.1: Sieblinie von Karlsruher Feinsand, Hamburger Sand und der Glaskugelmischung nach DIN EN ISO 17892-4:2017. Für die Glaskugelmischung sind nur Kugeln mit 3 mm und 5 mm Durchmesser verwendet worden.

Zur Bestimmung der Materialparameter

Materialparameter repräsentieren die Eigenschaften eines spezifischen Materials im Rahmen eines Stoffmodells. Typischerweise werden geotechnische Laborversuche durchgeführt, um daraus Materialparameter abzuleiten. In DGGT (2014), S. 18, wird empfohlen, für eine höhere Aussagekraft mehrere Verfahren zur Bestimmung einzelner Parameter zu nutzen.

Aus technischer Sicht wäre es ideal, wenn die Materialparameter weder vom Zustand des Bodenkörpers noch von den anderen Materialparametern abhängen. Typischerweise liegen aber beide Abhängigkeiten vor und es ist eine Idealisierung, die Materialparameter für alle in einer Untersuchung betrachteten Zustände als konstant anzusehen. Laut Walz (2006), S. 64, hängen maßgebende Bodenkennwerte, wie z. B. der Reibungswinkel oder das Steifemodul, vom Spannungszustand ab. Auch die hypoplastischen Parameter h_s und n hängen von zwei Spannungszuständen und Porenzahlen eines Ödometerversuchs mit locker gelagerter Bodenprobe ab. So empfiehlt Herle (1997), S. 34, für einen aussagekräftigen Parametersatz einen ausreichend großen Bereich zu wählen.

Zusätzlich sei auf die Unsicherheit bei Durchführung der Versuche und Erhebung der Messdaten in Abschnitt 2.2.3 verwiesen, aus denen die Materialparameter bestimmt werden. Speziell zu Unsicherheiten und Fehlern bei der Bestimmung der hypoplastischen Parameter mit Laborversuchen sei auf Herle (1997), S. 56–59, verwiesen. Es sollte also kritisch geprüft werden, unter welchen Bedingungen ein Satz an Materialparametern entstanden ist und ob er für eigene Untersuchungen verwendet werden kann.

miniSoilLAB

miniSoilLAB[1] ist im Rahmen dieser Arbeit erstellt worden, um einfacher auf die Ergebnisse der Laborversuche am Institut für Geotechnik und Baubetrieb zugreifen und diese miteinander vergleichen zu können.

Dabei kann miniSoilLAB auf zwei Arten verwendet werden: Entweder als grafische Anwendung oder als Python-Modul über die Konsole. miniSoilLAB ist in Python geschrieben und benötigt eine Python 3-Umgebung. Neben Modulen der Standardbibliothek sind `xlrd` oder `openpyxl` zum Lesen der xls-Dateien und `matplotlib` zur Erstellung der Plots erforderlich. Es ist betriebssystemunabhängig programmiert und unter Linux und Windows getestet worden.

Mit miniSoilLAB können aus den Daten der durchgeführten Laborversuche direkt die Parameter extrahiert werden, sofern die notwendigen Versuche durchgeführt worden sind. Unter anderem können für nichtbindige Böden die hypoplastischen und erweiterten hypoplastischen Parameter abgeschätzt werden.[2]

[1] miniSoilLAB ©D. Zobel: »https://github.com/d-zo/miniSoilLAB

[2] Ebenso ist für bindige Böden eine Abschätzung der viskohypoplastischen Parameter möglich.

6.1 Hamburger Sand

Hamburger Sand ist ein mittelsandiger Grobsand, der von der Heidelberger Sand und Kies GmbH aus dem Kieswerk Stade unter der Bezeichnung Quarzsand K4 vertrieben worden ist. Fotos der Sandoberfläche und einzelner Körner unter dem Lichtmikroskop sind in Bild 6.2 zu sehen.

Im Jahr 2016 ist eine interne Vergleichsstudie verschiedener Sande vom Laborteam des Instituts durchgeführt worden. Dabei ist u. a. für Hamburger Sand mehrmals die Korndichte (nach DIN EN ISO 17892-3:2016) und die lockerste und dichteste Lagerung (nach DIN 18126:2022) bestimmt worden. Zusammen mit Untersuchungen im Rahmen dieser Arbeit ergeben sich die in Tabelle 6.1 aufgelisteten Werte. Die angegebenen Unsicherheiten ermöglichen eine Einschätzung, in welchem Bereich eine Anpassung der Werte bspw. für die Parameterbestimmung von Stoffmodellen problemlos möglich ist. Der aus den Labordaten bestimmte Parametersatz für das Stoffmodell `Hypo-VW96` ist in Tabelle 6.2 dargestellt.

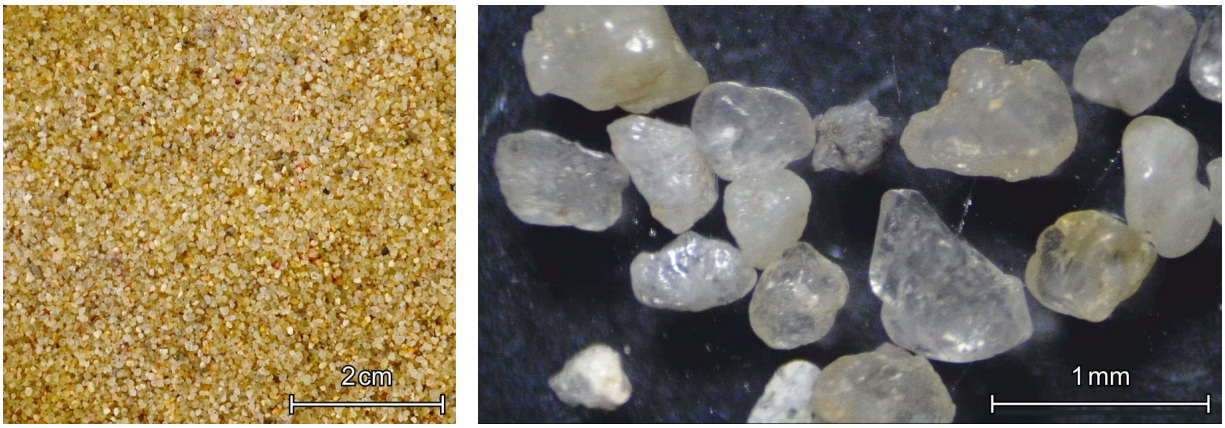


Bild 6.2: Fotografien von Hamburger Sand, angefertigt vom Laborteam des Instituts 2016. Links: Fotografie der Sandoberfläche; Rechts: Aufnahme einzelner Körner unter dem Lichtmikroskop.

Tabelle 6.1: Verwendete Referenzwerte für Hamburger Sand (basierend auf jeweils neun Datensätzen).

Korndichte (g/cm^3)	$\rho_{\min}$ (g/cm^3)	$e_{\max}$ (-)	$\rho_{\max}$ (g/cm^3)	$e_{\min}$ (-)
2,639 ($\sigma = 0{,}003$)	1,467 ($\sigma = 0{,}009$)	0,798 ($\sigma = 0{,}011$)	1,715 ($\sigma = 0{,}014$)	0,531 ($\sigma = 0{,}012$)

Tabelle 6.2: Materialparameter für Hamburger Sand mit dem Stoffmodell `Hypo-VW96` aus Labordaten und eigenen Untersuchungen.

φ_c (Grad)	h_s (kPa)	n_H	e_{d0}	e_{c0}	e_{i0}	α_H	β_H	m_T	m_R	$R_{\max}$	β_R	χ
31,0	$1{,}5 \cdot 10^6$	0,38	0,51	0,80	0,92	0,06	1,78	2,0	5,0	10^{-4}	0,5	6,0

6.2 Glaskugeln

Die verwendeten Glaskugeln SiLibeads Typ M sind polierte Glaskugeln aus Kalknatronglas und haben laut Hersteller eine Schüttdichte von 2,5 t/m^3. Der Hersteller gibt ein Elastizitätsmodul von 65 GPa, eine Toleranz von ±0,3 mm auf den Durchmesser und eine Rundheit ($r_{\min}/r_{\max}$) von mindestens 0,98 an.[3]

Die experimentelle Bestimmung der Korndichte nach DIN EN ISO 17892-3:2016 ergibt einen Wert von ca. 2,55 g/cm^3 und ist zusammen mit den Werten für die lockerste und dichteste Lagerung nach DIN 18126:2022 in Tabelle 6.3 eingetragen.[4] Der aus den Labordaten bestimmte Parametersatz ist in Tabelle 6.4 dargestellt.

Es wird eine Mischung mit 20,05 kg Glaskugeln mit 3 mm und 23,23 kg Glaskugeln mit 5 mm hergestellt (siehe Bild 6.3). In dieser Mischung entsprechen 46,3% des Probengewichts Glaskugeln mit einem Durchmesser von 3 mm und 53,7% des Probengewichts Glaskugeln mit einem Durchmesser von 5 mm. Da das Gewicht einer Kugel mit 5 mm dem Gewicht von $(5/3)^3$ Kugeln mit 3 mm entspricht, ist das Verhältnis von Kugeln mit 3 mm zu Kugeln mit 5 mm im Probenkörper ungefähr

$$\frac{0{,}463}{0{,}537} \cdot \left(\frac{5}{3}\right)^3 \approx 4:1 \quad .$$

Im Bereich der Kristallografie und Chemie gibt es Untersuchungen zum Packmaß von starren Kugeln mit zwei unterschiedlichen Größen. In Hopkins und Stillinger (2012) werden die dichtesten Packungen sowohl für $\alpha = r_s/r_l$, als auch für die Anzahl kleinerer Kugeln zur Gesamtanzahl $x = n_s/(n_l + n_s)$ betrachtet. In dem Artikel wird der Parameterbereich $0 \leq \alpha \leq 0{,}7$ und $0 \leq x \leq 1$ an 12 oder weniger Kugeln in periodischer Anordnung untersucht und grafisch aufgezeigt.

Tabelle 6.3: Verwendete Referenzwerte für die Glaskugelmischung (basierend auf jeweils drei Datensätzen). Durch die vergleichsweise großen Randporen wird ein 2 % kleineres Volumen bei der Bestimmung der dichtesten Lagerung angenommen (siehe auch Abschnitt 6.3).

Korndichte (g/cm^3)	$\rho_{\min}$ (g/cm^3)	$e_{\max}$ (-)	$\rho_{\max}$ (g/cm^3)	$e_{\min}$ (-)
2,553 ($\sigma = 0{,}001$)	1,530 ($\sigma = 0{,}001$)	0,67 ($\sigma = 0{,}001$)	1,630 ($\sigma = 0{,}005$)	0,566 ($\sigma = 0{,}004$)

[3] Siehe Datenblatt des Herstellers unter »https://www.sigmund-lindner.com/en/products/glass-beads/type-m-closures/

[4] Aus den Daten von Tabelle 6.3 ergibt sich eine geringste Packungsdichte von 0,6 und eine größte Packungsdichte von 0,626. Das liegt annähernd im Bereich $0{,}6 \leq \phi \leq 0{,}64$, der nach Experimenten von Scott (1960), S. 909, als Bereich der lockersten und dichtesten Lagerung einer beliebigen Befüllung („*random packing*“) gleichförmiger Kugeln angesehen wird. Bernal (1964), S. 311, weist darauf hin, dass die Packungsdichte bei Kugeln mit unterschiedlichen Radien (bzw. nicht-gleichförmigen Kugeln) im Allgemeinen höher ist.

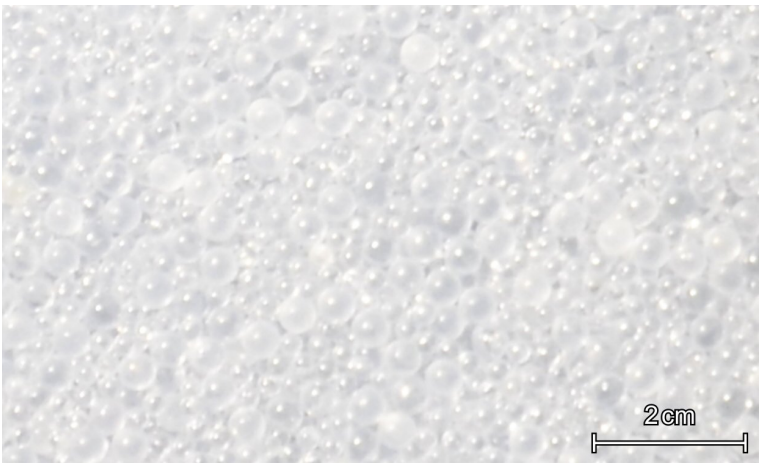


Bild 6.3: Aufnahme der Glaskugelmischung im Versuchsstand.

Tabelle 6.4: Materialparameter für die Glaskugelmischung mit dem Stoffmodell `Hypo-VW96`.

φ_c (Grad)	h_s (kPa)	n_H	e_{d0}	e_{c0}	e_{i0}	α_H	β_H	m_T	m_R	$R_{\max}$	β_R	χ
20,0	$80 \cdot 10^3$	0,7	0,56	0,67	0,77	0,01	1,76	2,0	5,0	10^{-4}	0,5	6,0

Für die vorliegende Glaskugelmischung ergeben sich die folgenden Kennwerte:

$$\alpha = \frac{r_s}{r_l} = \frac{1{,}5\,\mathrm{mm}}{2{,}5\,\mathrm{mm}} = 0{,}6 \qquad \text{und}$$

$$x = \frac{n_s}{n_l + n_s} = \frac{20{,}05\,\mathrm{kg} \cdot \left(\frac{5}{3}\right)^3}{23{,}23\,\mathrm{kg} + 20{,}05\,\mathrm{kg} \cdot \left(\frac{5}{3}\right)^3} \approx 0{,}8 \quad .$$

Aus der visuellen Darstellung der Ergebnisse (Bild 2 und 3 in Hopkins und Stillinger, 2012) kann angenommen werden, dass die größtmögliche, theoretische Packungsdichte $\phi = 1/(1+e)$ für eine derartige Kugelmischung zwischen 74 % und 76 % liegt ($e \approx 0{,}32$).

6.3 Randporen und Membrandurchdringungseffekt

Mit *Membrandurchdringungseffekt* wird das Anschmiegen einer Gummi- bzw. Latexmembran an eine Bodenprobe bei anliegendem Radialdruck und Entzug von Porenwasser bezeichnet, bei der die Kornstruktur bzw. Randporen von außen sichtbar werden („membrane compliance“ oder aus Sicht der Bodenpartikel *„membrane penetration“*, siehe Bild 6.4). Nach Nicholson u. a. (1993), S. 727, setzt sich die Volumenänderung ΔV einer gesättigten Bodenprobe mit Membran im Triaxialgerät aus zwei Komponenten zusammen: Der tatsächlichen volumetrischen Dehnung ε_V in Bezug auf das Gesamtvolumen V und einem Membrananschmiegungsfaktor δV_m bezogen auf die Membranfläche A_m:

$$\Delta V = \varepsilon_V V + \delta V_m A_m \quad .$$

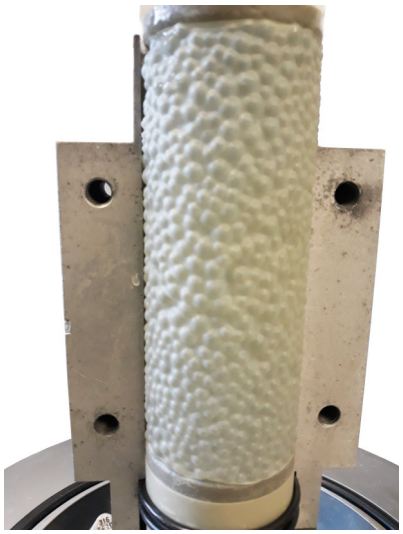


Bild 6.4: Glaskugelmischung in der Latexmembran im Triaxialversuchsstand. Das Porenwasser im Randbereich ist entzogen worden, sodass die Kugelstruktur gut erkannt werden kann (Membrandurchdringungseffekt).

In Nicholson u. a. (1993), S. 733, werden der anliegende Radialdruck und die Korngröße bzw. Korngrößenverteilung der Bodenprobe (insbesondere die starke Korrelation zu d_{20}) als wichtigste Einflussfaktoren auf den Membrandurchdringungseffekt bewertet. Anderen Faktoren (wie die Dichte der Bodenprobe, die Ungleichförmigkeit der Körner, die Art der Probenherstellung usw.) werden in der praktischen Anwendung eine untergeordnete Bedeutung zugeschrieben. Da der Zusammenhang dort zwischen δV_m und $\log_{10}(\sigma'_3)$ im betrachteten Druckbereich bis 400 kPa annähernd linear ist, wird der Membrandurchdringungseffekt mit einer normalisierten Änderungsrate S_M beschrieben. Mit Faktor 10 skaliert für eine Ausgabe in Millimeter lautet die Formel aus Nicholson u. a. (1993), S. 736,

$$S_M = 0{,}019 + 0{,}095 d_{20} - 0{,}000157 d_{20}^2 \quad . \tag{6.1}$$

Damit kann abgeschätzt werden, welcher Anteil $\Delta V_{Rp}/V$ des Gesamtvolumens durch die Randporen eingenommen wird. Die normalisierten Änderungsraten S_M und die Abschätzung der Randporenanteile der verwendeten Materialien sind in Tabelle 6.5 dargestellt. Es kann gesehen werden, dass die Randporen bei der Glaskugelmischung zwischen 4 und 9 Prozent des Gesamtvolumens betragen.[5] Wie z. B. in Bergholz (2020), S. 226, angemerkt, muss dieser Effekt bei der Auswertung berücksichtigt werden, um das volumetrische Verhalten richtig zu interpretieren. Studien wie bspw. Ji u. a. (2021) weisen zusätzlich auf den Einfluss der als untergeordnet eingestuften Faktoren, wie der Lagerungsdichte der Bodenprobe, hin.

[5] Die Ergebnisse decken sich mit einer eigenen Abschätzung aufgrund der Geometrie, die in Anhang H dargestellt ist.

Tabelle 6.5: Normalisierte Änderungsrate S_M nach Nicholson u. a. (1993) für die verwendeten Materialien abhängig vom Durchmesser d_{20}. Zur Berechnung von δV_m kann S_M näherungsweise mit dem dekadischen Logarithmus der Druckdifferenz für den betrachteten Zustand multipliziert werden. Die angegebenen Anteile $\Delta V_{Rp}/V$ der Randporen am Gesamtvolumen ergeben sich aus einem durchschnittlichen Triaxialversuch mit Probenhöhe $h = 125\,\text{mm}$, Probenradius $R = 25\,\text{mm}$ und effektiven Drücken $p = 50$, 100 und 200 kPa.

Material	d_{20} (mm)	S_M (-)	$\Delta V_{Rp,50}/V$ (%)	$\Delta V_{Rp,100}/V$ (%)	$\Delta V_{Rp,200}/V$ (%)
SiLibeads 5 mm	5,0	0,490	6,66	7,84	9,02
SiLibeads 3 mm	3,0	0,303	4,12	4,85	5,58
Hamburger Sand	0,56	0,072	0,98	1,15	1,33
Karlsruher Feinsand	0,1	0,028	0,38	0,45	0,52

6.4 Parametersätze aus der Literatur

Karlsruher Feinsand

Für Karlsruher Feinsand sind u. a. in Wichtmann und Triantafyllidis (2016) und Wichtmann (2016), S. 13, zahlreiche Daten und Untersuchungen veröffentlicht worden.[6] Auch wenn keine eigenen Laborversuche mit diesem Sand durchgeführt worden sind, werden die Parameter in Kapitel 7 für die Nachrechnung von Ergebnissen aus der Literatur verwendet.

Bild 6.5 zeigt eine Aufnahme einzelner Körner des Karlsruher Feinsands und Tabelle 6.6 gibt eine Übersicht über Korndichte, Lagerungsdichten und Porenzahlen.

Der Parametersatz für Karlsruher Feinsand mit dem Stoffmodell `Hypo-VW96` ist in Tabelle 6.7 dargestellt.

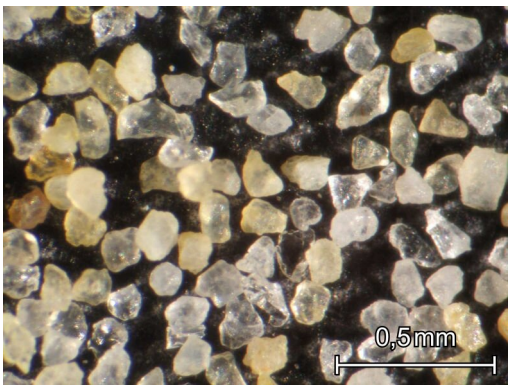


Bild 6.5: Aufnahme einzelner Körner des Karlsruher Feinsands unter dem Lichtmikroskop aus Wichtmann und Triantafyllidis (2016), S. 741.

[6] Daten zum Karlsruher Feinsand sind auch verfügbar unter »https://torsten-wichtmann.de/

Tabelle 6.6: Verwendete Referenzwerte für Karlsruher Feinsand basierend auf Daten aus Wichtmann und Triantafyllidis (2016) und Wichtmann (2016), S. 13.

Korndichte (g/cm^3)	$\rho_{\min}$ (g/cm^3)	$e_{\max}$ (-)	$\rho_{\max}$ (g/cm^3)	$e_{\min}$ (-)
2,635	1,290	1,054	1,580	0,677

Tabelle 6.7: Materialparameter für Karlsruher Feinsand mit dem Stoffmodell `Hypo-VW96` aus Wichtmann und Triantafyllidis (2016), S. 741.

φ_c (Grad)	h_s (kPa)	n_H	e_{d0}	e_{c0}	e_{i0}	α_H	β_H	m_T	m_R	$R_{\max}$	β_R	χ
33,1	$4{,}0 \cdot 10^6$	0,27	0,677	1,054	1,212	0,14	2,5	1,1	2,2	10^{-4}	0,1	5,5

Karlsruher Mittelsand

Da die Parameter für das Stoffmodell `Hypo-Wu92` von der Lagerungsdichte abhängig sind, unterscheiden sich die Parametersätze auch für das gleiche Material in verschiedenen Zuständen. In Wu (1992), S. 22 und S. 46, werden zwei Parametersätze vorgestellt, die für Nachrechnungen von dicht und locker gelagertem Karlsruher Mittelsand mit den Porenzahlen 0,53 und 0,81 verwendet werden. Diese Parameter werden auch in dieser Arbeit verwendet und finden sich in Tabelle 6.8. Weitere Parametersätze für die Porenzahlen 0,55 und 0,76 finden sich in Bauer (1992), S. 72.

Hostun-Sand

In den folgenden Tabellen 6.9 bis 6.12 sind Materialparameter aus Schranz (2018) und Kolymbas (2022) zusammengetragen. Die Materialien selbst werden nicht näher vorgestellt, da nur die Parameter für Nachrechnungen und Vergleiche in Kapitel 7 gebraucht werden.

Für das Stoffmodell `Hypo-VW96` werden die Materialparameter für Hostun-Sand mit den Anpassungen von G. Medicus aus Schranz (2018), S. 66, verwendet und sind in Tabelle 6.9 dargestellt. Diese Parameter unterscheiden sich von den Parametern aus Herle (1997), S. 54, für die $\varphi_c = 31\,\text{Grad}$, $e_{d0} = 0{,}61$, $e_{c0} = 0{,}91$ und $e_{d0} = 1{,}09$ angegeben ist. Im Vergleich zu den Parametern aus Herle (1997) wird in Herle und Gudehus (1999), S. 479, eine kritische Porenzahl $e_{c0} = 0{,}96$ verwendet.

Die verwendeten Materialparameter für Hostun-Sand mit dem Stoffmodell `Baro-Ko15` sind aus Schranz (2018), S. 66, entnommen. Im Vergleich dazu werden in Kolymbas (2015), S. 56 f., $c_3 = -2{,}3$ und $c_5 = 456\,\text{kPa}$ angegeben, aber weder e_{c0} noch $e_{\min}$. Die Parameter für Hostun-Sand mit dem Stoffmodell `Baro-Ko15` stammen ebenfalls aus Schranz (2018), S. 66.

Tabelle 6.12 zeigt die verwendeten Materialparameter für Hostun-Sand mit dem Stoffmodell `Baro-Ko21`. Die Parameter sind sowohl in Kolymbas (2021), als auch in Kolymbas (2022), S. 123, zu finden.

Tabelle 6.8: Parameter für dicht und locker gelagerten Karlsruher Mittelsand mit dem Stoffmodell `Hypo-Wu92` aus Wu (1992), S. 22 und S. 46.

Lagerungsdichte	C_1 (-)	C_2 (-)	C_3 (-)	C_4 (-)
dicht	−106,5	−801,5	−797,1	1077,7
locker	−33,3	−308,4	−306,8	321,3

Tabelle 6.9: Materialparameter für Hostun-Sand mit dem Stoffmodell `Hypo-VW96` aus Schranz (2018), S. 66.

φ_c (Grad)	h_s (kPa)	n_H	e_{d0}	e_{c0}	e_{i0}	α_H	β_H	m_T	m_R	$R_{\max}$	β_R	χ
33	$1{,}0 \cdot 10^6$	0,29	0,63	1,0	1,15	0,13	2,0	2,0	5,0	10^{-4}	0,5	6,0

Tabelle 6.10: Materialparameter für Hostun-Sand mit dem Stoffmodell `Baro-Ko15` aus Schranz (2018), S. 66, aber Parameter c_3 mit negativem Vorzeichen.

φ_c (Grad)	c_2 (-)	c_3 (-)	c_4 (-)	c_5 (-)	e_{c0} (-)	$e_{\min}$ (-)
33,8	1,0	−2,5076	1,0	40,0	0,91	0,5

Tabelle 6.11: Materialparameter für Hostun-Sand mit dem Stoffmodell `Baro-Sc18` und $p_{\text{ref}} = 1\,\text{kPa}$ aus Schranz (2018), S. 66.

φ_c (Grad)	K_r (-)	ξ (-)	κ (-)	e_{c0} (-)	c_e (-)	c_5 (-)	c_6 (-)	c_7 (-)
33,8	430,0	0,74	3,0	0,904	1,093	3,0	5,0	10,0

Tabelle 6.12: Materialparameter für Hostun-Sand mit dem Stoffmodell `Baro-Ko21` aus Kolymbas (2022), S. 123.

φ_c (Grad)	c_2 (-)	c_3 (-)	c_4 (-)	c_5 (-)	κ_1 (-)	κ_2 (-)	k_1 (-)	k_2 (-)	e_{c0} (-)
33,8	0,50	−0,20	5000,0	0,50	30,0	0,0	1,0	0,02	0,54

6.5 Optimierung der Materialparameter

Bei der Optimierung von Materialparametern wird eine möglichst hohe Übereinstimmung zu einem idealisierten Modell gesucht. Dabei muss beachtet werden,

- welches Verhalten das Stoffmodell tatsächlich abbilden kann,
- welche Werte sinnvoll miteinander verglichen werden können und
- ob die untersuchten Parameter überhaupt einen repräsentativen Rückschluss auf das allgemeine Verhalten zulassen.

Zusätzlich bietet eine gleichzeitige Optimierung mehrerer Parameter viele Herausforderungen. So können die Ergebnisse einer Optimierung stark von der Ausgangssituationen abhängen und auf verschiedene, lokale Optima konvergieren. Dabei ist es im Allgemeinen nicht möglich zu entscheiden, ob ein gefundenes (lokales) Optimum tatsächlich das globale Optimum ist. Ein optimierter Parametersatz kann eine Überanpassung an die Ausgangssituationen darstellen und sich nicht gut auf andere Versuche übertragen lassen.

Hajihassani u. a. (2018) geben einen Überblick über die Verwendung evolutionärer Algorithmen, wie bspw. *„Particle swarm optimization“*, in der Geotechnik. In Mendez u. a. (2021) wird aufgezeigt, wie evolutionäre Algorithmen zur Kalibrierung der Materialparameter der Hypoplastizität genutzt werden können (und verweist auf das ExCalibre[7]-Werkzeug zur Kalibrierung). Machaček u. a. (2022) untersuchen die automatische Kalibrierung der Materialparameter für das hypoplastische Stoffmodell nach von Wolffersdorff (1996). Cekerevac u. a. (2006) stellen eine Optimierungsstrategie für elasto-plastische Stoffmodelle vor, was Gemeinsamkeiten zum folgenden Optimierungswerkzeug hat.

MPO

MPO (Material-Parameter-Optimierung)[8] ist ein Werkzeug zur Untersuchung von Parametersätzen. Es wurde im Rahmen dieser Arbeit programmiert, um automatisiert den Einfluss einer Variation von Materialparametern zu untersuchen. Dabei wird der Unterschied zwischen einem nachgerechneten Versuch zu einem Referenzversuch nach einem wählbaren Kriterium an diskreten Stellen ermittelt. Neben der Methode der kleinsten Fehlerquadrate hat sich die Berechnung der Differenzfläche zwischen beiden Kurven als Kriterium bewährt. Typischerweise werden mehrere, ggf. unterschiedlich gewichtete Referenzversuche betrachtet. So kann aus den angegebenen Variationen der Satz an Materialparametern mit der größten Übereinstimmung ermittelt werden.

In MPO sind zwei Variationsansätze verfügbar: Zum einen können die Materialparameter direkt variiert werden, sodass die Änderung einer Größe sichtbar gemacht werden kann. Diese Vorgehensweise hat den Nachteil, dass die Materialparameter im Allgemeinen nicht mehr konsistent sein müssen. So wird bspw. für Stoffmodelle wie die Hypoplastizität nach von Wolffersdorff (1996) nicht geprüft, ob die Zusammenhänge noch erfüllt sind, die für die

[7] ExCalibre ©D. Mašin: »https://soilmodels.com/excalibre/
[8] MPO ©D. Zobel: »https://github.com/d-zo/MPO

Bestimmung der Materialparameter genutzt worden sind (siehe bspw. S. 31–32 in Abschnitt 2.3.3). Deshalb bietet sich diese Variante in Situationen an, in denen entweder einzelne Werte unabhängig vom Rest sind oder konsistente Kombinationen vorab bestimmt werden können.

Zum anderen können die Versuchsdaten bzw. Eingangswerte zur Berechnung der Parameter variiert werden. Unterschiedliche Offsets oder Referenzpunkte führen im Allgemeinen zu unterschiedlichen Sätzen an konsistenten Materialparametern. Der Nachteil dieser Methode ist die starke Abhängigkeit von den verwendeten Versuchsdaten. Je nach Stoffmodell und Datenlage können einzelne Materialparameter stark variieren, während andere kaum beeinflusst werden.

MPO selbst in Python geschrieben und benötigt eine Python 3-Umgebung. Neben Modulen der Standardbibliothek ist nur `matplotlib` zur Erstellung der Plots erforderlich. Die Simulationen selbst werden in Fortran durchgeführt. MPO ist betriebssystemunabhängig programmiert und unter Linux und Windows getestet worden. Auch wenn die Hauptentwicklung von Untersuchungen mit dem hypoplastischen Stoffmodell geprägt ist, können ebenso Parameter anderer Stoffmodelle untersucht werden.

7 Überprüfungen

'The Computer is certainly the perfect instrument for inevitable research',
that is, research which is certain to deliver some *'results',*
right or wrong or meaningless, on any problem proposed.

C. Truesdell zitiert H. R. Post, Truesdell (1984), S. 601

In diesem Kapitel wird die grundsätzliche Funktionalität der xMat aufgezeigt. Die gewählte Vorgehensweise wird in Abschnitt 7.1 beschrieben. Anschließend liegt der Fokus auf der Überprüfung einzelner Komponenten, beginnend mit dem Stoffmodell `Hypo-VW96` in Abschnitt 7.2.

Wichtige Erkenntnisse aus diesem Kapitel sind:

Funktionsweise

- Die xMat kann bereits für viele einfache Untersuchungen eingesetzt werden und
- mit der Testumgebung können Änderungen im Code schnell mit anderen Ergebnissen verglichen und überprüft werden.

Einschränkungen

- Einige Unterschiede zu anderen Benutzerroutinen können aktuell noch nicht erklärt werden und
- deshalb sind noch weitere Tests und Anpassungen an der xMat notwendig, bevor ein Produktiveinsatz möglich ist.

7.1 Teststrategie

Zuerst werden die implementierten Stoffmodelle über die `UMAT`-Schnittstelle aus einfachen Fortran-Programmen untersucht. Dabei werden die Eigenschaften aller Stoffmodelle an ausgewählten Berechnungen aufgezeigt.

Dann werden die Berechnungen von Triaxial- und Ödometerversuchen mit den Stoffmodellen der xMat mit Referenzdaten aus der Literatur verglichen. Für das Stoffmodell `Hypo-VW96` werden auch die Ergebnisse der in Kapitel 3.1 vorgestellten Benutzerroutinen betrachtet.

Im Vorfeld sind die Benutzerroutinen so angepasst worden, dass vergleichbare Ergebnisse erwartet werden können. Insbesondere werden

- die Skalierungsfaktoren t_S für das erste Zeitinkrement der numerischen Integration einheitlich auf 10^{-5} gesetzt,
- alle Zuweisungen und Rechenoperationen an die von der Schnittstelle erwartete Reihenfolge der Komponenten angepasst (Berechnung der Determinante in `ABQS04`) und
- die Beschränkung in `ABQE10` auf maximal 100 Subinkremente in einem Integrationsschritt deaktiviert.

Die Untersuchungen in der Triaxialebene stellen zwar einen wichtigen Beitrag dar, aber weitere Überprüfungen zeigen das Verhalten für beliebige, frei gewählte Spannungszustände auf. Dazu gehören u. a. Druckspannungszustände mit Schubspannungen, kleine Druckspannungen nahe null und Zugspannungen. Anschließend folgt eine Untersuchung der bereitgestellten Schnittstellen und Integrationsverfahren.

ElementSimulator

ElementSimulator[1] ist ein Hilfsprogramm, mit dem Untersuchungen an einem idealisierten Bodenelement durchgeführt werden können. Inspiriert von Incremental Driver[2] sind Berechnungen für eine Impulsantwort, idealisierte Elementversuche (Ödometerversuch, Triaxialversuch und Einfachscherversuch) sowie Spannungspfade oder Antwortumhüllende vordefiniert. Im direkten Vergleich zu Incremental Driver sollte die Modifikation der vorhandenen Versuche einfacher sein, aber das Hinzufügen neuer Versuche ist aufwändiger. Im Gegenzug werden neben der `UMAT`-Schnittstelle auch die `VUMAT`- und die Fortran-Schnittstelle unterstützt. ElementSimulator kann einige Versuche als Simulationen in Abaqus/Standard oder Abaqus/Explizit ausführen und es kann im Allgemeinen mehr als eine Benutzerroutine im Rahmen einer Auswertung betrachtet werden.

ElementSimulator ist in Python geschrieben und benötigt eine Python 3-Umgebung. Neben Modulen der Standardbibliothek ist nur `matplotlib` zur Erstellung der Plots erforderlich. Die Simulationen selbst werden entweder in Fortran oder in Abaqus durchgeführt, aber die Auswertung erfolgt wieder in Python. ElementSimulator ist betriebssystemunabhängig programmiert und unter Linux und Windows getestet worden.

Mit dem ElementSimulator werden in diesem Kapitel die folgenden vier Untersuchungen durchgeführt:

- Ödometerversuche mit vier Belastungsphasen und den Drücken $-10\,\mathrm{kPa}$, $-100\,\mathrm{kPa}$, $-50\,\mathrm{kPa}$ und $-200\,\mathrm{kPa}$. Die Referenzdehnungsrate in vertikaler Richtung ist $D_{\mathrm{ref}} = -0{,}05$,
- Triaxialversuche mit einem Zelldruck von $-100\,\mathrm{kPa}$, einer maximalen Stauchung von $20\,\%$ und einer Referenzdehnungsrate von $D_{\mathrm{ref}} = -1$,

[1] ElementSimulator ©D. Zobel: »https://github.com/d-zo/ElementSimulator
[2] Incremental Driver ©A. Niemunis: »https://soilmodels.com/idriver/

- die Berechnung der Spannungspfade aufgrund unterschiedlicher Richtungen einer vorgegebenen Referenzdehnungsrate von $D_{\text{ref}} = -1$. Ausgehend vom hydrostatischen Spannungszustand $T_{ii} = -400\,\text{kPa}$ und $T_{ij} = 0\,\text{kPa}$ oder von einem Spannungszustand $T_{11} = -600\,\text{kPa}$, $T_{22} = T_{33} = -300\,\text{kPa}$ und $T_{ij} = 0\,\text{kPa}$ werden die Pfade für 12 Dehnungsänderungen in ebenso vielen Richtungen betrachtet und
- die Berechnung der Antwortumhüllenden. Für fünf gegebene Spannungszustände mit $p = 100\,\text{kPa}$ wird in 64 Richtungen jeweils ein Schritt mit der Referenzdehnungsrate $D_{\text{ref}} = -5$ berechnet. Neben dem hydrostatischen Zustand $T^C_{ii} = -100\,\text{kPa}$ sind die gegebenen Spannungszustände

$$\mathbf{T}^A = \begin{pmatrix} -176 & \cdot & \cdot \\ \cdot & -62 & \cdot \\ \cdot & \cdot & -62 \end{pmatrix}, \qquad \mathbf{T}^B = \begin{pmatrix} -138 & \cdot & \cdot \\ \cdot & -81 & \cdot \\ \cdot & \cdot & -81 \end{pmatrix},$$
$$\mathbf{T}^D = \begin{pmatrix} -72 & \cdot & \cdot \\ \cdot & -114 & \cdot \\ \cdot & \cdot & -114 \end{pmatrix} \quad \text{und} \quad \mathbf{T}^E = \begin{pmatrix} -44 & \cdot & \cdot \\ \cdot & -128 & \cdot \\ \cdot & \cdot & -128 \end{pmatrix}.$$

  Fünf weitere Spannungszustände entsprechen mit $p = 200\,\text{kPa}$ jeweils den doppelten Werten.

Der Zeitschritt entspricht in allen Untersuchungen $dt = 10^{-4}$. Die Winkelangaben γ für die Antwortumhüllenden und die Spannungspfade sind relativ zur Belastungsrichtung einer isotropen Kompression angegeben (siehe Bild 2.8 auf S. 25). Ausgehend von einer konstanten Dehnungsrate D_{ref} mit $\gamma_{\text{iso}} = \operatorname{atan}\left(\sqrt{2}\right)$ ergibt sich die Dehnungsrate

$$\mathbf{D} = D_{\text{ref}} \begin{pmatrix} \cos(\gamma + \gamma_{\text{iso}}) & \cdot & \cdot \\ \cdot & \frac{1}{\sqrt{2}} \sin(\gamma + \gamma_{\text{iso}}) & \cdot \\ \cdot & \cdot & \frac{1}{\sqrt{2}} \sin(\gamma + \gamma_{\text{iso}}) \end{pmatrix}.$$

Damit entsprechen 0 und 180 Grad einer isotropen Kompression `+i` bzw. Extension `-i`, sowie 90 und 270 Grad der triaxialen Kompression `+c` bzw. Extension `-c`.

Im Folgenden ist ein beispielhafter Aufruf von `ElementSimulator` unter Verwendung der Fortran-Programme zu sehen, wobei `<einzeltest>` eine der folgenden Optionen sein kann: `impuls`, `oedo`, `triax`, `pfade` oder `ellipsen`.

```
python ElementSimulator.pyz -test=<einzeltest> -prog=fortran -zusatz=normal
```

Mit ElementSimulator können auch Simulationen in Abaqus/Standard oder Abaqus/Explizit durchgeführt werden. Dabei muss `<einzeltest>` entweder `oedo`, `triax` oder `scher` sein.

```
python ElementSimulator.pyz -test=<einzeltest> -prog=abaqus -zusatz=normal
```

7.2 Untersuchung der Stoffmodelle

Für die Ödometer- und Triaxialversuche in diesem Abschnitt werden die gleichen Einstellungen genutzt, wie sie für den ElementSimulator in Abschnitt 7.1 aufgelistet sind. Nur bei Abweichungen wird entsprechend auf die geänderten Werte verwiesen. Das Verhalten der Stoffmodelle der xMat wird über die `UMAT`-Schnittstelle mit dem expliziten Euler-Verfahren untersucht.

7.2.1 Eigenschaften der Stoffmodelle in ausgewählten Berechnungen

In den Bildern 7.1 bis 7.6 sind die Ergebnisse der Versuche im ElementSimulator für die verschiedenen Stoffmodelle dargestellt. Für das Stoffmodell `Hypo-VW96` werden Berechnungen mit mehreren Benutzerroutinen durchgeführt, die alle dargestellt sind. Sofern das Konzept der intergranularen Dehnungen aktiviert ist, werden die Ödometerversuche und die Antwortumhüllenden bei diesem Stoffmodell mit voll ausgebildeter intergranularer Dehnung in die Belastungsrichtung angenommen.

In den Ödometerversuchen kann bei `Hypo-Wu92` und `Hypo-VW96` ohne intergranulare Dehnung die typische Akkumulation der Verformung zwischen Belastung und Wiederbelastung gesehen werden. Die deviatorische Spannung in Berechnungen mit dem Stoffmodell `Hypo-Wu92` ist nach Erreichen des Peak-Zustands konstant. Dadurch ergibt sich mit zunehmender Stauchung eine lineare Änderung der volumetrischen Dehnung.

Für das Stoffmodell `Hypo-VW96` sind die Berechnungen auch mit anderen Benutzerroutinen durchgeführt worden. Es gibt keine nennenswerten Unterschiede zwischen den Benutzerroutinen für die dargestellten Versuche. Bei den anderen Schnittstellen ist nur die Implementierung der xMat gezeigt, aber Vergleiche folgen in Abschnitt 7.2.2.

Der Effekt der intergranularen Dehnung mit dem Stoffmodell `Hypo-VW96` kann vor allem am Ödometerversuch gesehen werden. Der Einfluss auf die Spannungsantwort ist am größten für abrupte Änderungen der Belastungsrichtung, wie dem Wechsel von Belastung auf Entlastung und von Entlastung auf Wiederbelastung. Auch bei den Antwortumhüllenden kann ein Unterschied erkannt werden, sofern, wie dargestellt, die Richtung der intergranularen Dehnung gleich bleibt und nicht an die Richtung der Dehnung angepasst wird.

Im Kontrast dazu weisen die Antwortumhüllenden bei `Baro-Ko15` und `Baro-Sc18` ausgeprägte Spitzen auf. Bei einem Boden mit isotropen Eigenschaften sind eher gleichmäßigere Übergänge zu erwarten, wie an den Ellipsen der restlichen Stoffmodelle gesehen werden kann. Auch im Vergleich der Spannungspfade fällt auf, dass die Stoffmodelle `Baro-Ko15` und `Baro-Sc18` ein anderes Verhalten zeigen als `Hypo-Wu92` und `Hypo-VW96` und `Baro-Ko21`.

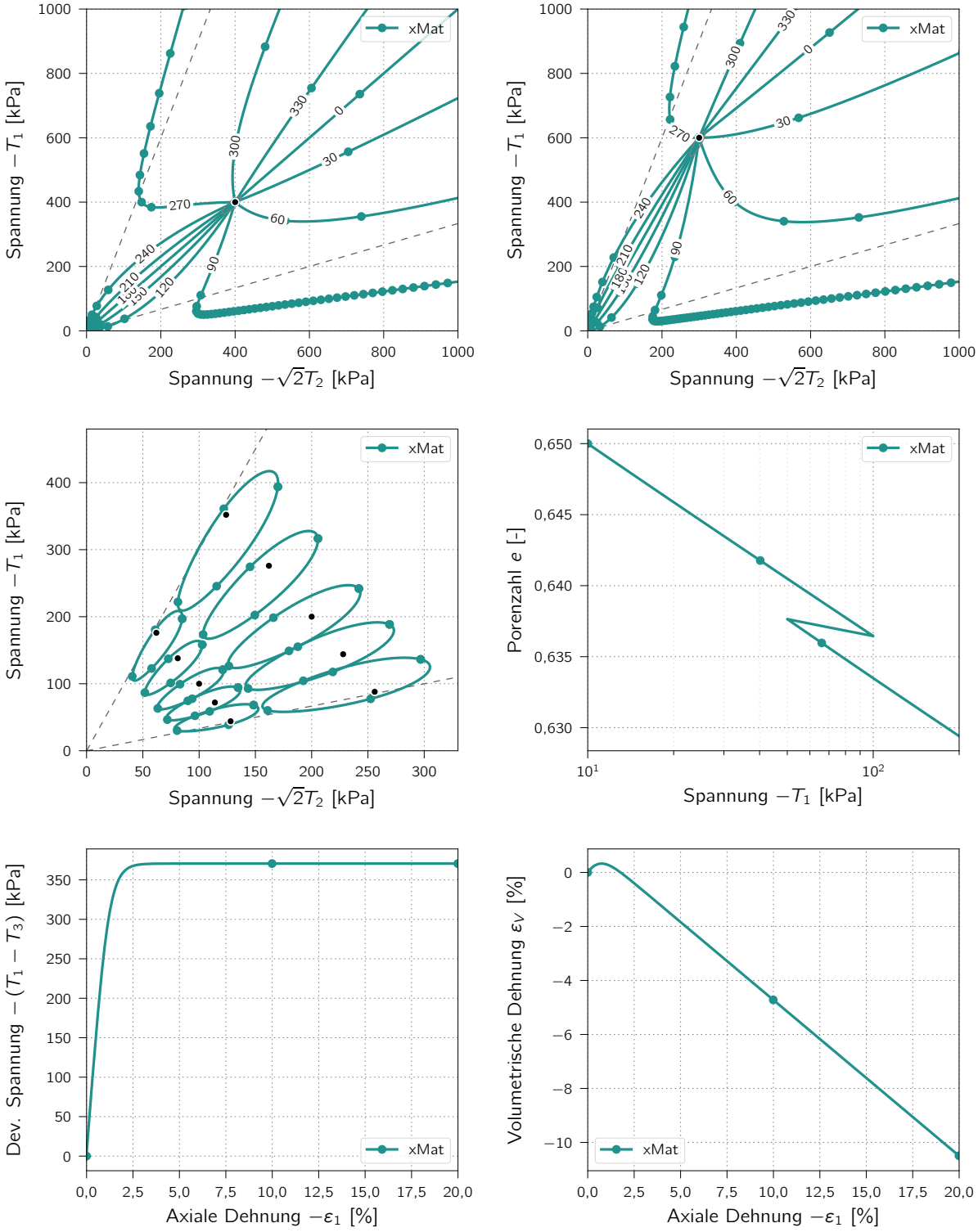


Bild 7.1: Spannungspfade für zwei Spannungszustände (obere Zeile), Antwortumhüllende und Ödometerversuch (mittlere Zeile) sowie Triaxialversuch (untere Zeile) aus dem ElementSimulator. Es wird das Stoffmodell **`Hypo-Wu92`** mit den Parametern für dicht gelagerten Sand aus Tabelle 6.8 auf S. 125 über die `UMAT`-Schnittstelle genutzt. Die Markierungen werden bei Spannungspfaden und Antwortumhüllenden nach jeweils 16 Werten sowie bei Ödometer- und Traxialversuchen nach jeweils 1000 Werten ausgegeben.

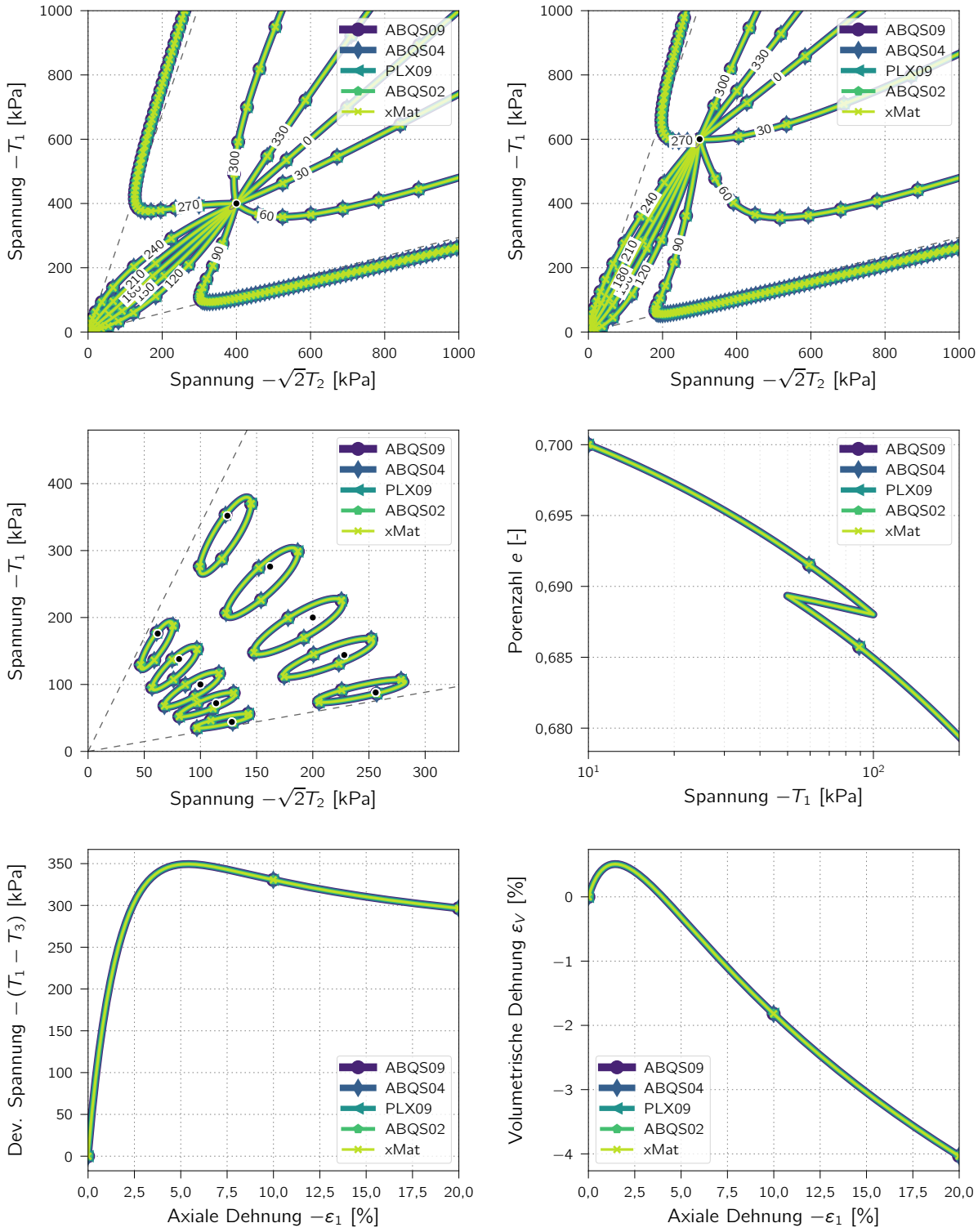


Bild 7.2: Spannungspfade für zwei Spannungszustände (obere Zeile), Antwortumhüllende und Ödometerversuch (mittlere Zeile) sowie Triaxialversuch (untere Zeile) aus dem ElementSimulator. Es wird das Stoffmodell **`Hypo-VW96`** mit den Parametern für Hostun-Sand aus Tabelle 6.9 auf S. 125 **ohne** intergranularer Dehnung über die `UMAT`-Schnittstelle genutzt. Die Markierungen werden bei Spannungspfaden und Antwortumhüllenden nach jeweils 16 Werten sowie bei Ödometer- und Traxialversuchen nach jeweils 1000 Werten ausgegeben.

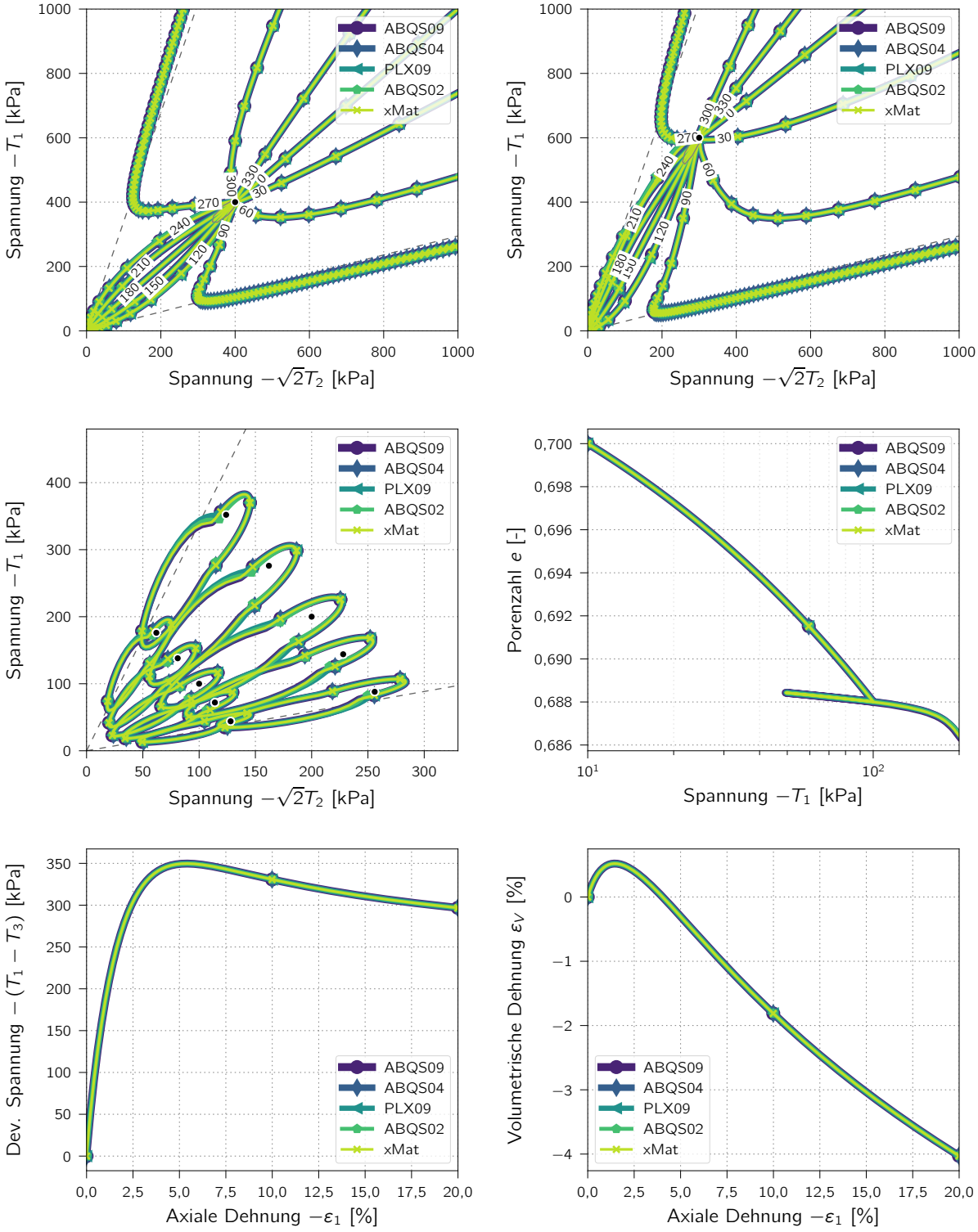


Bild 7.3: Spannungspfade für zwei Spannungszustände (obere Zeile), Antwortumhüllende und Ödometerversuch (mittlere Zeile) sowie Triaxialversuch (untere Zeile) aus dem ElementSimulator. Es wird das Stoffmodell **`Hypo-VW96`** mit den Parametern für Hostun-Sand aus Tabelle 6.9 auf S. 125 **mit** intergranularer Dehnung über die `UMAT`-Schnittstelle genutzt. Die Markierungen werden bei Spannungspfaden und Antwortumhüllenden nach jeweils 16 Werten sowie bei Ödometer- und Traxialversuchen nach jeweils 1000 Werten ausgegeben.

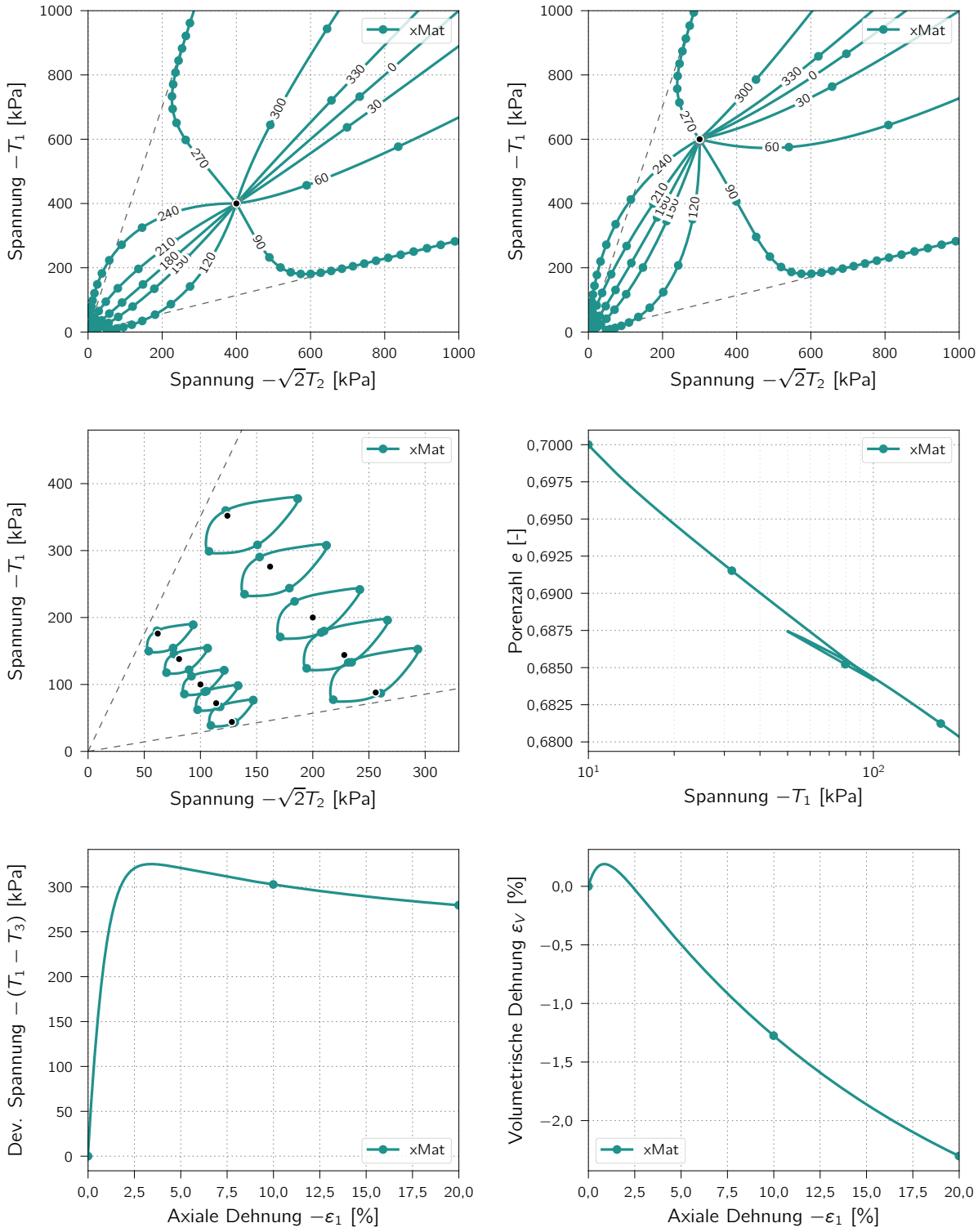


Bild 7.4: Spannungspfade für zwei Spannungszustände (obere Zeile), Antwortumhüllende und Ödometerversuch (mittlere Zeile) sowie Triaxialversuch (untere Zeile) aus dem ElementSimulator. Es wird das Stoffmodell **`Baro-Ko15`** mit den Parametern für Hostun-Sand aus Tabelle 6.10 auf S. 125 über die `UMAT`-Schnittstelle genutzt. Die Markierungen werden bei Spannungspfaden und Antwortumhüllenden nach jeweils 16 Werten sowie bei Ödometer- und Traxialversuchen nach jeweils 1000 Werten ausgegeben.

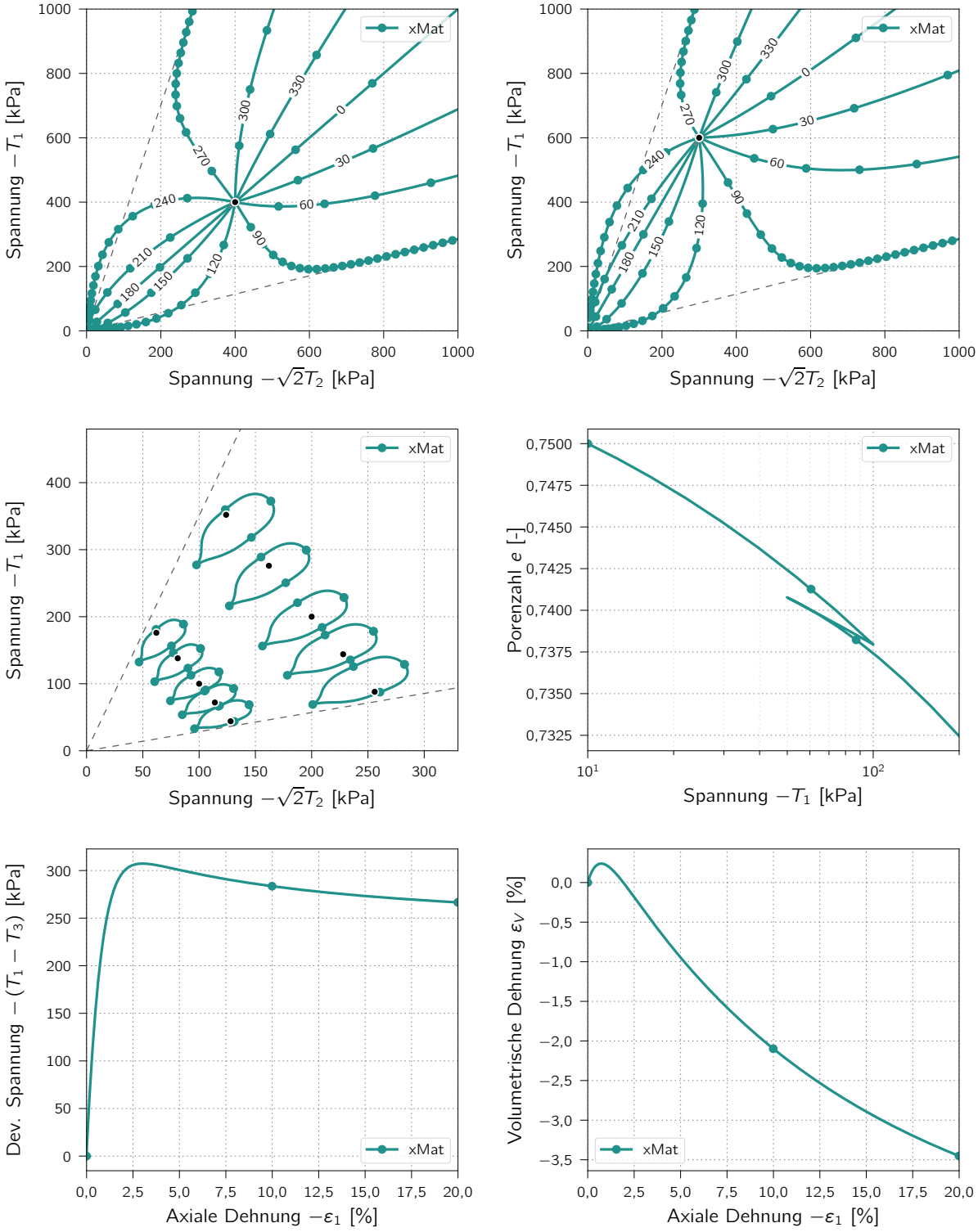


Bild 7.5: Spannungspfade für zwei Spannungszustände (obere Zeile), Antwortumhüllende und Ödometerversuch (mittlere Zeile) sowie Triaxialversuch (untere Zeile) aus dem ElementSimulator. Es wird das Stoffmodell **`Baro-Sc18`** mit den Parametern für Hostun-Sand aus Tabelle 6.11 auf S. 125 über die `UMAT`-Schnittstelle genutzt. Die Markierungen werden bei Spannungspfaden und Antwortumhüllenden nach jeweils 16 Werten sowie bei Ödometer- und Traxialversuchen nach jeweils 1000 Werten ausgegeben.

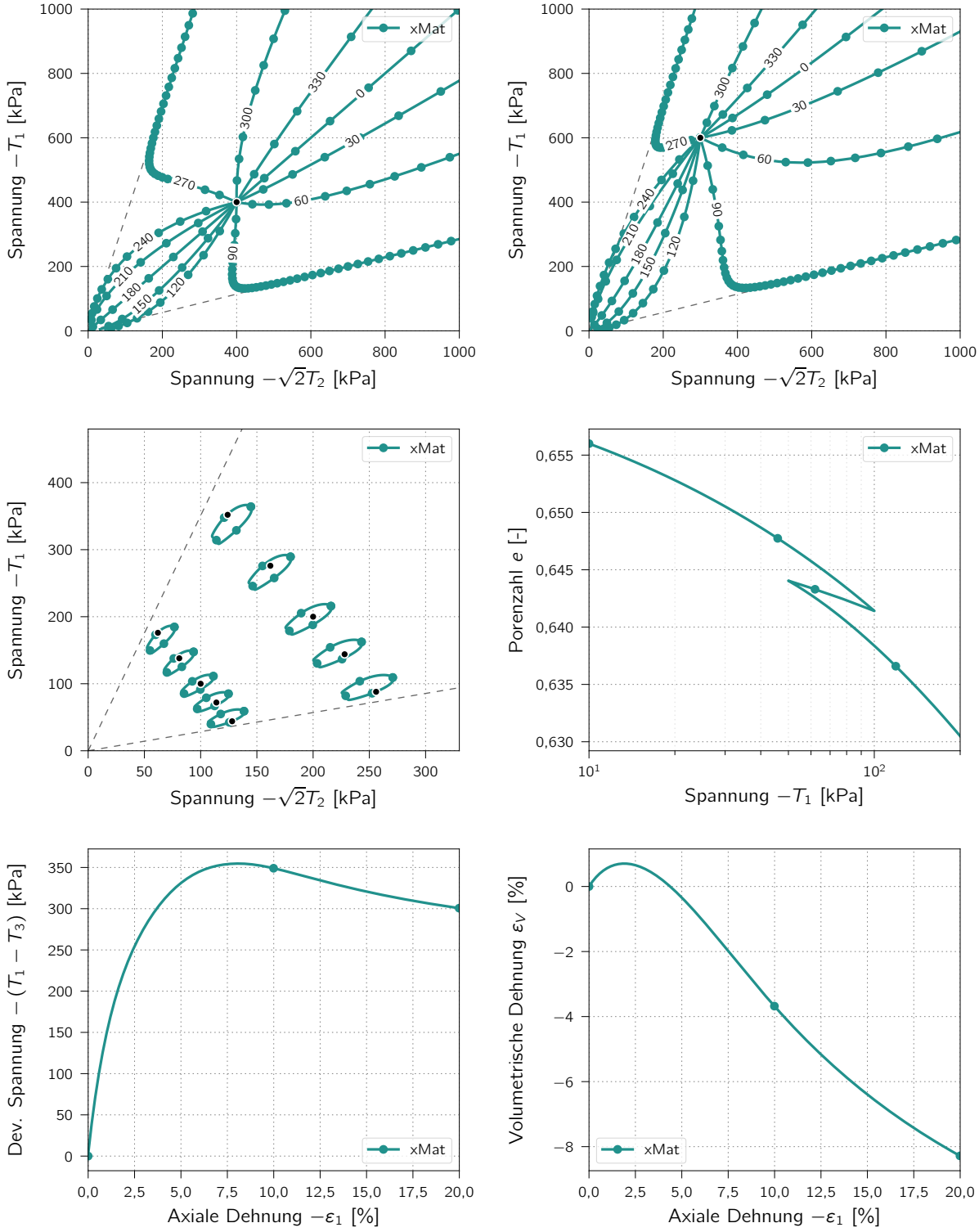


Bild 7.6: Spannungspfade für zwei Spannungszustände (obere Zeile), Antwortumhüllende und Ödometerversuch (mittlere Zeile) sowie Triaxialversuch (untere Zeile) aus dem ElementSimulator. Es wird das Stoffmodell **`Baro-Ko21`** mit den Parametern für Hostun-Sand aus Tabelle 6.12 auf S. 125 über die `UMAT`-Schnittstelle genutzt. Die Markierungen werden bei Spannungspfaden und Antwortumhüllenden nach jeweils 16 Werten sowie bei Ödometer- und Traxialversuchen nach jeweils 1000 Werten ausgegeben.

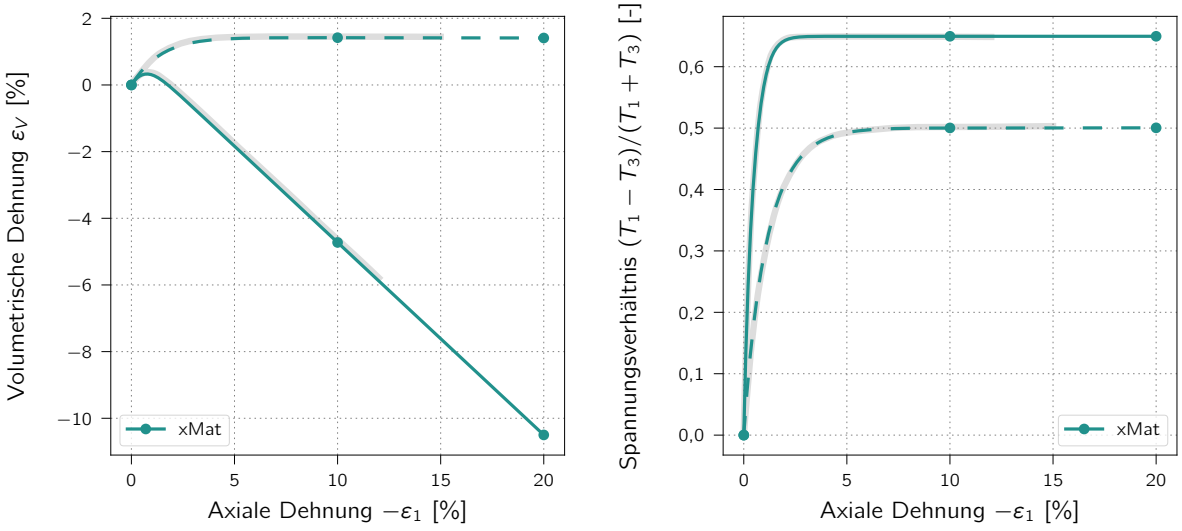
7.2.2 Vergleiche in der Triaxialebene

In diesem Abschnitt wird für die Nachrechnung der Ödometerversuche $D_{\mathrm{ref}} = -1$ verwendet und die Dehnung über den Zeitschritt bestimmt. In der xMat wird mit dem Wert `setting_restrict_initial_substep = .False.` vorgegeben, dass der Skalierungsfaktor t_S dem übergebenen Zeitinkrement dt entspricht. Sofern nicht anders angegeben, ist der Zeitschritt bei Ödometerversuchen $dt = 10^{-5}$.

Die verwendeten Referenzdaten sind von unterschiedlicher Qualität: Im besten Fall sind die Koordinaten der Referenzdaten direkt aus pdf-Dokumenten extrahiert worden. Im schlechtesten Fall ist eine Rekonstruktion aus einem Scan oder einer Fotografie durchgeführt worden. Die Abweichungen zu den in der jeweiligen Quelle veröffentlichten Daten sind zwar klein, führen aber zu Unterschieden zwischen Referenzdaten und Vergleichsrechnungen. Eine Bewertung der Übereinstimmung erfolgt deshalb abhängig vom Einzelfall.

Hypo-Wu92

Die Nachrechnungen mit dem Stoffmodell `Hypo-Wu92` nutzen die Ergebnisse der Triaxial- und Ödometerversuche aus Wu (1992), S. 43 und S. 47 f., als Referenz. Bild 7.7 zeigt den Vergleich der Triaxialversuche, Bild 7.8 den des Ödometerversuchs. Wie in der Quelle werden für dicht und locker gelagerten Sand die Materialparameter aus Tabelle 6.8 auf S. 125 verwendet.



(a) locker **(b)** dicht

Bild 7.7: Triaxialversuche aus Wu (1992), S. 47 f., nachgerechnet mit der xMat. Es wird das Stoffmodell `Hypo-Wu92` über die `UMAT`-Schnittstelle mit den Parametern für locker und dicht gelagerten Karlsruher Mittelsand verwendet (siehe Tabelle 6.8 auf S. 125). Der Zelldruck beträgt 300 kPa und die Anfangsporenzahlen sind 0,53 für die dicht (durchgezogen) und 0,83 für die locker gelagerte Bodenprobe (gestrichelt). Grau hinterlegt sind die Referenzlösungen aus der Quelle.

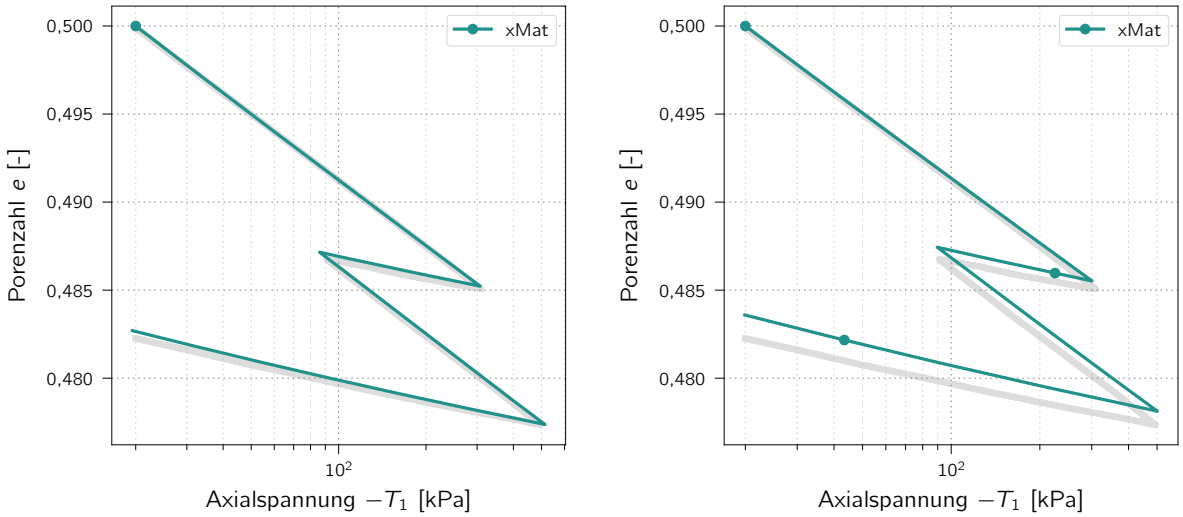


Bild 7.8: Ödometerversuch aus Wu (1992), S. 43, nachgerechnet mit einem Zeitschritt von 10^{-4} (links) und 10^{-5} (rechts). Es wird das Stoffmodell `Hypo-Wu92` über die `UMAT`-Schnittstelle mit den Parametern für dicht gelagerten Karlsruher Mittelsand verwendet (siehe Tabelle 6.8 auf S. 125). Die Anfangsporenzahl ist 0,5 und die fünf Belastungsschritte sind −20, −300, −90, −500 und −20 kPa. Grau hinterlegt ist die Referenzlösung aus der Quelle.

Für die Triaxialversuche kann eine sehr gute Übereinstimmung mit den Referenzdaten gesehen werden. Am Ödometerversuch in Bild 7.8 kann der Einfluss eines zu großen Zeitschritts erkannt werden. Für einen Zeitschritt von 10^{-4} entspricht die Nachrechnung der Referenz. Erst mit einem kleineren Zeitschritt wie 10^{-5} konvergiert die Integration mit dem expliziten Euler-Verfahren auf das ebenfalls in Bild 7.8 dargestellte Ergebnis. Insgesamt zeigen die Nachrechnungen bereits, dass die Ergebnisse der Implementierung von `Hypo-Wu92` in der xMat für die dargestellten Versuche mit den Referenzdaten übereinstimmen.

Hypo-VW96

Bild 7.9 und 7.10 zeigen Nachrechnungen von Triaxial- und Ödometerversuchen aus Wichtmann, Fuentes u. a. (2019), S. 174 f.[3] Als Grundlage für den Vergleich wird das Stoffmodell `Hypo-VW96` nach von Wolffersdorff (1996) mit dem optional zuschaltbaren Konzept der intergranularen Dehnung nach Niemunis und Herle (1997) verwendet. Das heißt, dass keine Anpassung der Schubsteifigkeit oder des Parameters f_d stattfindet.[4] Die Integrationsverfahren sind nicht angepasst worden, sodass alle Benutzerroutinen außer `PLX09` das explizite Euler-Verfahren nutzen. In Benutzerroutine `PLX09` wird die Simpsonregel für die Integration verwendet. Benutzerroutine `ABQS02` führt zwei Halbschritte pro Schritt durch und `ABQS04` nutzt für die Subinkremente eine feste Schrittweite von 10^{-5}.

[3] In dieser Arbeit wird die Porenzahl angegeben, während in Wichtmann, Fuentes u. a. (2019) die relative Lagerungsdichte $D_R = 100\,(e_{\max} - e) \,/\, (e_{\max} - e_{\min})$ verwendet wird.

[4] Deshalb sind zur Reproduktion Anpassungen in der Benutzerroutine `ABQS09` erforderlich.

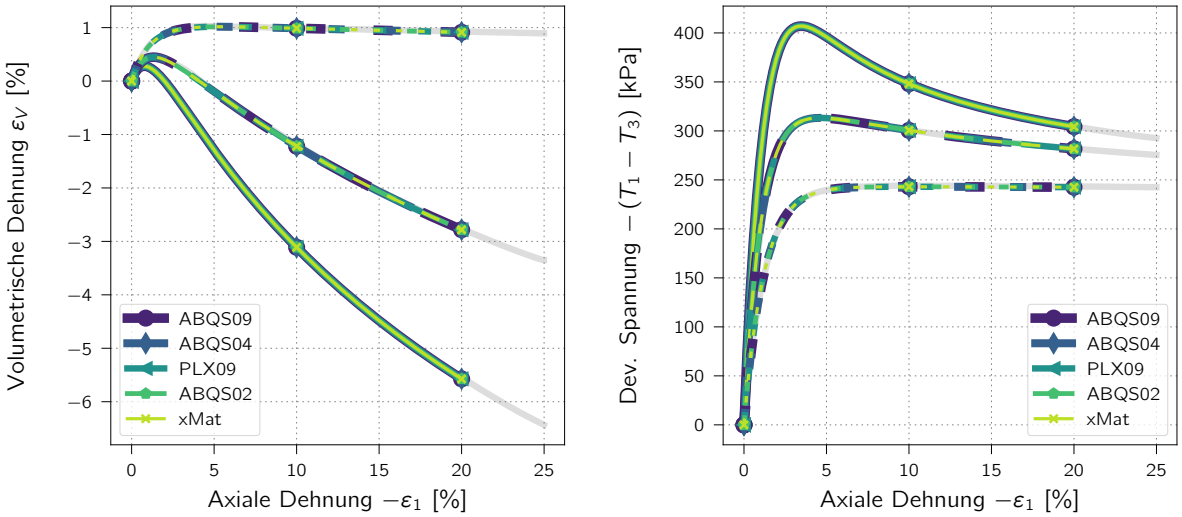


Bild 7.9: Triaxialversuche aus Wichtmann, Fuentes u. a. (2019), S. 175, nachgerechnet mit den verschiedenen Benutzerroutinen. Es wird jeweils das Stoffmodell `Hypo-VW96` über die `UMAT`-Schnittstelle mit den Parametern für Karlsruher Feinsand ohne intergranulare Dehnung verwendet (siehe Tabelle 6.7 auf S. 124). Der Zelldruck beträgt 100 kPa und die Anfangsporenzahlen sind 0,734 (durchgezogen), 0,816 (lang gestrichelt) und 0,975 (kurz gestrichelt). Für die numerische Nachrechnung wird $dt = 10^{-4}$ verwendet. Grau hinterlegt sind die Referenzlösungen aus der Quelle.

In Bild 7.9 kann eine sehr gute Übereinstimmung zwischen den Referenzdaten und allen Benutzerroutinen gesehen werden. Da die Benutzerroutinen mit Unterschieden bei der numerischen Integration annähernd identische Ergebnisse erzeugen, ist das zusätzlich ein Indikator für die Konvergenz der Lösung.

Unterschiede können bspw. in der Nachrechnung in Bild 7.10 gesehen werden. Für ein Zeitinkrement von 10^{-4} stimmen `ABQS09` und die xMat mit der Referenz überein, unterscheiden sich aber vor allem am Ende der Wiederbelastungsphase von den anderen Ergebnissen. Wird das Zeitinkrement auf 10^{-5} verringert, so stimmen die Ergebnisse aller Benutzerroutinen überein, unterscheiden sich aber bei der Wiederbelastung von den Referenzdaten. Das deutet darauf hin, dass für die Berechnung in der Referenz ein zu großer Zeitschritt gewählt worden ist. Im Rahmen der Nachrechnung kann aber gesehen werden, dass diese Versuche gut mit der xMat berechnet werden können.

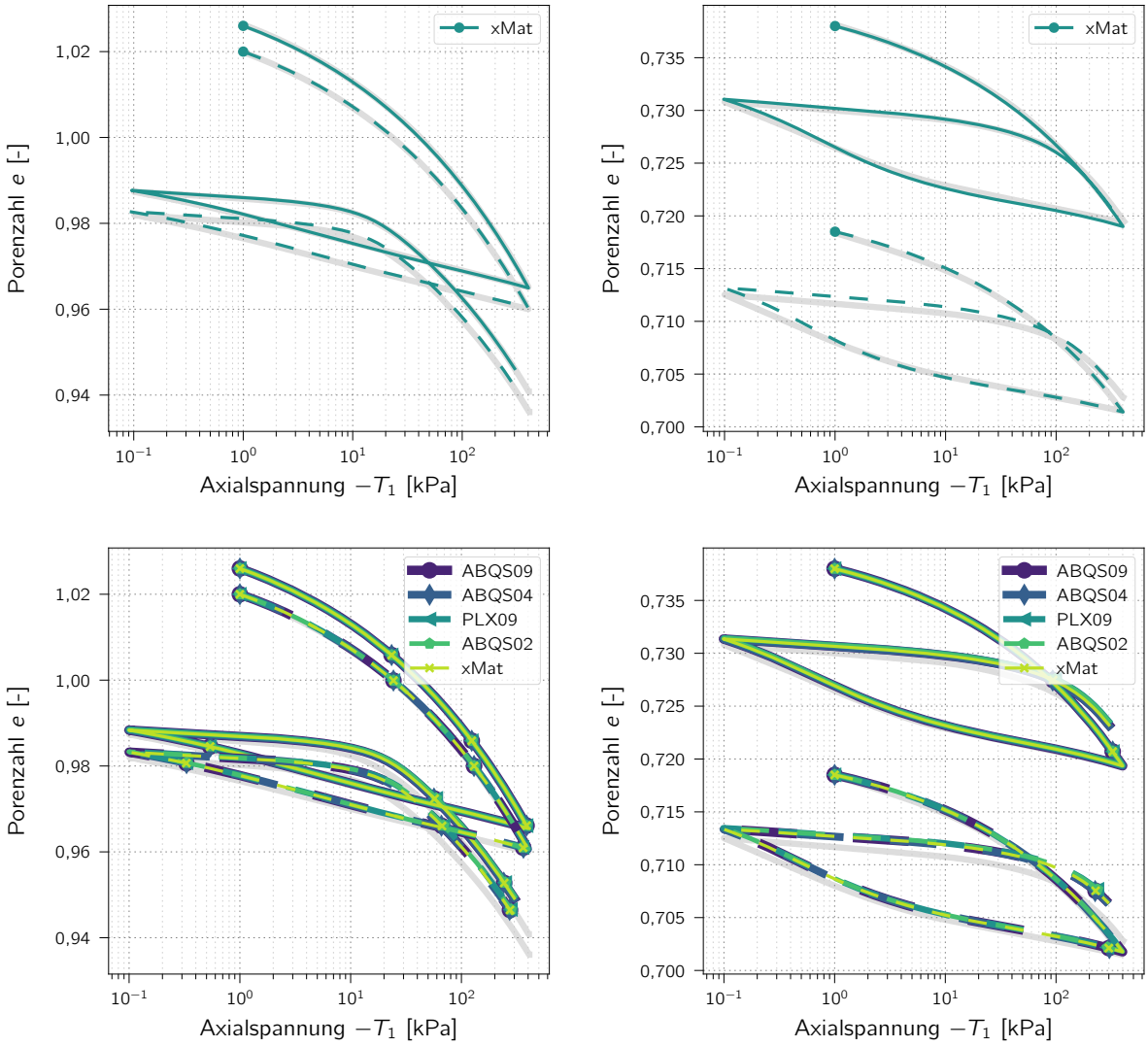


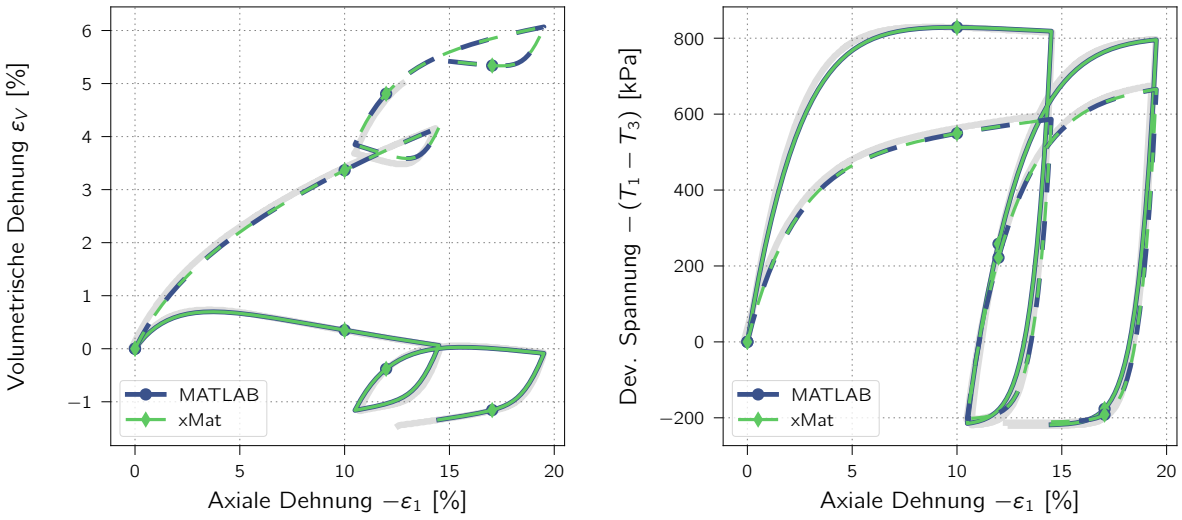
Bild 7.10: Ödometerversuche aus Wichtmann, Fuentes u. a. (2019), S. 174, nachgerechnet mit den verschiedenen Benutzerroutinen. Es wird jeweils das Stoffmodell `Hypo-VW96` über die `UMAT`-Schnittstelle mit den Parametern für Karlsruher Feinsand mit intergranularer Dehnung verwendet (siehe Tabelle 6.7 auf S. 124). Die vier Belastungsschritte sind -1, -400, $-0{,}1$ und $-300\,\mathrm{kPa}$. Die Anfangsporenzahlen 1,026 und 1,02 für die locker gelagerten Proben (links) sowie 0,738 und 0,7185 für die dicht gelagerten Proben (rechts). Für die numerische Nachrechnung wird $dt = 10^{-4}$ (obere Zeile) bzw. $dt = 10^{-5}$ (untere Zeile) verwendet. Grau hinterlegt sind die Referenzlösungen aus der Quelle.

Baro-Ko15

Das Stoffmodell `Baro-Ko15` ist in Kolymbas (2015) definiert und eine Implementierung als MATLAB-Skript findet sich im Anhang von Schranz (2018), S. 172 f. Zusammen mit einer angepassten Version der MATLAB-Dateien für Ödometer- und Triaxialversuche aus Kolymbas (2022), S. 133–135, können neben den Fortran-Programmen auch Vergleiche aus Matlab oder GNU Octave berechnet und verglichen werden.

Zusätzlich zu den mit der xMat berechneten Ergebnissen werden die Berechnungen mit den MATLAB-Skripten für die Triaxialversuche in Bild 7.11 und Ödometerversuche in Bild 7.12 berücksichtigt. Es werden die Materialparameter aus Tabelle 6.10 auf S. 125 genutzt. Da aus der Quelle nicht deutlich wird, welche kritische Porenzahl e_{c0} und welche minimale Porenzahl $e_{\min}$ verwendet wird, sind mit MPO passende Werte ermittelt worden.

Durch die Anpassung von e_{c0} und $e_{\min}$ kann keine klare Aussage über die Ursache der leichten Abweichung der Triaxialversuche in Bild 7.11 getroffen werden. Das etwas steifere Verhalten der dichten Bodenproben in der Quelle spricht für eine abweichende Wahl an Parametern oder einen Unterschied in der Implementierung. Die gute Übereinstimmung des gesamten Verlaufs deutet darauf hin, dass die xMat das Verhalten des Stoffmodells zumindest anhand der Versuche bereits gut darstellen kann.



(a) locker **(b)** dicht

Bild 7.11: Triaxialversuche aus Kolymbas (2015), S. 60, nachgerechnet mit der xMat und einem MATLAB-Skript. Es wird das Stoffmodell `Baro-Ko15` mit den Parametern für Hostun-Sand verwendet (siehe Tabelle 6.10 auf S. 125). Für den Triaxialversuch mit dicht gelagerter Probe (durchgezogen) wird $e_{c0} = 0{,}77$, $e_{\min} = 0{,}24$ und $e_{\text{ini}} = 0{,}63$ genutzt, für den Triaxialversuch mit locker gelagerter Probe (gestrichelt) $e_{c0} = 0{,}83$, $e_{\min} = 0{,}34$ und $e_{\text{ini}} = 0{,}85$. Der Zelldruck beträgt 300 kPa und die Umkehrpunkte für die Belastung sind bei den axialen Dehnungen 14,5 %, 10,5 %, 19,5 % und 14,5 %. Grau hinterlegt sind die Referenzlösungen aus der Quelle. Für die volumetrische Dehnung der locker gelagerten Bodenprobe liegen nur Daten bis $\varepsilon_v \leq 5\,\%$ vor.

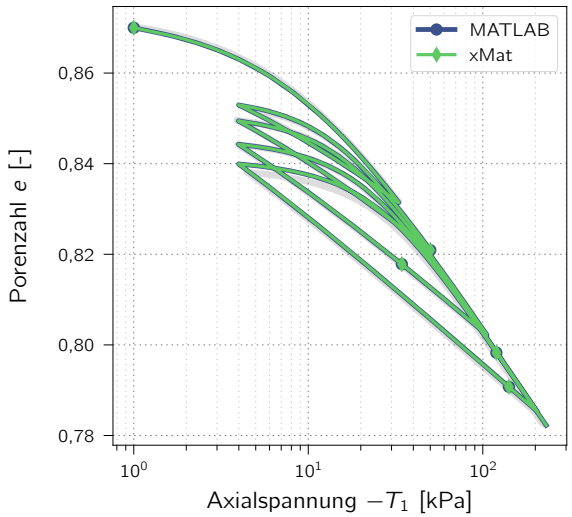


Bild 7.12: Ödometerversuch aus Kolymbas (2015), S. 60, nachgerechnet mit der xMat und einem MATLAB-Skript. Es wird das Stoffmodell `Baro-Ko15` mit den Parametern für Hostun-Sand verwendet (siehe Tabelle 6.10 auf S. 125). Für den Ödometerversuch mit locker gelagerter Bodenprobe werden $e_{c0} = 0{,}9$, $e_{\min} = 0{,}36$ und $e_{\text{ini}} = 0{,}87$ und eine Referenzdehnungsrate von $D_{\text{ref}} = -0{,}5$ genutzt. Die zehn Belastungsschritte sind -1, -33, -4, -40, -4, -105, -4, -205, -4 und $-230\,\text{kPa}$. Grau hinterlegt ist die Referenzlösung aus der Quelle.

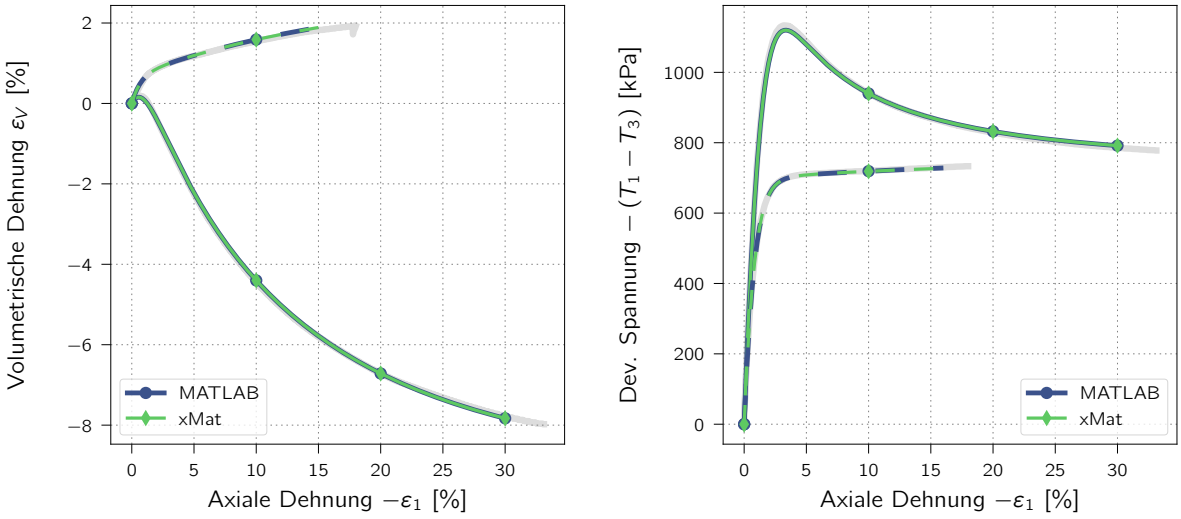
Die gleiche Beobachtung kann am Ödometerversuch gemacht werden. Hier ist mit den ermittelten Werten von e_{c0}, $e_{\min}$ und e_{ini} eine sehr gute Übereinstimmung für vier der fünf Ent- und Wiederbelastungszyklen zu sehen. Wahrscheinlich ist die Ursache der Abweichung die etwas frühere Wiederbelastung im fünften Zyklus der Quelle, sodass der tatsächliche Unterschied als vernachlässigbar einzustufen ist.

Baro-Sc18

Wie bereits für `Baro-Ko15` kommt auch für `Baro-SC18` das MATLAB-Skript für eine zweite Berechnung zum Einsatz. Als Referenz dienen die Triaxial- und Ödometerversuche aus Schranz (2018), S. 70–73. Die Ergebnisse der Nachrechnungen sind in Bild 7.13 und 7.14 dargestellt.

Es sind die Materialparameter aus Tabelle 6.11 auf S. 125 verwendet worden. Während alle Daten für die Triaxialversuche vorliegen, fehlen in der Quelle die Anfangsporenzahlen für beide Ödometerversuche. Die beste Übereinstimmung ist erneut mit MPO abgeschätzt worden.

Der Vergleich der nachgerechneten Triaxial- und Ödometerversuche zeigt eine sehr gute Übereinstimmung mit den Referenzdaten. Für die Ödometerversuche liegt das auch daran, dass die Anfangsporenzahl aufgrund der besten Übereinstimmung mit den Referenzdaten gewählt worden ist. Dennoch kann gesehen werden, dass die xMat das Verhalten des Stoffmodells gut abbilden kann.



(a) locker **(b)** dicht

Bild 7.13: Triaxialversuche aus Schranz (2018), S. 70 f., nachgerechnet mit der xMat und einem MATLAB-Skript. Es wird das Stoffmodell `Baro-Sc18` mit den Parametern für Hostun-Sand verwendet (siehe Tabelle 6.11 auf S. 125). Für den Triaxialversuch mit dicht gelagerter Bodenprobe (durchgezogen) wird $e_{\text{ini}} = 0{,}66$ verwendet, für den Triaxialversuch mit locker gelagerter Bodenprobe (gestrichelt) $e_{\text{ini}} = 0{,}87$. Der Zelldruck beträgt 300 kPa und die max. Dehnung ist 16 % für die locker und 30 % für die dicht gelagerte Bodenprobe. Grau hinterlegt sind die Referenzlösungen aus der Quelle.

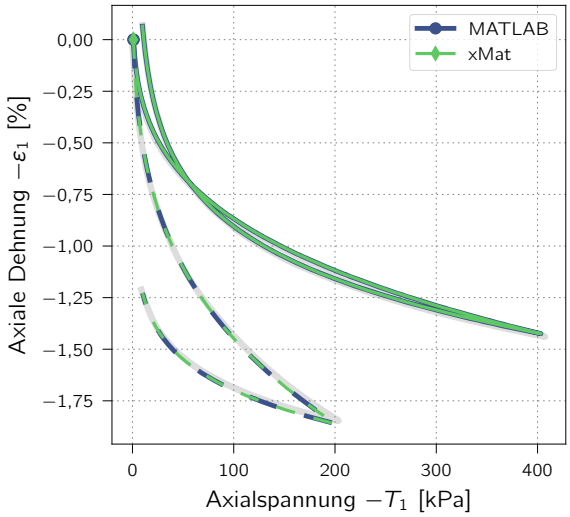
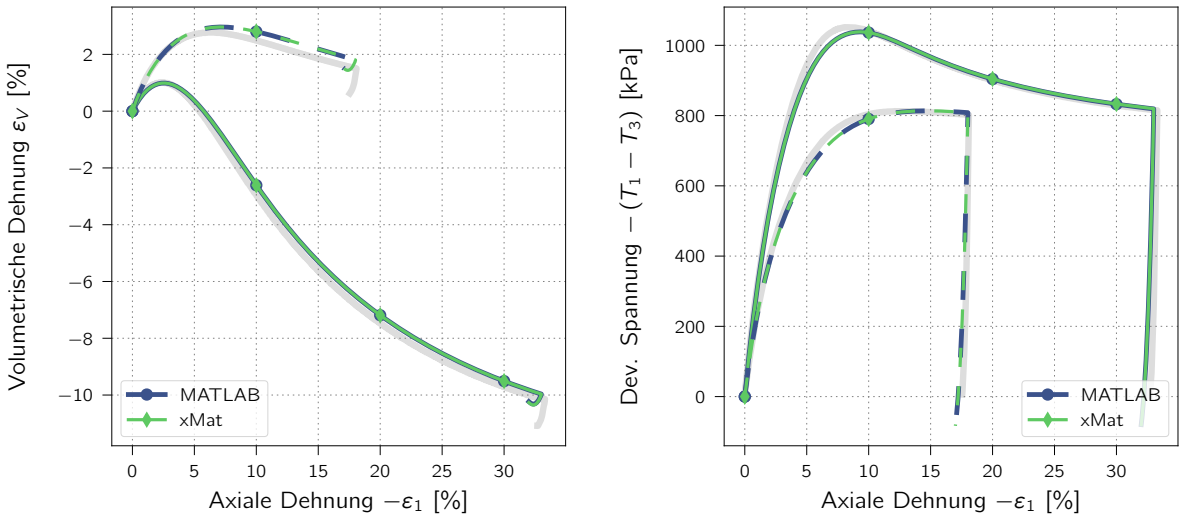


Bild 7.14: Ödometerversuche aus Schranz (2018), S. 73, nachgerechnet mit der xMat und einem MATLAB-Skript. Es wird das Stoffmodell `Baro-Sc18` mit den Parametern für Hostun-Sand verwendet (siehe Tabelle 6.11 auf S. 125), wobei für den Ödometerversuch mit dicht gelagerter Probe (durchgezogen) $e_{\text{ini}} = 0{,}67$ und für den Ödometerversuch mit locker gelagerter Probe (gestrichelt) $e_{\text{ini}} = 0{,}88$ genutzt wird. Die drei Belastungsschritte sind -1, -400 und -10 kPa für die dichte Bodenprobe und -1, -200 und -10 kPa für die lockere Bodenprobe. Grau hinterlegt sind die Referenzlösungen aus der Quelle.

Baro-Ko21

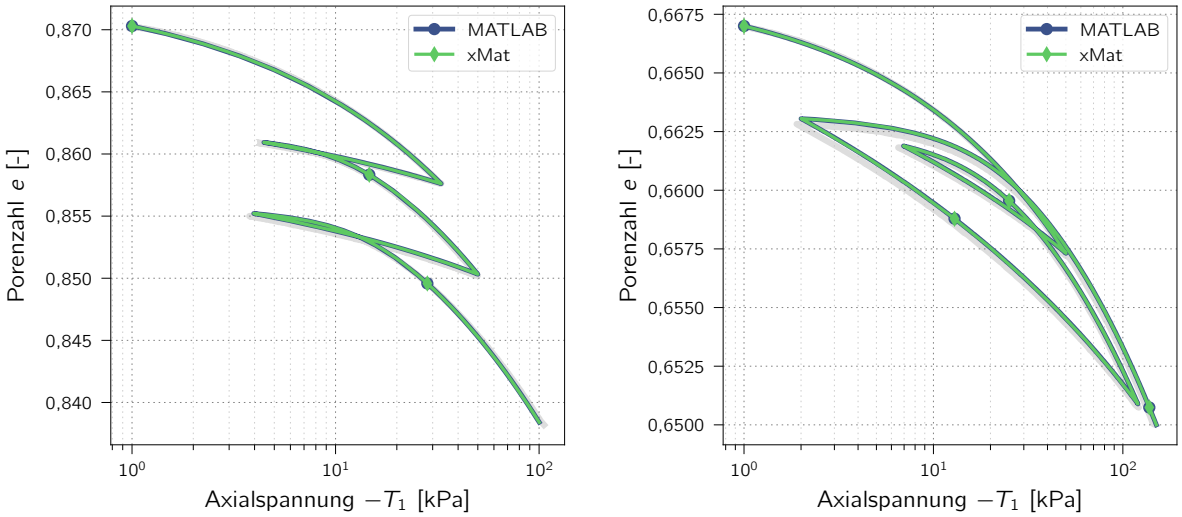
Das Stoffmodell wird in Kolymbas (2021) definiert und ist dort sowie in Kolymbas (2022), S. 127 f., als MATLAB-Implementierung abgedruckt. Wie bei den vorangegangenen barodetischen Stoffmodellen werden die bereits erwähnten Skripte für Ödometer- und Triaxialversuche aus Kolymbas (2022), S. 133–135, für zusätzliche Nachrechnungen aus GNU Octave genutzt. Als Referenzdaten für das Stoffmodell `Baro-Ko21` werden die Triaxial- und Ödometerversuche aus Kolymbas (2021), S. 211 f. und S. 217 f. genutzt. Es werden die dort eingesetzten Materialparameter für Hostun-Sand aus Tabelle 6.12 auf S. 125 verwendet. Die Ergebnisse sind in Bild 7.15 und 7.16 dargestellt.

Wie bereits in Gleichung (2.12) auf Seite 36 zu sehen ist und auch bei der Implementierung in Abschnitt 5.3 angesprochen wird, setzt sich die Integration der kritischen Porenzahl e_c aus zwei Teilen zusammen. Der erste Teil wird vom Zeitschritt kontrolliert, der zweite von der Dehnungsrate. Während in der xMat immer beide Teile betrachtet werden, wird der zweite Summand von Gleichung (2.12) in den Referenzlösungen der Triaxialversuche in Bild 7.15 nicht berücksichtigt. Ohne den zweiten Term wird eine etwas kleinere volumetrische Dehnung bestimmt und der Peak-Zustand etwas früher erreicht.



(a) locker **(b)** dicht

Bild 7.15: Triaxialversuche aus Kolymbas (2021), S. 5, nachgerechnet mit der xMat und einem MATLAB-Skript. Es wird das Stoffmodell `Baro-Ko21` mit den Parametern für Hostun-Sand verwendet (siehe Tabelle 6.12 auf S. 125) und $e_{c0} = 0{,}54$. Für den Triaxialversuch mit dicht gelagerter Bodenprobe (durchgezogen) wird $e_{\text{ini}} = 0{,}656$ verwendet, für den Triaxialversuch mit locker gelagerter Bodenprobe (gestrichelt) $e_{\text{ini}} = 0{,}869$. Der Zelldruck beträgt 300 kPa und der Umkehrpunkt für die Belastung ist bei einer axialen Dehnung von 33 % bzw. 17 %. Grau hinterlegt sind die Referenzlösungen aus der Quelle.



(a) locker **(b)** dicht

Bild 7.16: Ödometerversuche aus Kolymbas (2021), S. 5 f., nachgerechnet mit der xMat und einem MATLAB-Skript. Es wird das Stoffmodell `Baro-Ko21` mit den Parametern für Hostun-Sand verwendet (siehe Tabelle 6.12 auf S. 125), wobei für den Ödometerversuch mit locker gelagerter Probe (links) $e_{\text{ini}} = 0{,}8703$ und $e_{c0} = 0{,}75$ genutzt wird und für den Ödometerversuch mit dicht gelagerter Probe (rechts) $e_{\text{ini}} = 0{,}70$ und $e_{c0} = 0{,}667$. Mit locker gelagerter Bodenprobe werden die sechs Belastungsschritte -1, -50, -7, -120, -2 und -150 kPa berechnet. Mit dicht gelagerter Bodenprobe werden die Belastungsschritte -1, -33, $-4{,}5$, -50, -4 und -100 kPa verwendet. Grau hinterlegt sind die Referenzlösungen aus der Quelle.

Bei den Ödometerversuchen in Bild 7.16 zeigt vor allem die Berechnung mit der dicht gelagerten Bodenprobe kleine Abweichungen zur Referenzlösung. Der Unterschied ergibt sich aus dem gewählten Zeitinkrement, da die Referenzdaten mit $dt = 10^{-4}$ berechnet worden sind.

7.2.3 Weitere Spannungszustände

Im Folgenden wird die Antwort des Stoffmodells `Hypo-VW96` für eine Reihe an Spannungs- und Dehnungszuständen betrachtet. Dazu werden zuerst verschiedene Referenzen für Spannungen und Dehnungsraten definiert (siehe Tabelle 7.1). Die Werte der Varianten sind beliebig mit den folgenden Einschränkungen gewählt worden:

- Es werden drei Werte ungleich null für jede Variante genutzt,
- die Werte und Kombinationen davon sollen gültige Spannungs- und Dehnungszustände für das Stoffmodell erzeugen und
- als vierte Variante wird der Nullzustand verwendet.

Aus den vorgegebenen Varianten werden verschiedene Kombinationen zusammengestellt. Für einen Spannungs- und Dehnungszustand gilt

$$\mathbf{T} = \begin{pmatrix} T_{R1} & T_{X1} & T_{X3} \\ T_{X1} & T_{R2} & T_{X2} \\ T_{X3} & T_{X2} & T_{R3} \end{pmatrix} \quad \text{und} \quad \mathbf{D} = \begin{pmatrix} D_{R1} & D_{X1} & D_{X3} \\ D_{X1} & D_{R2} & D_{X2} \\ D_{X3} & D_{X2} & D_{R3} \end{pmatrix} .$$

Aus den elf in Tabelle 7.2 dargestellten Kombinationen wird jeweils eine für $\boldsymbol{T}_R$, $\boldsymbol{T}_X$, $\boldsymbol{D}_R$ und $\boldsymbol{D}_X$ ausgewählt. So ergibt sich z. B. für die Indizes 2, 7, 9 und 4 aus beiden Tabellen

$$\mathbf{T} = \begin{pmatrix} -100 & -4 & -3 \\ -4 & -80 & -5 \\ -3 & -5 & -80 \end{pmatrix} \quad \text{und} \quad \mathbf{D} = \begin{pmatrix} -10^{-5} & 5 \cdot 10^{-7} & -10^{-6} \\ 5 \cdot 10^{-7} & -5 \cdot 10^{-6} & 5 \cdot 10^{-7} \\ -10^{-6} & 5 \cdot 10^{-7} & 5 \cdot 10^{-6} \end{pmatrix} .$$

Tabelle 7.1: Die folgenden 16 Referenzwerte für Spannungen und Dehnungsraten bilden die Grundlage für alle Variationen. Aus den einzelnen Varianten werden Kombinationen und letztendlich Spannungs- und Dehnungszustände definiert (siehe Tabelle 7.2).

Variablen	Variante 1	Variante 2	Variante 3	Variante 4
Spannungen $\boldsymbol{T}_R$	-100	-90	-80	0
Spannungen $\boldsymbol{T}_X$	-5	-4	-3	0
Dehnungsraten $\boldsymbol{D}_R$	-10^{-5}	$-5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	0
Dehnungsraten $\boldsymbol{D}_X$	-10^{-6}	$-5 \cdot 10^{-7}$	$5 \cdot 10^{-7}$	0

Tabelle 7.2: Die Variation einer Versuchsreihe. Zum einen wird ein Zustand mit drei gleichen Werten betrachtet (`iso`) und der Nullzustand (`null`). Außerdem drei Triaxialzustände, bei denen sich nur zwei Werte unterscheiden und die sechs Permutationen aller drei Werte.

Index	Bezeichnung	Kombination der Varianten	Beispiel für Spannung $\boldsymbol{T}_R$
1	`iso`	1, 1, 1	$(-100, -100, -100)$
2	`triax_u`	1, 3, 3	$(-100, -80, -80)$
3	`triax_v`	3, 1, 3	$(-80, -100, -80)$
4	`triax_w`	3, 3, 1	$(-80, -80 - 100)$
5	`perm1`	3, 2, 1	$(-80, -90, -100)$
6	`perm2`	1, 3, 2	$(-100, -80, -90)$
7	`perm3`	2, 1, 3	$(-90, -100, -80)$
8	`perm4`	3, 1, 2	$(-80, -100, -90)$
9	`perm5`	1, 2, 3	$(-100, -90, -80)$
10	`perm6`	2, 3, 1	$(-90, -80, -100)$
11	`null`	4, 4, 4	$(0, 0, 0)$

Insgesamt sind jeweils elf Werte für die vier Größen $\boldsymbol{T}_R$, $\boldsymbol{T}_X$, $\boldsymbol{D}_R$ und $\boldsymbol{D}_X$ definiert, was zu 14641 Kombinationen führt. Für eine kompakte Darstellung aller Ergebnisse werden die Kombinationen mit einem Laufindex angegeben. Die Reihenfolge entspricht der Verschachtelung an Schleifen im Hauptprogramm. Am langsamsten variieren die Dehnungsraten $\boldsymbol{D}_R$ auf der Hauptdiagonalen und somit entsprechen die ersten 1331 Versuche Index 1 bzw. `iso`. In der zweiten Schleife variieren Spannungen $\boldsymbol{T}_R$ auf der Hauptdiagonalen und wechseln somit in Blöcken von 121 Versuchen. Als Nächstes variieren die Dehnungsraten $\boldsymbol{D}_X$ auf der Nebendiagonalen und am schnellsten variieren die Spannungen $\boldsymbol{T}_X$ auf der Nebendiagonalen (siehe Bild 7.17).

Die Stoffmodellantwort wird für alle Variationen mit $dt = t_S = 10^{-5}$ berechnet. Wie bereits im vorherigen Abschnitt beschrieben worden ist, müssen Zeitschritt und Dehnungsrate passend gewählt sein, da sonst Unterschiede durch Integrationsfehler gemessen werden.

Spannungen

In Bild 7.18 ist die Spannung T_{11} für die unterschiedlichen Variationen über dem Laufindex dargestellt. Es kann gesehen werden, dass die Ergebnisse für alle Benutzerroutinen identisch sind. Das Gleiche trifft auch für die anderen Komponenten der berechneten Spannung zu, die hier nicht dargestellt sind.

Jacobi-Matrix

Bild 7.19 zeigt die Einflüsse in der Komponente J_{1111} der Jacobi-Matrizen. Mit intergranularer Dehnung sind die Ergebnisse aller Benutzerroutinen annähernd gleich. Da die intergranulare Dehnung mit null initialisiert wird, ergeben sich für die Jacobi-Matrix die Einträge des Tensors $m_R\mathcal{L}$.

Ohne das Konzept der intergranularen Dehnung kann eine deutliche Trennung in zwei Gruppen erkannt werden: Für `PLX09` und die xMat wird der Tensor $\mathcal{L}$ als Jacobi-Matrix genutzt. Mit dem verwendeten $m_R = 5$ entsprechen die Werte einem Fünftel der Ergebnisse mit intergranularer Dehnung. Da das Stoffmodell und somit auch die Jacobi-Matrix durch einen zweiten Term definiert wird, entspricht das nicht dem Wert, der sich durch die anderen Benutzerroutinen bzw. die numerische Abschätzung in der xMat ergibt. Es können auch kleine Abweichungen in den Ergebnissen der zweiten Gruppe gesehen werden, vor allem zwischen Laufindex 6655 und 7986 sowie zwischen 10648 und 11979. Hier weicht `ABQ09` von den Werten der anderen drei Benutzerroutinen `ABQS02`, `ABQS04` und xMat (num) ab.

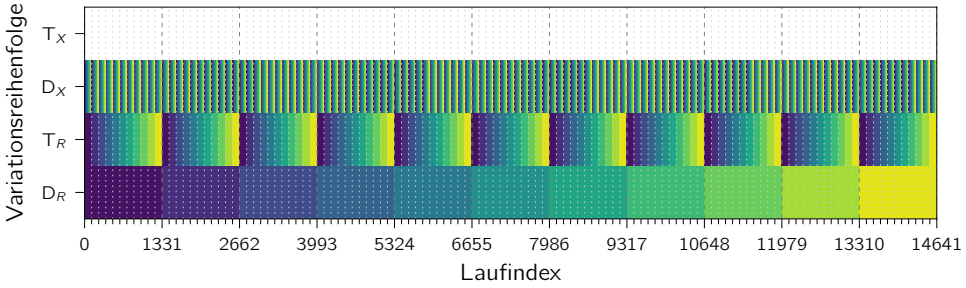


Bild 7.17: Farbkodierung der einzelnen Variationen. $\boldsymbol{D}_R$ in der äußersten Schleife variiert am langsamsten und ist in Blöcken von 1331 Versuchen identisch. $\boldsymbol{T}_R$ variiert alle 121 Versuche, $\boldsymbol{D}_X$ alle 11 Versuche und $\boldsymbol{T}_X$ in der innersten Schleife in jedem Versuch.

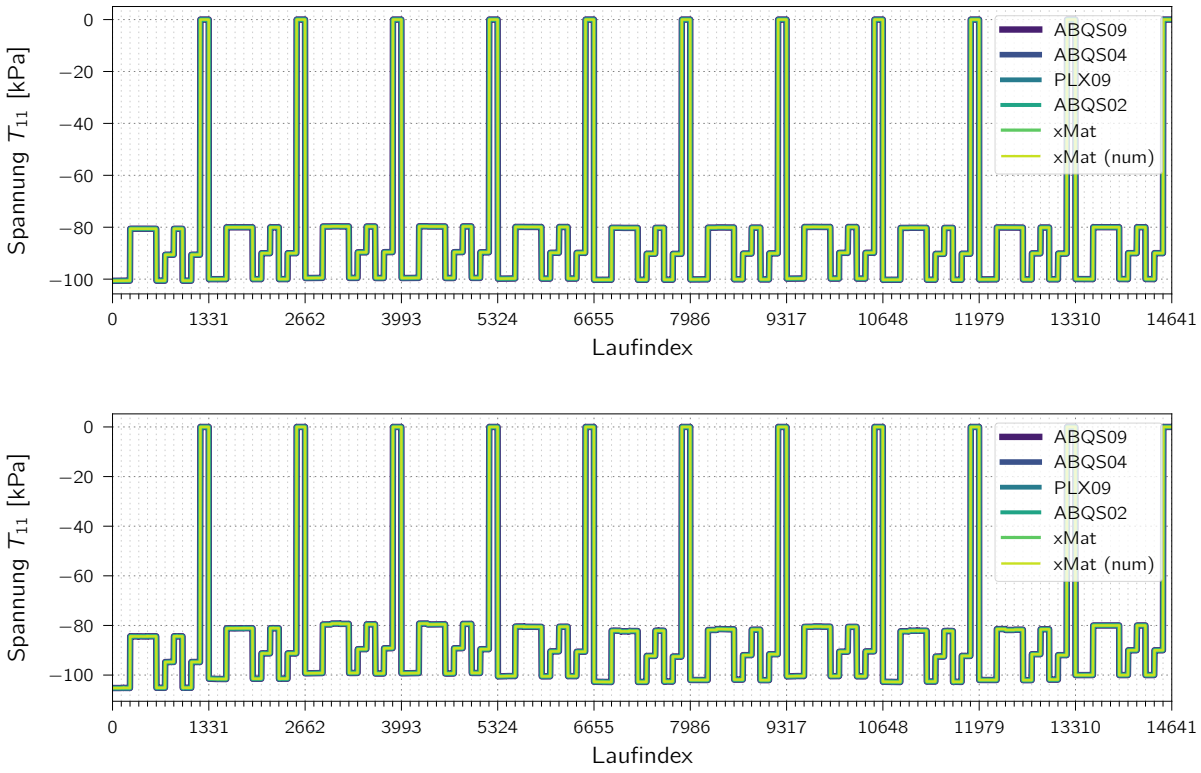


Bild 7.18: Spannungskomponente T_{11} der betrachteten Benutzerroutinen für verschiedene Spannungszustände und Dehnungsänderungen mit dem Stoffmodell `Hypo-VW96`. In der oberen Zeile ohne und in der unteren Zeile mit dem Konzept der intergranularen Dehnungen.

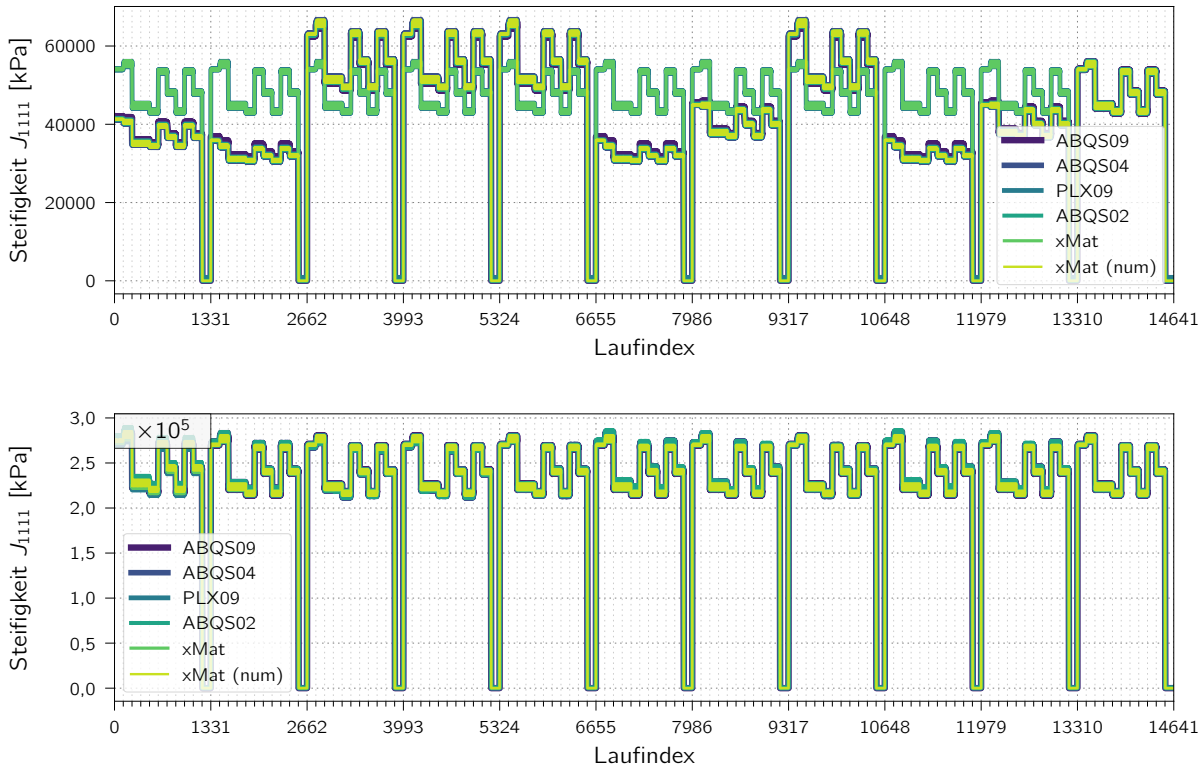


Bild 7.19: Komponente J_{1111} der Jacobi-Matrix der betrachteten Benutzerroutinen für verschiedene Spannungszustände und Dehnungsänderungen mit dem Stoffmodell `Hypo-VW96`. In der oberen Zeile ohne und in der unteren Zeile mit dem Konzept der intergranularen Dehnungen.

Zugspannungen

Stoffmodelle für nichtbindige Böden sind im Allgemeinen nur für Druckspannungen definiert. Wie bereits in Abschnitt 4.2.2 erwähnt, können im Rahmen von FE-Analysen Spannungszustände mit verschwindenden Druckspannungen oder Zugspannungen auftreten. Die Benutzerroutinen `ABQS04`, `ABQS09` und die xMat stellen dafür ein elastisches Ersatzmodell bereit und halten dabei die Porenzahl des letzten gültigen Druckspannungszustands konstant (siehe Bild 7.20). In `ABQS02` wird als Ersatzmodell die Kompressionsbeziehung aus Gleichung (3.2) genutzt. Da die xMat dieselben Werte wie `ABQS04` nutzt, um auf das Ersatzmodell umzuschalten, sind die beiden Kurven identisch.

In Bild 7.20 fällt auf, dass `ABQS09` bei einer Eingangsspannung $-\operatorname{tr}(\mathbf{T}) < 10^{-6}$ abrupt vom hypoplastischen Stoffmodell auf das Ersatzmodell umschaltet. Im Gegensatz dazu ist für `ABQS02`, `ABQS04` und mit der xMat ein Übergangsbereich zu sehen. Die Größe des Übergangsbereichs ergibt sich aus der Anzahl der durchgeführten Subinkremente. In einem Subinkrement kommt entweder das hypoplastische Modell oder das Ersatzmodell zum Einsatz, aber die berechnete Modellantwort setzt sich aus allen durchgeführten Subinkrementen zusammen.

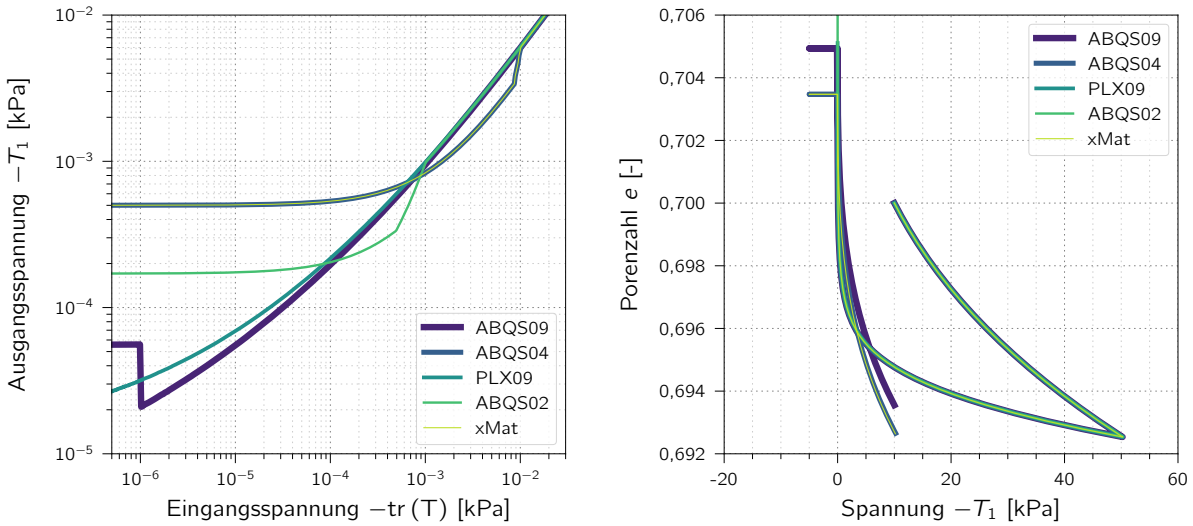


Bild 7.20: Verhalten der Benutzerroutinen mit dem Stoffmodell `Hypo-VW96` für kleine Druckspannungen und Zugspannungen. Es werden die Parameter für Hostun-Sand ohne intergranulare Dehnung aus Tabelle 6.9 auf S. 125 mit einer Anfangsporenzahl $e_{\mathrm{ini}} = 0{,}7$ verwendet. Die Referenzdehnungsrate ist $D_{\mathrm{ref}} = -5$ und der Zeitschritt 10^{-5}. Links: Alle Benutzerroutinen außer `PLX09` stellen ein Ersatzmodell bereit, das für kleine Druckspannungen aktiviert wird. Bei `ABQS09` beträgt der Grenzwert -10^{-6} kPa, bei `ABQS02` -10^{-3} kPa und bei `ABQS04` sowie der xMat -10^{-2} kPa. Rechts: Die Benutzerroutinen mit einem Ersatzmodell für Zugspannungen halten die Porenzahl im Zugspannungsbereich konstant. Bei `PLX09` ändert sich für eine verschwindende Spannung die Porenzahl nicht mehr, während bei `ABQS02` ein weiteres Anwachsen der Porenzahl für verschwindende Spannungen beobachtet werden kann.

Interne Repräsentation

Wie in Abschnitt 5.2 beschrieben, sind in der Entwicklungsumgebung der xMat die drei internen Repräsentationen `TENS3333`, `MAT99` und `MAT66` auswählbar. Zur Überprüfung werden ebenfalls verschiedene Spannungs- und Dehnungszustände mit dem Stoffmodell `Hypo-VW96` betrachtet. Es werden die Materialparameter für Hostun-Sand und optional zuschaltbare intergranulare Dehnungen verwendet. Die Anfangsporenzahl ist jeweils $e_{\mathrm{ini}} = 0{,}8$.

Die Ergebnisse werden wiederum mit einem Laufindex angegeben, der der Berechnungsreihenfolge entspricht. Die äußere Schleife variiert die Spannungszustände $\mathbf{T}_A$, $\mathbf{T}_B$ und $\mathbf{T}_C$ mit

$$\mathbf{T}_A = \begin{pmatrix} -250 & \cdot & \cdot \\ \cdot & -250 & \cdot \\ \cdot & \cdot & -250 \end{pmatrix} \quad \text{und} \quad \mathbf{T}_B = \begin{pmatrix} -100 & 50 & \cdot \\ 50 & -250 & 10 \\ \cdot & 10 & -300 \end{pmatrix}$$

sowie $\mathbf{T}_C = \mathbf{0}$. In einer weiteren Schleife wird das Konzept der intergranularen Dehnung entweder aktiviert oder deaktiviert. Es folgt die Variation der Dehnungsraten $\mathbf{D}_A$, $\mathbf{D}_B$ und $\mathbf{D}_C$ mit

$$\mathbf{D}_A = \begin{pmatrix} -10^{-4} & \cdot & \cdot \\ \cdot & 3 \cdot 10^{-5} & \cdot \\ \cdot & \cdot & 3 \cdot 10^{-5} \end{pmatrix} \quad \text{und} \quad \mathbf{D}_B = \begin{pmatrix} -2 \cdot 10^{-4} & -2 \cdot 10^{-5} & \cdot \\ -2 \cdot 10^{-5} & 8 \cdot 10^{-5} & -10^{-5} \\ \cdot & -10^{-5} & 4 \cdot 10^{-5} \end{pmatrix}$$

sowie $\mathbf{D}_C = \mathbf{0}$. Am schnellsten variiert die Länge des Zeitinkrements

$$dt_A = 2 \quad , \qquad dt_B = 0{,}1 \quad \text{und} \qquad dt_C = 0 \quad .$$

Die berechnete Spannungsantwort und die ermittelten Statusvariablen sind unabhängig von der gewählten Repräsentation identisch. Mit der analytischen Berechnung der Jacobi-Matrix sind auch deren Einträge identisch. Nur bei der numerischen Berechnung der Jacobi-Matrix kann es zu Abweichungen kommen. Die Komponenten der berechneten Jacobi-Matrix sind in den Bildern 7.21 und 7.22 zu sehen. Dabei kann das Folgende festgestellt werden:

- Grundsätzlich sind jeweils zwei Gruppen an Kurven zu sehen. So liegen alle Repräsentationen mit der numerischen Bestimmung der Jacobi-Matrix übereinander und ebenso alle Repräsentationen mit der analytischen Berechnung.
- Die Abweichungen zwischen analytischer und numerischer Berechnung sind vor allem für einen Spannungszustand ungleich null, einer Dehnungsänderung mit Nebendiagonalelementen ungleich null und aktivierter intergranularer Dehnung zu sehen (Laufindex 10 bis 15 und 28 bis 33).
- Für die Laufindizes 4 und 22, also einem Spannungszustand ungleich null, einer Dehnungsänderung mit Nebendiagonalelementen ungleich null, ohne das Konzept der intergranularen Dehnung und für große Zeitinkremente, können ebenfalls Abweichungen erkannt werden.
- Außerdem sind für Laufindizes 10 bis 15 und 28 bis 33 in den Plots der Komponenten `(1, 4)` bis `(3, 6)` kleine Unterschiede bei der numerischen Berechnung zwischen `MAT66` und den anderen beiden Repräsentationen sichtbar. Die Abweichungen ergeben sich durch die unterschiedliche Störung der einzelnen Komponenten in der Funktion `Perturbate()`. Jedes Paar symmetrischer Nebendiagonalelemente in der `MAT66`-Repräsentation wird gleichzeitig mit dem halben Wert gestört und in den anderen Repräsentationen einzeln mit dem angegebenen Wert, was zu einem Unterschied zwischen den Repräsentationen führen kann. Die Unterschiede sind auch in Block `(4, 4)` bis `(6, 6)` vorhanden, aber bei der dargestellten Skalierung nicht sichtbar.

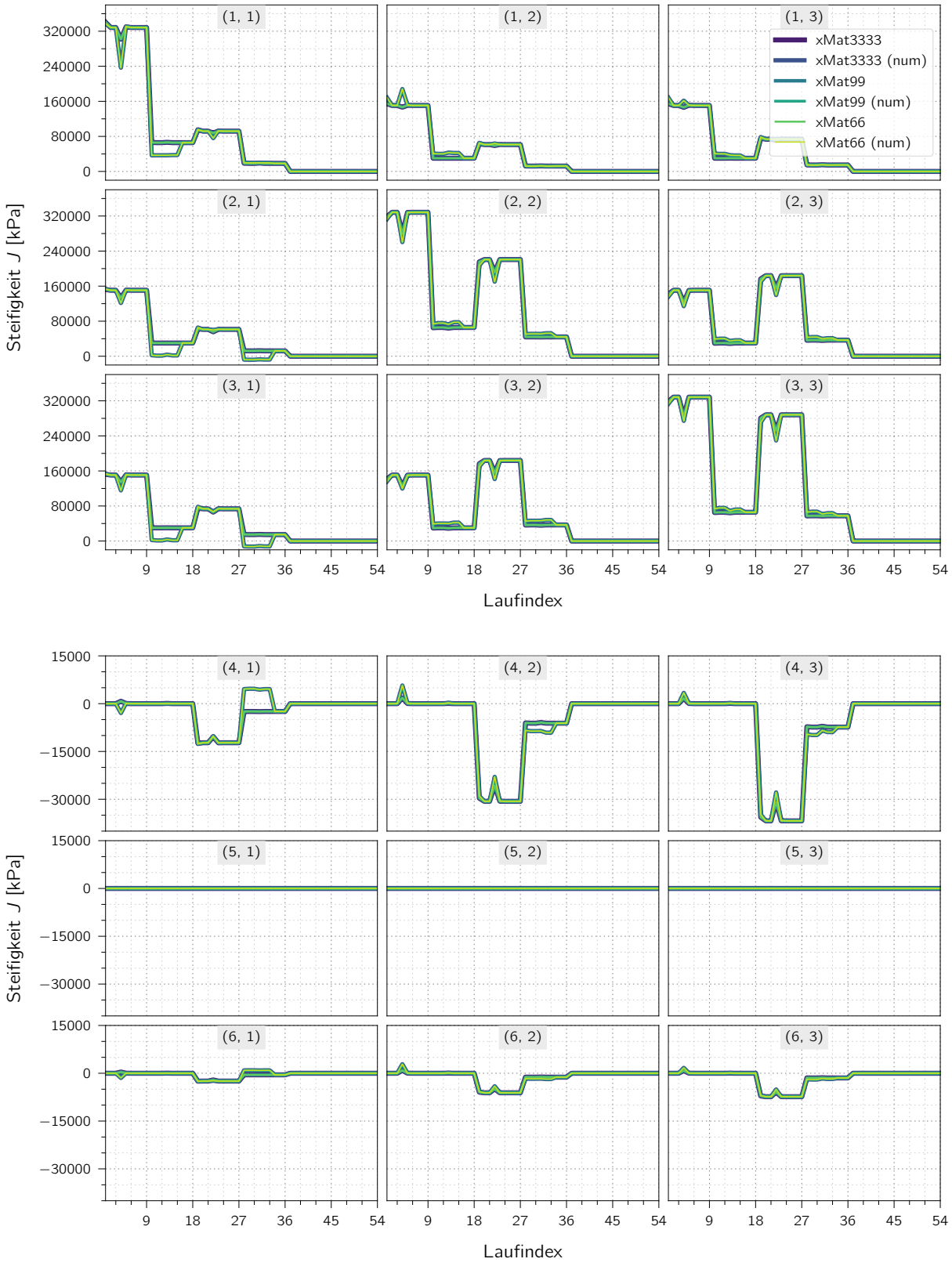


Bild 7.21: Komponenten der Jacobi-Matrix für die internen Repräsentationen `MAT66`, `MAT99` und `TENS3333` mit den vorgegebenen Spannungszuständen und Dehnungsänderungen für das Stoffmodell `Hypo-VW96`.

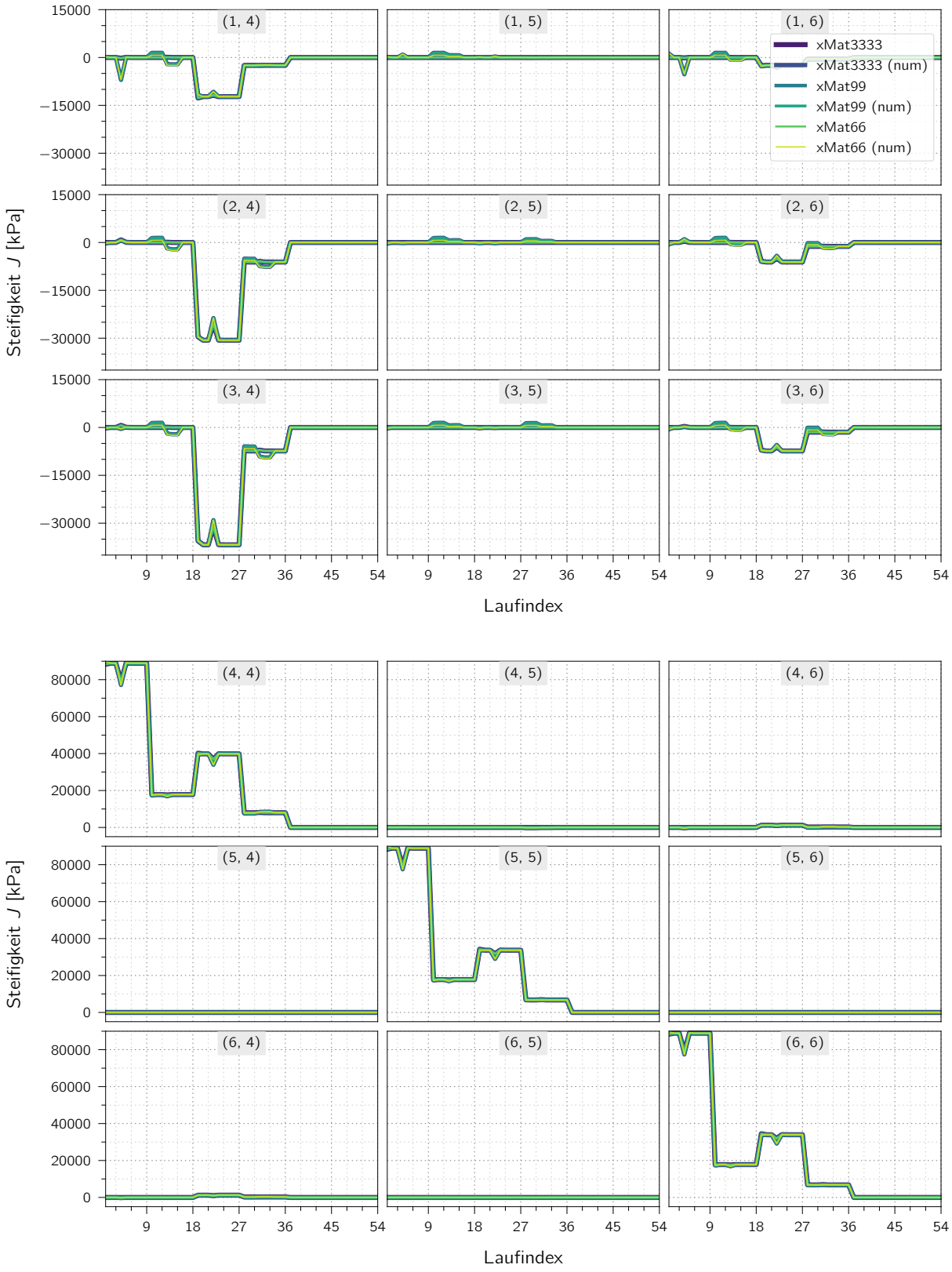


Bild 7.22: Fortsetzung der Komponenten der Jacobi-Matrix für die internen Repräsentationen `MAT66`, `MAT99` und `TENS3333` mit den vorgegebenen Spannungszuständen und Dehnungsänderungen für das Stoffmodell `Hypo-VW96`.

7.3 Untersuchung der Schnittstellen

7.3.1 Überprüfung der Funktionalität

Im vorherigen Abschnitt ist das Verhalten der Stoffmodelle über die UMAT-Schnittstelle betrachtet worden. An dieser Stelle wird überprüft, ob für Aufrufe über alle Schnittstellen das gleiche Ergebnis berechnet wird, sofern die Eingabewerte denselben Zustand beschreiben. Dabei ist zu beachten, dass sich Aufruf und Verarbeitung zwischen den Schnittstellen unterscheiden können. So erwartet z. B. die UMAT-Schnittstelle eine andere Reihenfolge der Komponenten von Spannungs- und Dehnungstensor und bei der USER_MOD-Schnittstelle sind zwei Aufrufe erforderlich, um den neuen Spannungszustand und die Jacobi-Matrix zu erhalten.

Für die Untersuchung wird das Stoffmodell Hypo-VW96 mit den Parametern für Karlsruher Feinsand aus Tabelle 6.7 auf S. 124 über alle Schnittstellen genutzt. Außerdem werden die folgenden Einstellungen verwendet:

$$\mathbf{T} = \begin{pmatrix} -100 & 50 & 0 \\ 50 & -250 & 10 \\ 0 & 10 & -250 \end{pmatrix} \quad , \qquad \mathbf{D} = \begin{pmatrix} -5 \cdot 10^{-5} & 0 & 0 \\ 0 & 1{,}5 \cdot 10^{-5} & 0 \\ 0 & 0 & 1{,}5 \cdot 10^{-5} \end{pmatrix} \quad ,$$

$e_{\mathrm{ini}} = 0{,}8 \quad$ und $\qquad dt = t_S = 10^{-3} \quad .$

Für diesen Test werden die in Tabelle 7.3 dargestellten Programmiersprachen und Schnittstellen unter Linux und Windows untersucht. Die Schnittstellen UMAT, VUMAT und USER_MOD werden für diesen Test nicht aus Abaqus bzw. Plaxis, sondern, wie dargestellt, aus Fortran aufgerufen. Die C-Schnittstellen xmat_console_CInter und Dot_Values_CInter werden nur von C direkt genutzt und sind an Python, GNU Octave und Simulink über zusätzliche Hilfsfunktionen angebunden. Die verwendeten Skripte, Programme und Anweisungen zum Kompilieren und Ausführen sind in Anhang F zusammengestellt.

Tabelle 7.3: Verwendete Programme und Programmiersprachen zur Untersuchung der Schnittstellen der xMat.

Programm bzw. Sprache	Schnittstelle (mit Int.)	Schnittstelle (ohne Int.)
Fortran	xmat_console	Dot_Values
Fortran	UMAT	
Fortran	VUMAT	
Fortran	USER_MOD	
C	xmat_console_CInter	Dot_Values_CInter
Python	xmat_console_CInter	Dot_Values_CInter
GNU Octave	xmat_console_CInter	Dot_Values_CInter
MATLAB	mexFunction	
Simulink		Dot_Values_CInter

Damit die Ergebnisse zwischen den Schnittstellen mit und ohne Integration vergleichbar sind, muss die Integration aus einem einzigen Zeitschritt mit dem expliziten Euler-Verfahren (`Eul-Exp`) bestehen. Das kann mit einer passenden Konfiguration aus dem übergebenen Zeitinkrement *dt*, dem Faktor `setting_initial_substep_scale` und den Dehnungsraten gesteuert oder mit `setting_restrict_initial_substep=.False.` erzwungen werden. Beim Vergleich muss die Sonderstellung der Statusvariablen beachtet werden, die als `direct_variables` gekennzeichnet sind (siehe Abschnitt 4.2.2).[5]

Die Ergebnisse aller Aufrufe werden mit einer Variation von `printf("%16.7f", var)` auf sieben Nachkommastellen ausgegeben. Die Rückgabewerte sind `dotstress`, `dotstate` und `jacobian` für die Schnittstellen ohne Integration und `newstress`, `newstate` und `jacobian` für die Schnittstellen mit Integration. Die Rückgabe der Jacobi-Matrix erfolgt von allen Schnittstellen außer `VUMAT`.

Für alle Schnittstellen ohne Integration und alle Schnittstellen mit Integration wird jeweils das gleiche Ergebnis und die gleiche Jacobi-Matrix berechnet (siehe Code 7.1). Das gilt auch für weitere, zufällig gewählte Spannungs- und Dehnungszustände. Auch wenn die Vermutung nahe liegt, kann allein aus den Übereinstimmungen keine Allgemeingültigkeit abgeleitet werden. Es wird aber aufgezeigt, dass grundsätzlich gleiche Ergebnisse über alle Schnittstellen erwartet werden können.

Code 7.1: Von der xMat berechnete Werte für einen Aufruf mit den vorab beschriebenen Eingangswerten.

```
dotstress =  -2114.0591677    2161.3875544    2161.3875544    -272.7098160     -54.5419632       0.0000000
dotstate =      -0.0360000      -0.0175960       0.0052788       0.0052788       0.0000000       0.0000000
 0.0000000       0.0000000       0.0000000       0.0000000  151364.3507748       0.0000000       0.0000000
 0.0000000       0.0000000       0.0000000       0.0000000       0.0000000       0.0000000       0.0000000

newstress =   -102.1140592    -247.8386124    -247.8386124      49.7272902       9.9454580       0.0000000
newstate =       0.7999640      -0.0000176       0.0000053       0.0000053       0.0000000       0.0000000
 0.0000000       0.0000000       0.0000000       0.0000000  151364.3507748       0.0000000       0.0000000
 0.0000000       0.0000000       0.0000000       0.0000000       0.0000000       0.0000000       0.0000000

jacobian =
( 1, :)      75006.1274656   54541.5735204   54541.5735204  -10908.3144520   -2181.6628904       0.0000000
( 2, :)      54541.4755398  189543.4624293  136353.9596655  -27270.7861299   -5454.1572260       0.0000000
( 3, :)      54541.4755398  136353.9596655  189543.4624293  -27270.7861299   -5454.1572260       0.0000000
( 4, :)     -10908.2813220  -27270.7960689  -27270.7960689   32048.9086079    1090.8314452       0.0000000
( 5, :)      -2181.6562644   -5454.1592138   -5454.1592138    1090.8314452   26812.9176710       0.0000000
( 6, :)          0.0000000       0.0000000       0.0000000       0.0000000       0.0000000   26594.7513819
```

[5] Das trifft auf die elfte Statusvariable des Stoffmodells `Hypo-VW96` zu.

7.3.2 Einfache FE-Simulationen

Im nächsten Schritt werden Ödometer- und Triaxialversuche in Abaqus/Standard und Abaqus/Explizit mit der xMat nachgerechnet. Für die Untersuchungen mit Abaqus kommt erneut der ElementSimulator zum Einsatz, der Python-Skripte zur Modellerstellung und -auswertung bereitstellt. Zusätzlich zu den in Abschnitt 7.1 beschriebenen Einstellungen werden die Bodenkörper in den Untersuchungen nach den Vorgaben von Abschnitt 2.2.2 als axialsymmetrische Elemente modelliert (Elementtyp *CAX4* bzw. *CAX4R*).

Die Ödometerversuche nutzen ein Element mit einer radialen Länge von 35 mm und einer Höhe von 20 mm. Die vertikalen Spannungen sind überall gleich und die Anfangsspannungen in radialer Richtung entsprechen überall der Hälfte der vertikalen Anfangsspannung, da keine Spannungsänderung über die Höhe des Elements berücksichtigt wird. Das Bodenelement ist an der unteren Kante fixiert und die obere Kante kann sich nur vertikal verformen (siehe Bild 7.23).

Für die Triaxialversuche hat das Bodenelement eine radiale Länge von 25 mm und eine Höhe von 125 mm. Hier sind alle Spannungen im Bodenkörper gleich, da ein hydrostatischer Anfangszustand ohne Spannungsänderung über die Höhe des Elements definiert wird. Das Bodenelement ist an der unteren Kante vertikal und an der Mittelachse radial fixiert. Zu jedem Zeitpunkt ist die obere Kante parallel zur unteren Kante.

In den FE-Simulationen wird das Stoffmodell `Hypo-VW96` mit den Materialparametern für Hostun-Sand verwendet (siehe Tabelle 6.9 auf S. 125). Dabei wird eine Anfangsporenzahl $e_{\text{ini}} = 0{,}7$ und eine voll ausgebildete intergranulare Dehnung in Belastungsrichtung vorgegeben. Bei den Triaxialversuchen wird das Abscheren in einer Belastungsphase mit einer Dauer von 0,1 s durchgeführt. Dabei wird das Element gleichmäßig um 20 % gestaucht und

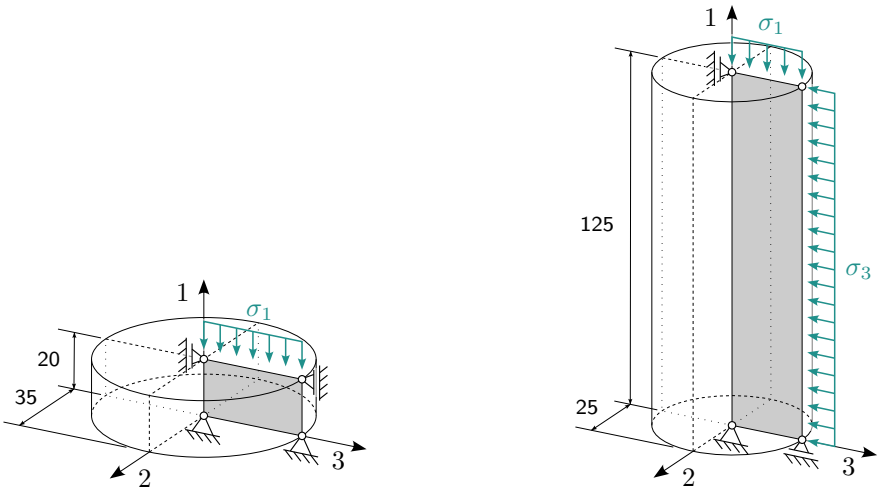


Bild 7.23: Randbedingungen für Ödometerversuch (link) und Triaxialversuch (rechts) in den FE-Simulationen. Für den Triaxialversuch gilt im Anfangszustand $\sigma_1 = \sigma_3$. Die obere Kante, auf der die Drucklast σ_1 aufgebracht wird, ist in beiden Versuchen immer parallel zur unteren Kante.

Verschiebungen, Spannungen und Porenzahlen alle $5 \cdot 10^{-5}$ s aufgezeichnet (2000 Datenpunkte). Die Ödometerversuche werden in Abaqus/Standard mit einer Belastungsphase pro Belastungsrichtung modelliert, während in Abaqus/Explizit eine einzige Belastungsphase mit wechselnder Belastung (über `TabularAmplitude`) genutzt wird.

Zusätzlich werden Berechnungen in Fortran und im SoilTest-Modul von Plaxis durchgeführt, das neben vordefinierten Elementversuchen die Definition allgemeiner Belastungszustände erlaubt (siehe auch Anhang F). In Plaxis werden die Triaxialversuche mit einer Belastungsphase mit 0,1 s gerechnet und 2000 Datenpunkte ausgegeben und die Ödometerversuche mit drei Belastungsphasen von je 1 s und mit je 500 Datenpunkten (siehe Bild 7.24).

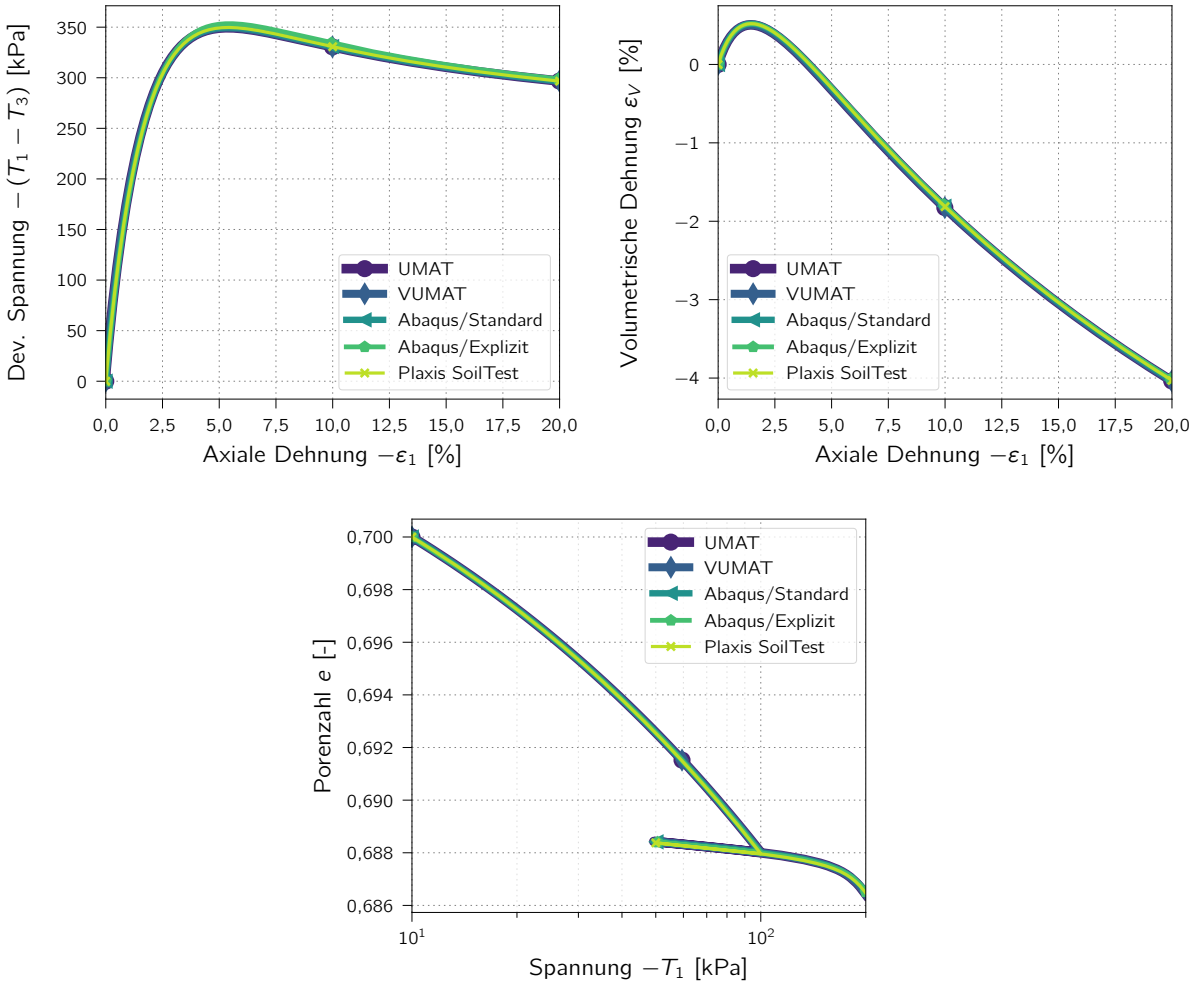


Bild 7.24: Triaxialversuch (obere Zeile) und Ödometerversuch (untere Zeile) aus dem ElementSimulator, ergänzt mit den Daten des SoilTest-Moduls von Plaxis. Zum einen werden die idealisierten Elementversuche über die `UMAT`- und `VUMAT`-Schnittstelle in Fortran nachgerechnet, zum anderen über die jeweilige Schnittstelle in Abaqus/Standard, Abaqus/Explizit und in Plaxis. Es wird das Stoffmodell `Hypo-VW96` mit den Parametern für Hostun-Sand aus Tabelle 6.9 auf S. 125 mit intergranularer Dehnung verwendet (siehe auch Bild 7.3). Die Markierungen werden nach jeweils 1000 Werten ausgegeben.

Auf den ersten Blick sehen die dargestellten Ergebnisse für alle Schnittstellen gleich aus. Bei näherer Betrachtung können minimale Abweichungen der Ergebnisse von Abaqus/Explizit, Abaqus/Standard und Plaxis SoilTest zu den Fortran-Aufrufen über die UMAT- und VUMAT-Schnittstelle gesehen werden. Nach der vorherigen Überprüfung der Schnittstellen und Stoffmodelle kann davon ausgegangen werden, dass diese kleinen Unterschiede entweder aus den Einstellungen des numerischen Modells und der FE-Analyse stammen, oder noch nicht berücksichtigten Eigenschaften der übergebenen Werte in den Schnittstellen.

In jedem Fall sind weitere Untersuchungen an diesen einfachen Modellen notwendig, um die Ursache zu lokalisieren und geeignete Anpassungen durchzuführen. Anschließend sind weitere Untersuchungen für beliebige Spannungs- und Dehnungszustände an einem einfachen Modell empfehlenswert, bevor größere und komplexere FE-Simulationen mit der xMat durchgeführt werden.

7.4 Untersuchung der Integrationsverfahren

In der xMat sind verschiedene Integrationsverfahren implementiert, die im Folgenden etwas näher betrachtet werden. Bereits in Abschnitt 2.4 wird beschrieben, dass mit steigender Ordnung eines expliziten Runge-Kutta-Verfahrens die Anzahl erforderlicher Funktionsauswertungen in einem einzigen Integrationsschritt steigt. Tabelle 7.4 listet die Anzahl der Auswertungen eines Stoffmodells auf, wenn nur ein einziger Integrationsschritt durchgeführt wird. Wie dargestellt kann die Anzahl benötigter Funktionsauswertungen für die gleiche Bereitstellungsart der Jacobi-Matrix bis zum Faktor 7 voneinander abweichen. Bei unterschiedlicher Bereitstellung der Jacobi-Matrix kann sich die Anzahl der Auswertungen bis zu einem Faktor 49 unterscheiden.

Gleichzeitig ist bei einem Integrationsverfahren höherer Ordnung grundsätzlich eine höhere Konvergenzgeschwindigkeit möglich. Die tatsächliche Geschwindigkeit zur Berechnung des nächsten Zeitschritts hängt von vielen Faktoren ab, vor allem der Differentialgleichung selbst, des Zeitschritts und des anfänglichen Zeitinkrements, sowie dem zulässigen Integrationsfehler und den Parametern für die Skalierung der nächsten Schrittweite. Mit den in der xMat gewählten Voreinstellungen sind dazu Vergleichsrechnungen mit Ödometer- und Triaxialversuchen durchgeführt worden. Eine Übersicht findet sich in Bild 7.25 und die Ergebnisse einzelner Versuche in den Bildern 7.26 und 7.27. Die Anzahl der insgesamt akzeptierten und abgelehnten Integrationsschritte ist in Tabelle 7.5 und 7.6 zu sehen.

Für die Versuche sind Fortran-Programme mit dem Intel Fortran-Compiler erzeugt worden, die die UMAT-Schnittstelle nutzen. Anstatt die Experimente direkt auszuführen, werden sie mit dem Programm Valgrind[6] beim Ausführen analysiert. Dadurch können die jeweils benötigten CPU-Instruktionen bei der Programmausführung ermittelt werden, die in Bild 7.25 über dem gewählten Zeitschritt dargestellt sind. Diese Herangehensweise hat den Vorteil, dass die Anzahl an Instruktionen beim Ausführen eines Programms im Allgemeinen deutlich stabiler ist als die gemessene Programmlaufzeit. Dabei wird jedoch nicht berücksichtigt, dass die Ausführungsdauer verschiedener Instruktionen im Allgemeinen nicht gleich ist.

[6] Valgrind ©The Valgrind Developers: »https://valgrind.org/

Tabelle 7.4: Anzahl an Aufrufen von `Calculate_Dot_State()` und somit Auswertungen des Stoffmodells der unterschiedlichen Integrationsverfahren in einem einzigen Integrationsschritt. Falls die Jacobi-Matrix numerisch abgeschätzt werden soll, sind siebenmal so viele Auswertungen erforderlich. Die Integrationsverfahren mit der FSAL-Eigenschaft können bei mehreren Integrationsschritten Auswertungen einsparen.

Verfahren	Auswertungen	Auswertungen (num. Jac.)
`Eul-Exp`	1	7
`Richard`	2	14
`RK-S 23`	3	21
`RK-BS23`	4 (−1)	28 (−7)
`RK-F 45`	6	42
`RK-CK45`	6	42
`RK-DP45`	7 (−1)	49 (−7)

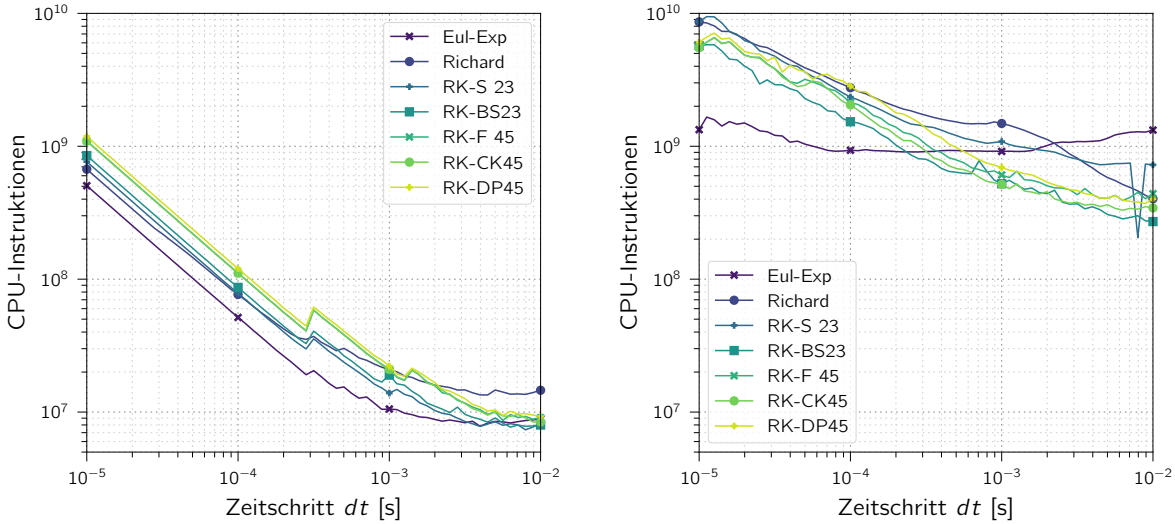


Bild 7.25: Vergleich der notwendigen CPU-Instruktionen für die Berechnung von Ödometerversuchen (links) und Triaxialversuchen (rechts) mit dem maximalen Fehler `setting_max_rel_error` von 10^{-3}. Die Versuche mit dem Stoffmodell `Hypo-VW96`, aktivierter intergranularer Dehnung und Anfangsporenzahl $e_{\mathrm{ini}} = 0{,}7$ sind vergleichbar mit denen aus den vorherigen Abschnitten. Lediglich der Zeitschritt dt und das Integrationsverfahren sind angepasst worden. Die Dehnungsrate ist für alle Ödometerversuche $D = -0{,}05$ und für Triaxialversuche anfangs $D = -1$. Die Markierungen werden nach jeweils 20 Werten ausgegeben. Für $dt \approx 7{,}9 \cdot 10^{-3}$ ist bei den Triaxialversuchen mit `RK-S 23` ein Ausreißer zu sehen, bei dem die Integration fehlgeschlagen ist.

Der Skalierungsfaktor t_S beträgt bei allen dargestellten Ergebnissen 10^{-5}. Es kann gesehen werden, dass Verfahren höherer Ordnung bei größeren Zeitinkrementen weniger Auswertungen benötigen können. In Tabelle 7.5 kann für die Ödometerversuche gesehen werden, dass die Anzahl an Schritten bei `Eul-Exp` oder `Richard` langsamer sinkt, als bei den anderen Verfahren. Das gleiche Verhalten zeigt sich auch für die Triaxialversuche in Tabelle 7.6. Mit größer werdendem Zeitschritt kann für $10^{-3} < dt < 10^{-2}$ jedoch eine sichtbare Abweichung einiger Ergebnisse zur Referenzlösung gesehen werden. Somit ist der gewählte Zeitschritt für diese Berechnungen bereits zu groß gewählt.

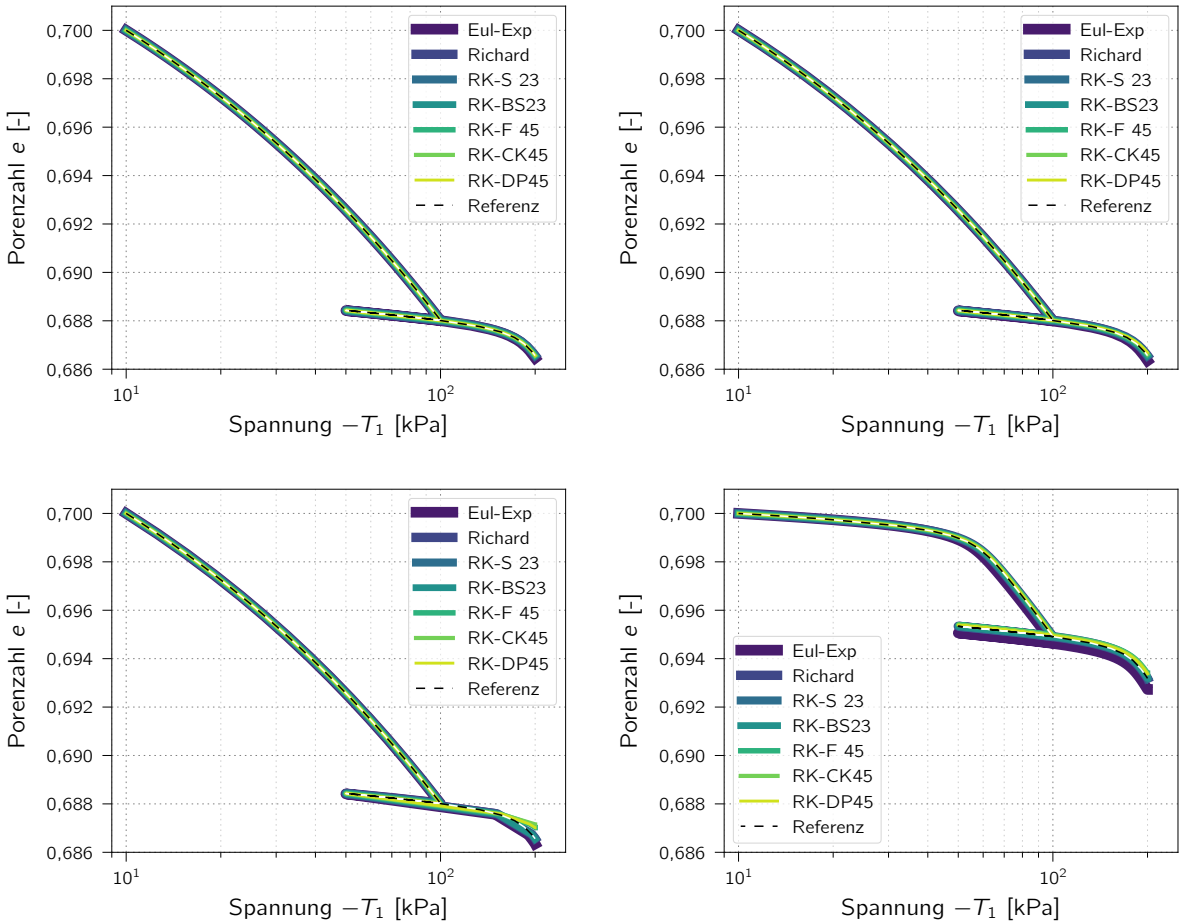


Bild 7.26: Berechnete Ödometerversuche aus Bild 7.25 im Vergleich zur Referenz (`RKDP-45` mit $dt = 10^{-5}$). In der oberen Reihe wird $dt = 10^{-4}$ (links) und $dt = 10^{-3}$ verwendet, in der unteren Zeile links $dt = 10^{-2}$. Der Unterschied wird bei großen Zeitschritten deutlich, wenn sich die intergranularen Dehnungen nach einer Änderung der Belastungsrichtung ebenfalls anpassen, wie bei der Wiederbelastung gesehen werden kann. Im Vergleich dazu zeigen Berechnungen ohne eine zu Beginn voll ausgebildete intergranulare Dehnung, wie im Plot unten rechts mit $dt = 10^{-3}$ dargestellt, bereits in der ersten Belastungsphase deutlichere Abweichungen.

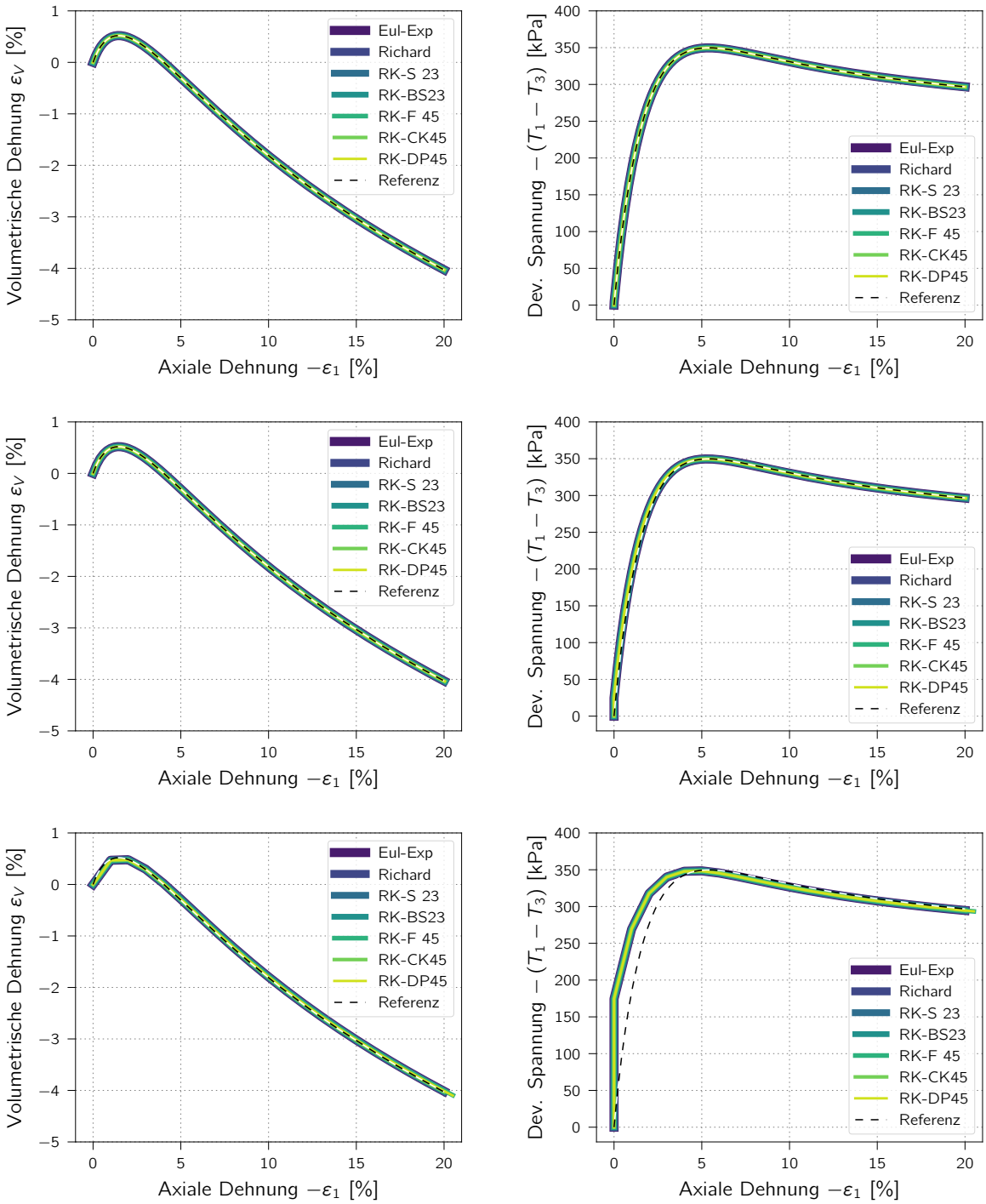


Bild 7.27: Berechnete Triaxialversuche aus Bild 7.25 im Vergleich zur Referenz (`RKDP-45` mit $dt = 10^{-5}$). In der oberen Reihe wird $dt = 10^{-4}$ verwendet, in der mittleren Reihe $dt = 10^{-3}$ und in der unteren Zeile $dt = 10^{-2}$. Die Abweichung ist bei großen Zeitschritten im Plot der deviatorischen Spannung zu erkennen, da durch den unterschiedlichen ersten Wert ein Versatz für den restlichen Verlauf entsteht.

Tabelle 7.5: Akzeptierte und zurückgewiesene (schwarz hinterlegte) Integrationsschritte bei der Berechnung der Ödometerversuche mit dem angegebenen Zeitschritt.

	Eul-Exp	Richard	RK-S 23	RK-BS23	RK-F 45	RK-CK45	RK-DP45
10^{-4}	+1701	+2077	+1712	+1703	+1690	+1691	+1690
	-0	-272	-15	-11	-0	-0	-0
10^{-3}	+863	+906	+421	+539	+356	+356	+352
	-0	-52	-12	-12	-1	-0	-0
10^{-2}	+867	+420	+166	+137	+98	+87	+95
	-0	-30	-15	-9	-0	-0	-1

Tabelle 7.6: Akzeptierte und zurückgewiesene (schwarz hinterlegte) Integrationsschritte bei der Berechnung der Triaxialversuche mit dem angegebenen Zeitschritt.

	Eul-Exp	Richard	RK-S 23	RK-BS23	RK-F 45	RK-CK45	RK-DP45
10^{-4}	27612	8363	6166	7801	6008	6011	6009
	0	39	18	11	1	10	2
10^{-3}	26564	2426	4396	1646	1266	1225	1275
	0	390	544	561	177	171	174
10^{-2}	27629	1646	4701	1184	997	824	931
	0	51	45	69	94	133	58

8 Anwendungsbeispiel

A simulation is the answer to the question: "What if. . . ?"

Hamming (1997), S. 128

In diesem Kapitel wird in einer Versuchsreihe die Verformung eines quaderförmigen Bodenkörpers durch eine parallel verschiebbare Wand (Schieber) untersucht. Es werden Stauchungen von mehr als 30 % betrachtet, wobei ein Versuch Be-, Ent- und Wiederbelastung des Bodenkörpers umfasst. Dabei wird sowohl die Verformung des Bodenkörpers, als auch die Horizontalkraft am Schieber in Abhängigkeit vom Weg gemessen. Als Bodenmaterialien kommen hauptsächlich Hamburger Sand und die Glaskugelmischung zum Einsatz. Es werden experimentelle Untersuchungen und numerische Simulationen durchgeführt und die Ergebnisse von beiden bewertet.

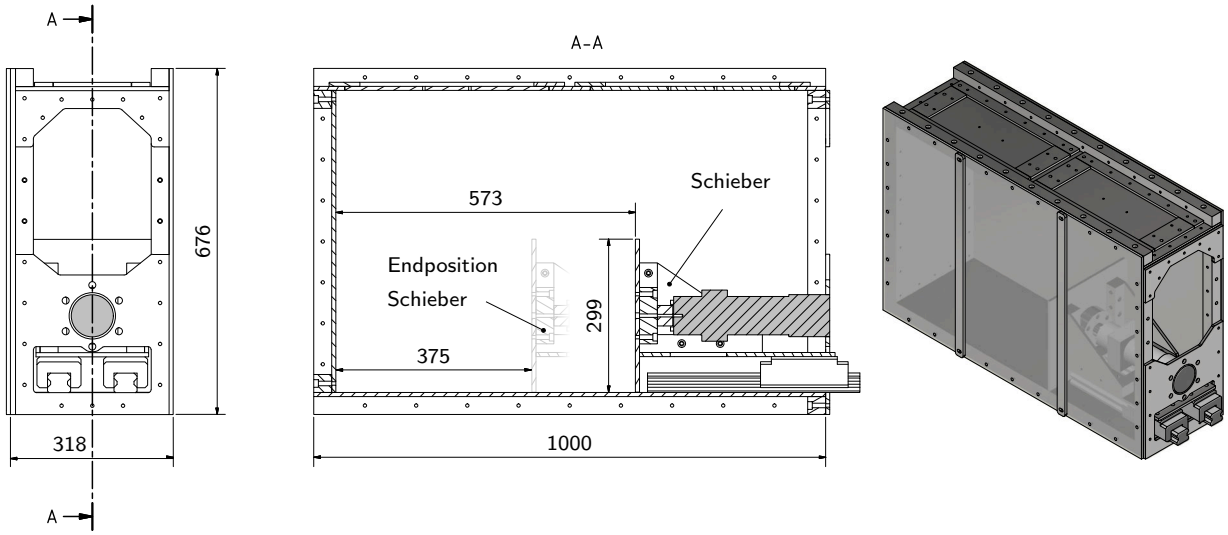
8.1 Planung der experimentellen Versuche

Erddruckversuchsstand

Für die experimentellen Untersuchungen ist der im Folgenden beschriebene Versuchsstand entwickelt worden. Im Innenraum kann ein quaderförmiger Bodenkörper mit einer Breite von 300 mm hergestellt werden. Die Länge des Bodenkörpers ist durch einen parallel beweglichen Schieber begrenzt (siehe Bild 8.1). Der Schieber hat eine Stirnfläche von 299 mm × 299 mm und ist mit einem Schlitten auf Führungsschienen befestigt, sodass er mit einem konstanten Abstand von 0,5 mm zu Boden und Wand bewegt werden kann.[1] Er wird im Versuchsstand von einer Prüfpresse mit max. 100 kN Last und 198 mm Hub angetrieben.

Durch die Anfangsposition des an der Presse befestigten Schiebers ergibt sich für den Bodenkörper bei allen Versuchen eine Einbaulänge von 573 mm und eine gestauchte Länge von 375 mm. Zwischen der Rückwand des Schiebers und der Presse ist eine Verbindungsstange mit Kraftaufnehmer angebaut. Der Kraftaufnehmer GMT K der Genauigkeitsklasse 0,02 erfasst Zug- und Druckkräfte bis zu 1 kN. Leerversuche ohne eingebauten Bodenkörper haben gezeigt, dass für die Bewegung von Schieber und Schlitten mit einem Eigengewicht von 29,6 kg eine absolute Kraft von 66,9 ($\sigma = 7{,}2$) N nötig ist (siehe Tabelle 8.1).

[1] Der Verfahrweg des Schiebers ist konstruktionsbedingt auf 240 mm begrenzt. Die Führungswägen sind für eine ausreichend großes statisches Moment von 2,68 kNm in Bewegungsrichtung ausgelegt.



(a) Seitenansicht und Schnitt A–A **(b)** Isometrische Ansicht

Bild 8.1: Zeichnungen und gerenderte Ansicht zum Erddruckversuchsstand. Die Bauteile sind entweder 8 mm dicke Stahlplatten oder Stangen mit quadratischem Profil von 35 mm × 35 mm, die mit M8-Schrauben miteinander verschraubt sind. An der Vorderseite befindet sich anstelle einer Stahlplatte eine durchsichtige, 10 mm dicke Platte aus kratzfestem Polycarbonat (Makrolon), die durch zwei Stahlleisten versteift wird. Auf der Oberseite befinden sich zwei optional verschließbare Aussparungen mit je 40 cm × 22 cm zum Ein- und Ausbau des Bodenkörpers. Die grau markierte Verbindungsstange hinter dem Schieber ist eine Hubverlängerung mit Kraftaufnehmer für die anschließbare Presse.

Tabelle 8.1: Gemessene Kraft bei Leerversuchen mit verschiedenen Geschwindigkeiten. Es kann eine leichte Abnahme der Kraft mit zunehmender Geschwindigkeit des Schiebers v_{Schieb} erkannt werden. Insgesamt ergibt sich für die Belastungsphase eine Kraft von $64{,}8\,(\sigma = 6{,}4)$ N und für die Entlastungsphase eine Kraft von $-69{,}0\,(\sigma = 7{,}4)$ N.

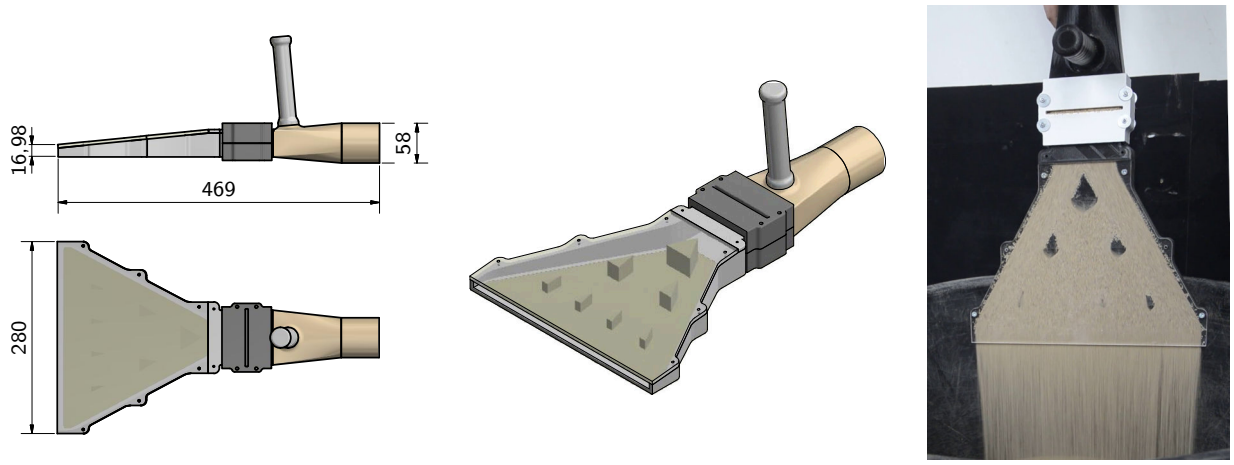
Versuch	v_{Schieb} (mm/min)	Kraft Belastung (N)	Kraft Entlastung (N)
15.1	40	$68{,}6\,(\sigma = 5{,}8)$	$-72{,}9\,(\sigma = 7{,}3)$
15.2	40	$68{,}0\,(\sigma = 5{,}8)$	$-73{,}0\,(\sigma = 7{,}3)$
16.1	80	$66{,}0\,(\sigma = 6{,}0)$	$-70{,}3\,(\sigma = 6{,}6)$
16.2	80	$65{,}6\,(\sigma = 5{,}8)$	$-70{,}9\,(\sigma = 6{,}8)$
09.1	120	$63{,}8\,(\sigma = 6{,}1)$	$-67{,}7\,(\sigma = 6{,}4)$
09.2	120	$63{,}9\,(\sigma = 5{,}5)$	$-67{,}6\,(\sigma = 6{,}1)$
14.1	240	$61{,}7\,(\sigma = 5{,}9)$	$-64{,}9\,(\sigma = 6{,}7)$
14.2	240	$60{,}9\,(\sigma = 5{,}9)$	$-64{,}4\,(\sigma = 6{,}3)$

Diffusor

Für die Versuche soll jeweils ein möglichst homogener Probenkörper mit gleichmäßiger Lagerungsdichte hergestellt werden. Wie in Hettler (2008), S. 305, beschrieben, ist für definierte Verhältnisse bei nichtbindigen Böden ein gleichmäßiges und schichtweises Einrieseln empfehlenswert. Zur Unterstützung wird der Sand mit einem speziell angefertigten Diffusor eingerieselt, der die Sandmenge annähernd gleichmäßig über die Breite des Versuchsstandes verteilen kann (siehe Bild 8.2). Ohne zusätzliche Automatisierung muss der Versuchsleiter eine gleichbleibende Fallhöhe und die gleichmäßige Verteilung über die Längsrichtung sicherstellen.

Versuchsablauf

Die Presse fährt mit einer konstanten Geschwindigkeit v_{Schieb} für Belastung, Entlastung und Wiederbelastung die gleichen Positionen an. Es werden vier verschiedene Geschwindigkeiten $v_{\text{Schieb}} = 40$, 80, 120 und 240 mm/min untersucht, wobei v_{Schieb} innerhalb eines Versuchs jeweils konstant ist. Die Versuche werden von einer Kamera gefilmt, die frontal auf den Bodenkörper gerichtet ist. So kann die Verformung des Probenkörpers sowie die Größe und Form der Scherfugen und Gleitkeile am Computer z. B. mit der *„Particle Image Velocimetry“* ausgewertet werden. Im Rahmen dieser Arbeit ist die PIVlab[2]-Toolbox für MATLAB genutzt worden (siehe Thielicke und Stamhuis, 2014).



(a) Seitenansicht und Draufsicht **(b)** Isometrische Ansicht **(c)** Einrieseln von Sand

Bild 8.2: Eingefärbte Zeichnungen, gerenderte Ansicht und Diffusor im Einsatz. Die Sandmenge aus einem 50 mm-Rohr mit 48 mm Innendurchmesser wird durch den Diffusor auf 27 cm Breite annähernd gleichmäßig verteilt. Dazu ist die Anordnung der Keilstücke zur Umleitung des Materialflusses für die gewählten Materialien und Fallhöhen angepasst worden. Der Schlitz im Übergangsstück erlaubt, den Sandfluss mit einem 80 mm breiten Spatel zu stoppen.

[2] PIVlab ©W. Thielicke, E. J. Stamhuis: »https://pivlab.blogspot.com

Zum Einbau des Bodenkörpers wird Sand in einen Trichter in ca. 2,35 m Höhe gefüllt, der über einen Schlauch mit dem Diffusor verbunden ist. Nach dem Befüllen kann der Diffusor in Position gebracht und das Einrieseln gestartet werden (siehe Bild 8.3). Schichtweise wird so entweder ein ca. 84 mm hoher Bodenkörper aus Hamburger Sand mit einer durchschnittlichen Lagerungsdichte von 1,565 ($\sigma = 0{,}029$) g/cm^3 erzeugt, oder ein ca. 154 mm hoher Bodenkörper aus der Glaskugelmischung mit einer durchschnittlichen Lagerungsdichte von 1,621 ($\sigma = 0{,}012$) g/cm^3 hergestellt (siehe Abschnitt 8.2).

Nach Einbau der Probe wird die Daten- und Videoaufzeichnung gestartet. Anschließend wird die Probe zuerst in der Belastungsphase durch den Schieber um 198 mm gestaucht, dann auf den Ausgangszustand entlastet und in der Wiederbelastungsphase erneut bis auf 198 mm gestaucht. In Tabelle 8.2 ist eine Übersicht der in dieser Arbeit dargestellten Versuche zu sehen.

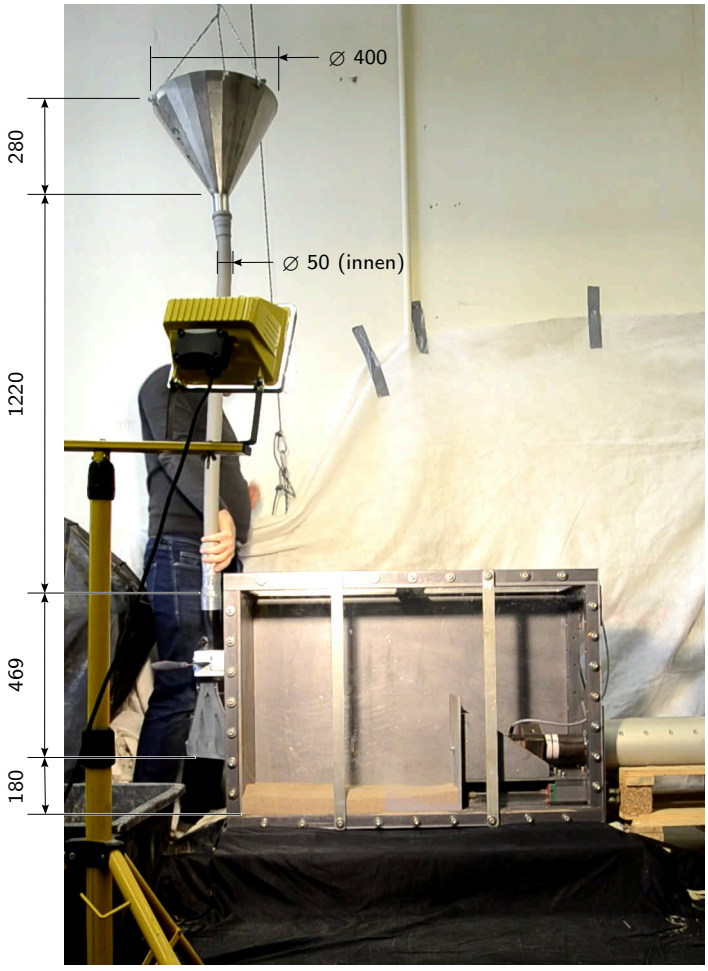


Bild 8.3: Foto von Erddruckversuchsstand mit Einrieselanlage, überlagert mit Längenangaben einzelner Bauteile. Die Videoaufzeichnung ist von der gleichen Position mit Teleobjektiv und Fokussierung auf den relevanten Teil des Versuchsstands durchgeführt worden (siehe bspw. Bild 8.4).

Tabelle 8.2: Die 39 durchnummerierten Versuche, die im Rahmen dieser Arbeit durchgeführt worden sind. Für jede Kombination aus Geschwindigkeit v_{Schieb} und Bodenkörper sind zwischen zwei und fünf Versuche für eine höhere Aussagekraft durchgeführt worden. Die fett markierten Versuche werden für Vergleiche, wie in Bild 8.5 und 8.9, verwendet.

Bodenkörper	40 mm/min	80 mm/min	120 mm/min	240 mm/min
Glaskugelmischung	**10**, **12**	01, **02**, **03**	05, 06, **07**, **08**	**11**, **13**
Hamburger Sand	26, 28, **34**, **35**	19, 20, 21, **30**, **31**	22, 25, 27, **32**, **33**	23, 24, 29, **36**, **37**
— (Leerversuche)	15, 38	04, 16, 40	09, 39	14, 41

8.2 Auswertung der Modellversuche

Glaskugeln SiLibeads 3 mm und 5 mm

Da Länge und Breite des Bodenkörpers zu Versuchsbeginn 573 mm und 300 mm betragen, kann über das Gewicht der Glaskugeln und der mittleren Höhe die Lagerungsdichte zu Versuchsbeginn abgeschätzt werden (siehe Tabelle 8.3 und Bild 8.4).

Die Kräfte über den Weg sind in Bild 8.5 dargestellt. Im Kraft-Weg-Diagramm können mehrere Auffälligkeiten beobachtet werden. Zum einen scheint die Presse in der Belastungsphase eine Art Anfahrwiderstand überwinden zu müssen, bevor sie kontinuierlich bis zum Ende ansteigt. Zu Beginn der Entlastungsphase springt die Kraft auf ungefähr null und sinkt bis zum Ausgangszustand auf ca. −75 N. Das entspricht der notwendigen Kraft zur Bewegung des Schiebers, die in Leerversuchen ermittelt worden ist.

In der Entlastungsphase können Kraftschwankungen erkannt werden, die sich im Abstand von 16 mm wiederholen und abgeschwächt auch in den anderen Phasen auftreten. Die Unterschiede zwischen den Versuchen und unterschiedlichen Geschwindigkeiten sind relativ gering und kaum größer als das Rauschen der Kraftdaten von ±5 N. Lediglich in der Belastungsphase fällt ein Ausreißer bei 170 mm und eine wachsende Abweichung ab ca. 150 mm auf, die aber nicht an der Wahl der Geschwindigkeit zu liegen scheint.

Tabelle 8.3: Daten zu den Versuchen im Erddruckversuchsstand mit Glaskugeln. Aus dem Gewicht des Bodenkörpers m und der Anfangshöhe h_{ini} sowie Daten aus Tabelle 6.3 auf S. 120 wird die durchschnittliche Lagerungsdichte ρ_{ini} und die Anfangsporenzahl e_{ini} abgeleitet.

v_{Schub} (mm/min)	m (kg)	h_{ini} (mm)	ρ_{ini} (g/cm^3)	e_{ini} (-)	Versuch
40	42,72	153	1,62	0,625	10
40	42,74	152	1,64	0,613	12
80	42,90	154	1,62	0,628	2
80	42,86	154	1,62	0,63	3
120	42,74	155	1,60	0,645	7
120	42,72	155	1,60	0,646	8
240	42,72	152	1,64	0,614	11
240	42,72	153	1,62	0,625	13

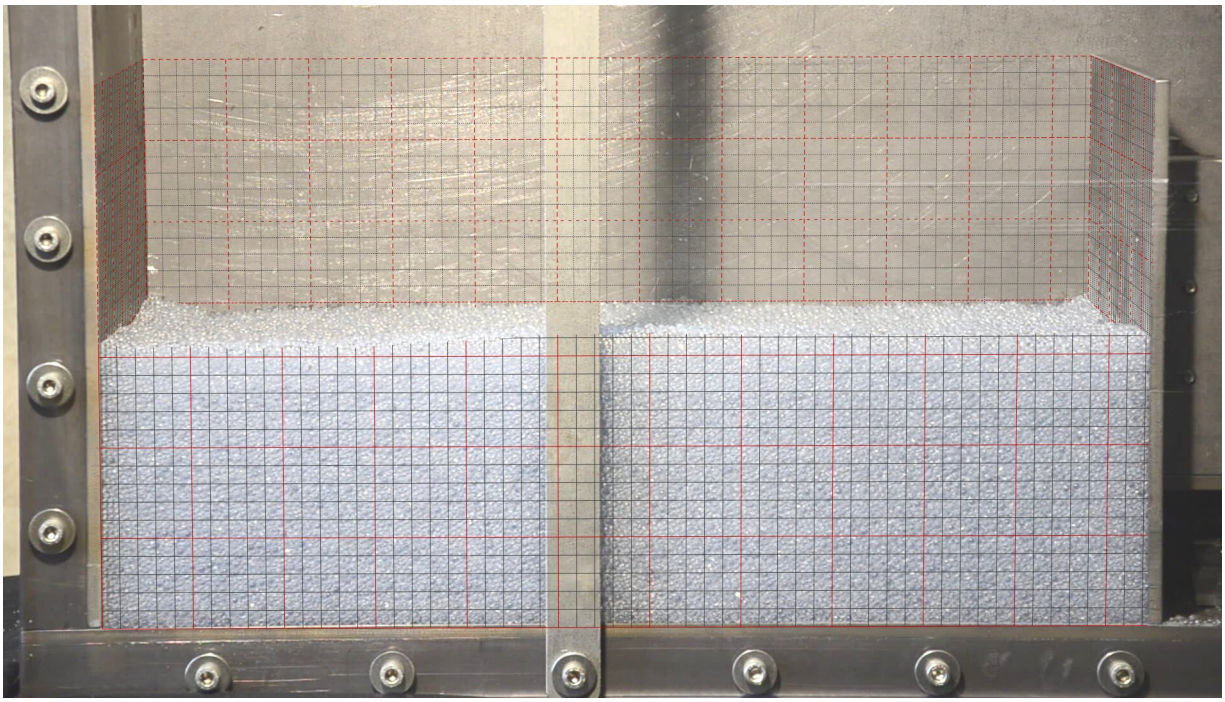


Bild 8.4: Foto vom Anfangszustand eines typischen Versuchs mit Glaskugeln (Versuch 13) im Erddruckversuchsstand. Das überlagerte Gitter dient zur Abschätzung der Längen des Probenkörpers ($573 \times 300 \times 153\,\mathrm{mm}$).

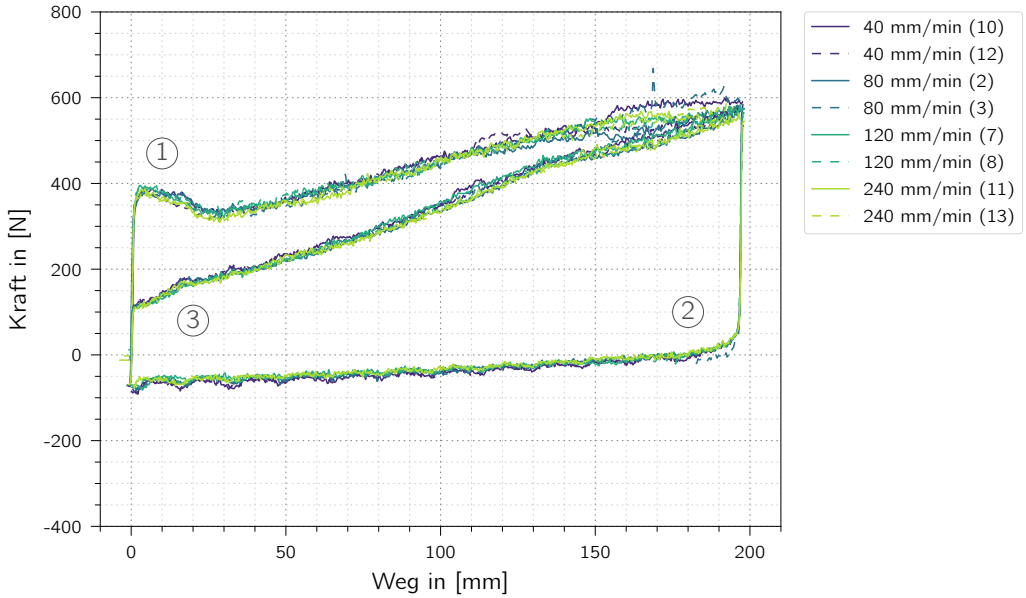


Bild 8.5: Kraftverlauf am Messaufnehmer hinter dem Schieber des Erddruckversuchsstands für ausgewählte Versuche mit der Glaskugelmischung. Dargestellt über der Position des Schiebers sind die Kräfte in (1) der Belastungsphase, (2) der Entlastungsphase und (3) der Wiederbelastungsphase für vier verschiedene Geschwindigkeiten v_{Schieb}.

Aus den Videodaten der Versuche können mit PIVlab die Bewegungsvektoren einzelner Bereiche zwischen zwei Bildern berechnet und visualisiert werden. In dieser Arbeit sind pro Bewegungsvektor Felder mit einer Größe von 16×16 Pixel genutzt worden. In Bild 8.6 sind die Gleitkörper und Gleitwinkel für einzelne Zeitpunkte der Be-, Ent- und Wiederbelastungsphase dargestellt.

Wird in jeder Belastungsphase die Form des Bodenkörpers in der Ausgangssituation und nach vier gleich großen Längenänderungen um jeweils 49,5 mm betrachtet, so ergeben sich die ebenfalls in Bild 8.6 dargestellten Verteilungsformen. Es wird die Bodenform an der Makrolonplatte als Grundlage für die Verteilungsform herangezogen, was einerseits in den Videodaten besser erkannt werden kann als bspw. das Mittelprofil, andererseits aber stärker tiefenabhängigen Effekten wie Reibung unterliegt.

Insgesamt ist anzumerken, dass die Glaskugeln durch Bewegung des Probenkörpers zur Entmischung neigen und die Konzentration kleinerer Kugeln im unteren Teil des Probenkörpers zunimmt.

Hamburger Sand

Wie bei der Glaskugelmischung wird auch für Hamburger Sand die Lagerungsdichte zu Versuchsbeginn über Gewicht und mittlere Höhe des Bodenkörpers abgeschätzt (siehe Tabelle 8.4 und Bild 8.7). Ebenso sind auch für die Versuche mit Hamburger Sand Verteilungsformen und die Bewegungsvektoren zwischen Einzelbildern mit PIVlab berechnet und visualisiert worden (siehe Bild 8.8). Das Kraft-Weg-Diagramm ist in Bild 8.9 zu sehen.

Im Gegensatz zur Glaskugelmischung fällt das große Rauschen von ca. ± 30 N der einzelnen Datensätze auf, was durch den vorbeigedrückten Sand erzeugt oder maßgeblich beeinflusst wird. Ursprünglich ist auf eine Dichtung rund um den Schieber verzichtet worden, um die gemessenen Kräfte am Schieber nicht zu verfälschen. Da die Abstände zwischen Schieber und Behälter für Hamburger Sand etwas zu groß sind, entstehen dadurch die großen Signalschwankungen.

Tabelle 8.4: Daten zu den Versuchen im Erddruckversuchsstand mit Hamburger Sand. Aus dem Gewicht des Bodenkörpers m und der Anfangshöhe h_{ini} sowie Daten aus Tabelle 6.1 auf S. 119 wird die durchschnittliche Lagerungsdichte ρ_{ini} und die Anfangsporenzahl e_{ini} abgeleitet. Bei den Versuchen hat sich bis zum Versuchsende eine kleine Menge Sand Δm am Schieber vorbeigedrückt.

v_{Schub} (mm/min)	m (kg)	Δm (kg)	h_{ini} (mm)	ρ_{ini} (g/cm^3)	e_{ini} (-)	Versuch
40	22,45	0,18	83	1,57	0,623	34
40	22,45	0,28	80	1,63	0,564	35
80	22,45	0,16	84	1,56	0,642	30
80	22,45	0,14	84	1,56	0,642	31
120	22,45	0,22	84	1,56	0,642	32
120	22,45	0,12	84	1,56	0,642	33
240	22,45	0,10	84	1,56	0,642	36
240	22,45	0,14	85	1,54	0,662	37

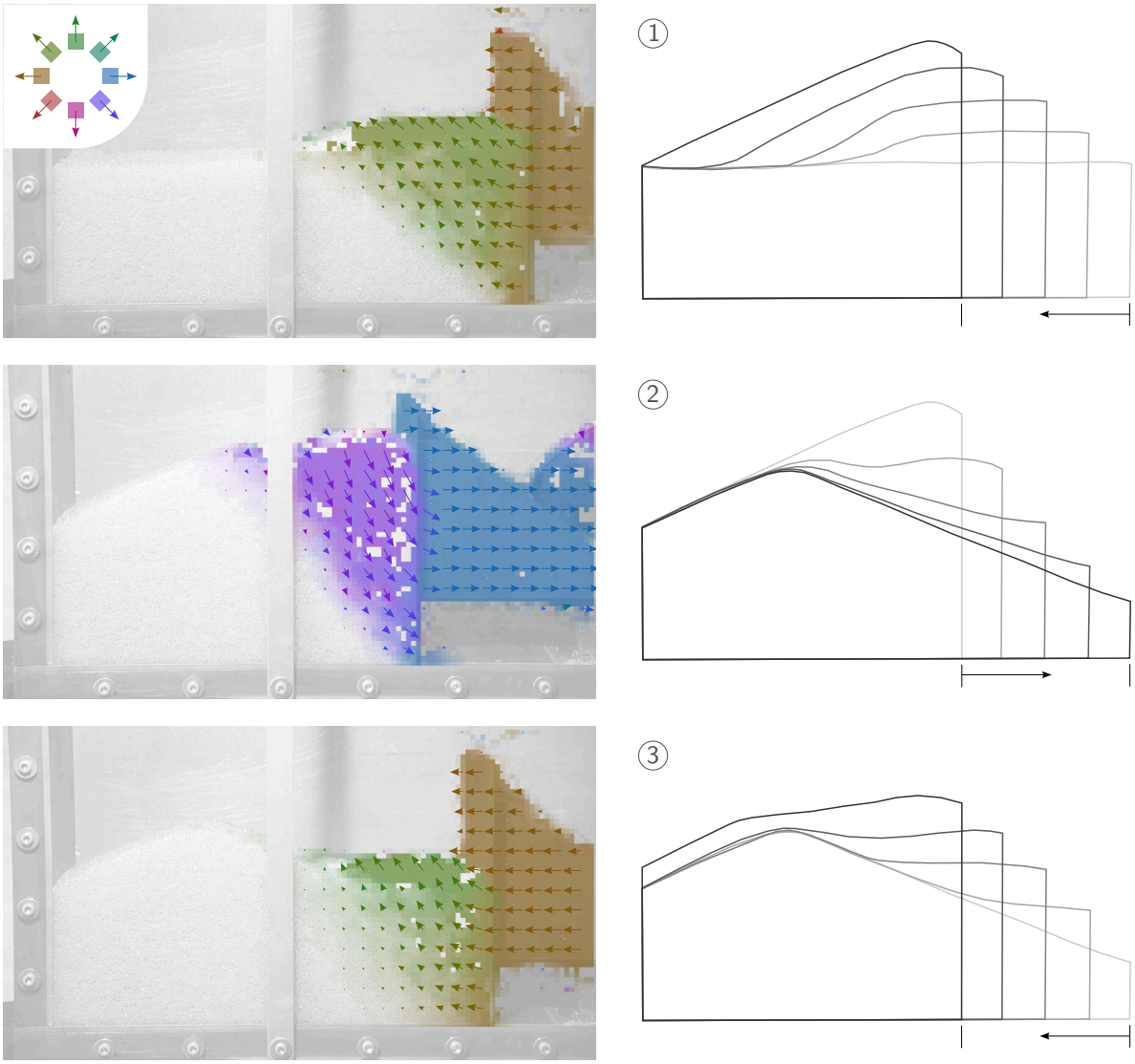


Bild 8.6: Ergebnisse in der Belastungsphase (obere Reihe), Entlastungsphase (mittlere Reihe) und Wiederbelastungsphase (untere Reihe) für Versuch 13 mit der Glaskugelmischung. Links: Einzelaufnahmen aus den drei Phasen mit überlagerten Geschwindigkeitsvektoren der PIV-Analyse. Die Bewegungsrichtung ist sowohl mit Pfeilen als auch über die Farbgebung symbolisiert. Nicht gekennzeichnete Bereiche im Gleitkörper stellen Felder dar, in denen die Geschwindigkeitsvektoren nicht eindeutig ermittelt werden konnten. In den Videodaten kann jedoch eine gleichmäßige Bewegung beobachtet werden. Rechts: Aus den Videodaten ermittelte Form des Probenkörpers an der Makrolonplatte zu einzelnen Zeitpunkten der Versuchsdurchführung. Die hellste Kontur entspricht dem Ausgangszustand der jeweiligen Belastungsphase und die dunkelste Kontur dem Endzustand.

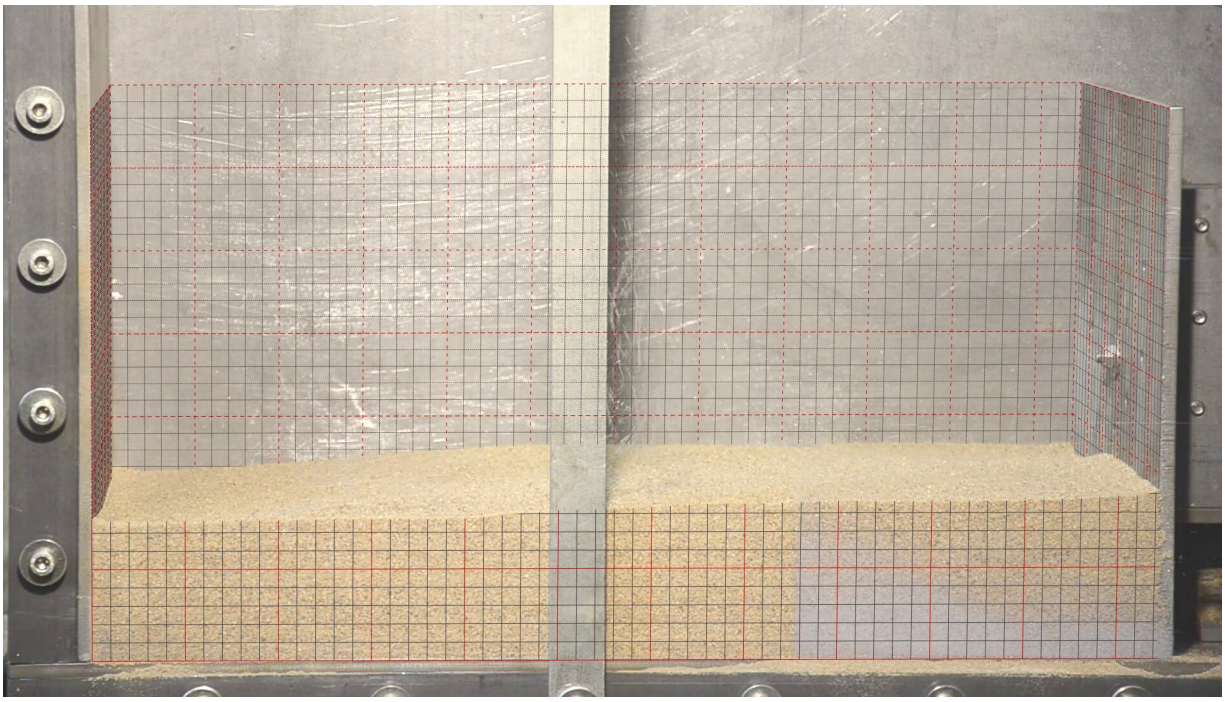


Bild 8.7: Foto vom Anfangszustand eines typischen Versuchs mit Hamburger Sand (Versuch 35) im Erddruckversuchsstand. Das überlagerte Gitter dient zur Abschätzung der Längen des Probenkörpers ($573 \times 300 \times 80$ mm). Im rechten unteren Bereich ist die Makrolonplatte aufgeraut und getrübt, da bei den Versuchen Sand zwischen Schieber und der Platte vorbeigedrückt worden ist.

Bei den Versuchen mit Hamburger Sand ist kein Anfahrwiderstand am Anfang der Belastungsphase zu erkennen. Die Belastungsphase startet bei allen Versuchen mit Glaskugelmischung und Hamburger Sand bei ungefähr 400 N. Aufgrund des unterschiedlichen Gleitwinkels und der größeren Reibung bei Hamburger Sand ist trotz der unterschiedlichen Menge (ca. 22 kg Hamburger Sand und ca. 42 kg Glaskugeln) eine größere Kraft für die Stauchung des Bodenkörpers mit Hamburger Sand notwendig.

Im Kraftverlauf der Entlastungsphase wird trotz andrückendem Bodenkörper eine ca. dreimal größere Kraft gemessen, als für die Bewegung des Schiebers in den Leerversuchen notwendig ist (siehe Tabelle 8.1). Der Hauptgrund für diese zusätzliche Kraft liegt in den Sandkörnern, die bei der Belastungsphase unter dem Schlitten vorbeigedrückt worden sind und bei der Entlastung erneut einen Widerstand beim Überfahren erzeugen. Obwohl diese Kraft anfangs gleichbleibend wirkt, ist eine Abnahme zwischen 110 mm und 80 mm erkennbar. Nach 80 mm liegt der Bodenkörper nicht mehr am Schieber an, sodass sich die gemessene Kraft anschließend nicht mehr wesentlich ändert. In der Belastungsphase wird der Bodenkörper ebenfalls erst nach ca. 80 mm erreicht, was der Kraftverlauf widerspiegelt.

Durch die Reibung an den Seitenflächen ist die Verformung in Tiefenrichtung nicht gleichmäßig (siehe Bild 8.10). Außerdem stellen sich mehrere Abstufungen in der Böschung ein, die wahrscheinlich auf kleine Schwankungen beim Verfahren des Schiebers zurückzuführen sind.

Aus der Auswertung der Videodaten und mit PIVlab verarbeiteten Einzelbilder werden für beide Materialien durchschnittliche Böschungswinkel φ und Gleitwinkel im passiven Fall ϑ_p und aktiven Fall ϑ_a bestimmt (siehe Tabelle 8.5).

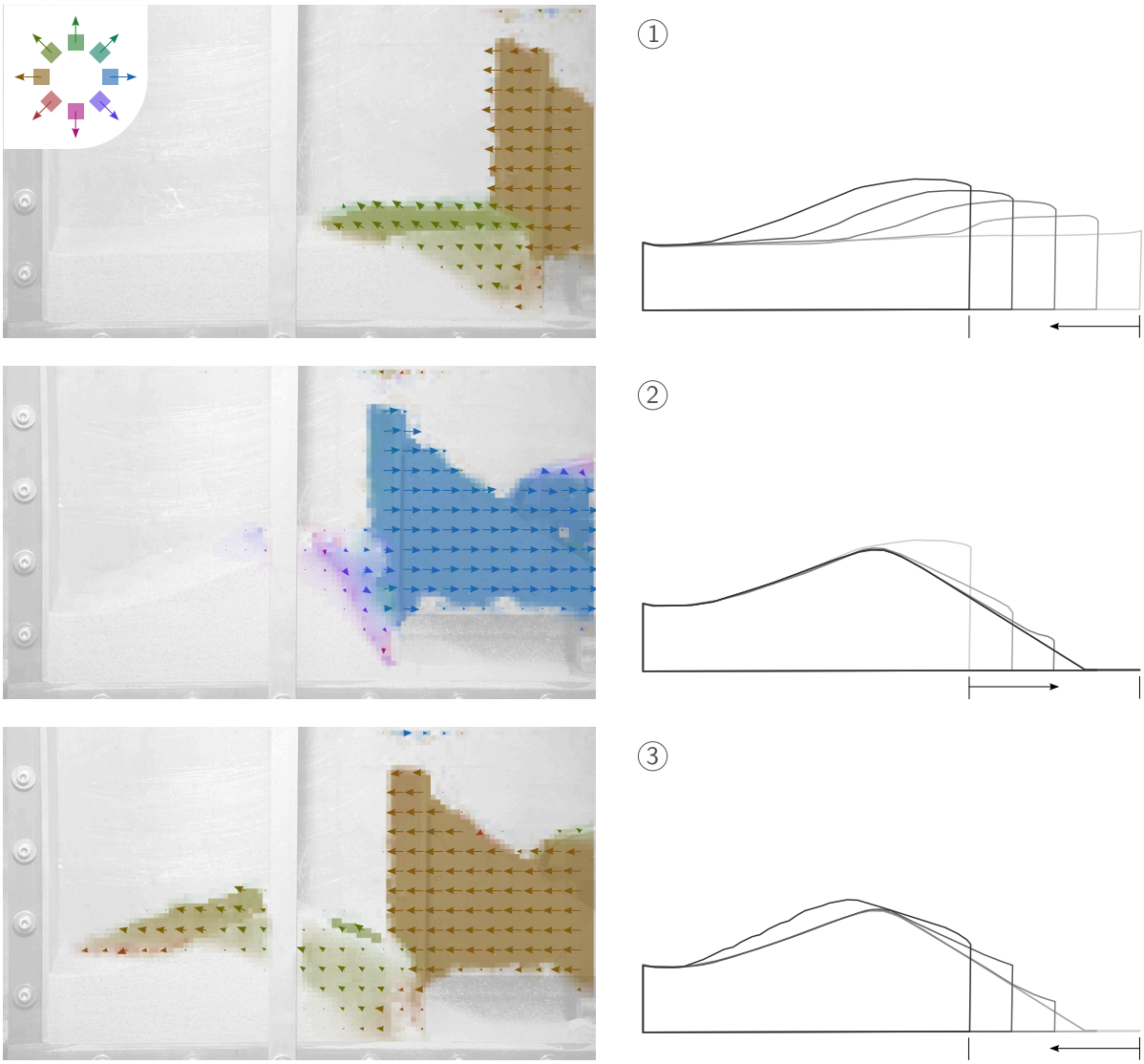


Bild 8.8: Ergebnisse in der Belastungsphase (obere Reihe), Entlastungsphase (mittlere Reihe) und Wiederbelastungsphase (untere Reihe) für Versuch 35 mit Hamburger Sand. Links: Einzelaufnahmen aus den drei Phasen mit überlagerten Geschwindigkeitsvektoren der PIV-Analyse. Die Bewegungsrichtung ist sowohl mit Pfeilen als auch über die Farbgebung symbolisiert. Die Bewegung der Sandoberfläche wird bei der Entlastung und Wiederbelastung nur teilweise erkannt. Rechts: Aus den Videodaten ermittelte Form des Probenkörpers an der Makrolonplatte zu einzelnen Zeitpunkten der Versuchsdurchführung.

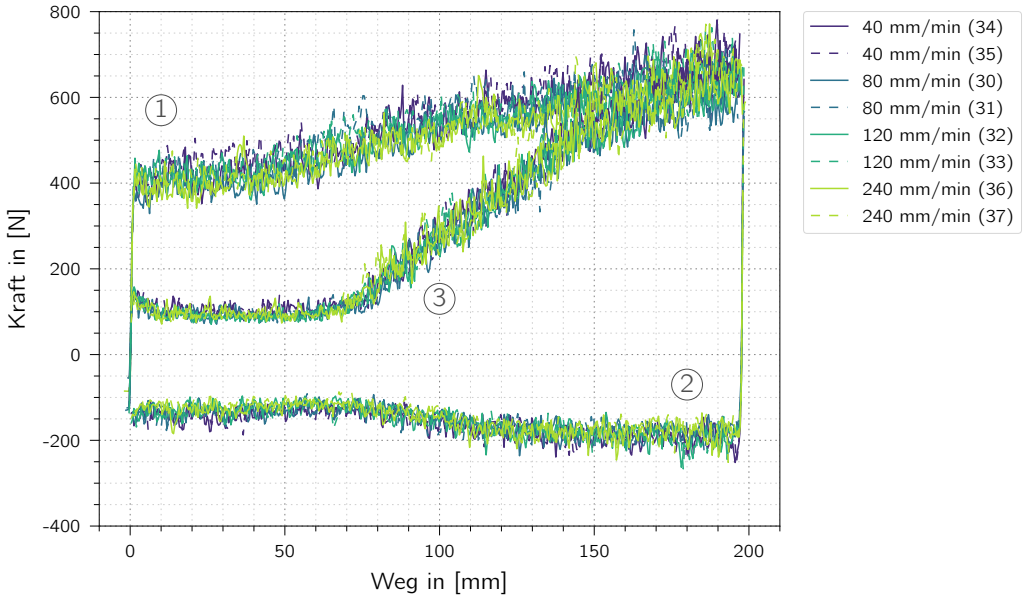


Bild 8.9: Kraftverlauf am Messaufnehmer hinter dem Schieber des Erddruckversuchsstands für ausgewählte Versuche mit Hamburger Sand. Dargestellt über der Position des Schiebers sind die Kräfte in (1) der Belastungsphase, (2) der Entlastungsphase und (3) der Wiederbelastungsphase für vier verschiedene Geschwindigkeiten v_{Schieb}.



(a) Seitenansicht **(b)** Draufsicht **(c)** Rückansicht

Bild 8.10: Fotos vom Endzustand eines typischen Versuchs im Erddruckversuchsstand mit Hamburger Sand. Wie aus der Seitenansicht und der Draufsicht gesehen werden kann, ist eine leicht ausgebauchte Böschung mit mehreren Abstufungen erzeugt worden. Links unten in der Rückansicht kann der am Schieber vorbeigedrückte Sand als Anhäufung nach Be-, Ent- und Wiederbelastung gesehen werden.

Tabelle 8.5: Aus den experimentellen Untersuchungen ergeben sich im Mittel die dargestellten Böschungswinkel φ, Gleitwinkel im passiven Fall ϑ_p und aktiven Fall ϑ_a.

Material	φ (Grad)	ϑ_p (Grad)	ϑ_a (Grad)
Glaskugeln	24	40	55
Hamburger Sand	22	34	60

8.3 Idealisierte Verteilungsformen

Mit der beschriebenen Versuchsdurchführung wird der Boden zuerst belastet, anschließend entlastet und am Ende wieder belastet. In diesem Abschnitt wird die Verformung des Bodenkörpers in den drei Phasen an einem idealisierten Beispiel untersucht. Dazu werden folgende Annahmen getroffen (siehe auch Anhang H):

- Das Verformungsverhalten ist über die Breite konstant, sodass nur die Fläche in der Seitenansicht betrachtet werden kann.
- Die betrachtete Fläche ändert ihren Flächeninhalt nicht.
- Verformungen werden nur von äußeren Effekten gesteuert.
- Bei jeder Verformung wird eine starre Fläche über einer geradlinigen Scherfuge mit dem Winkel ϑ entlang der Scherfuge bewegt. Ein schmaler Übergangsbereich dA_x mit der Dicke dx in der Scherfuge wird dabei verzerrt, ohne den Flächeninhalt zu ändern. In einem zweiten Schritt wird die vormals starre Fläche an freien Enden ggf. passend unter dem Böschungswinkel φ neu verteilt.
- Der Böschungswinkel φ ist konstant, der Scherfugenwinkel ist im Belastungsfall ϑ_p und im Entlastungsfall ϑ_a.

Anschließend können die Verteilungsformen für einen anfangs quaderförmigen Bodenkörper allein aus der Geometrie hergeleitet werden. Für die Erstbelastung sei hier auf die Zusammenhänge in Bild 8.11 verwiesen. Mit den Werten aus Tabelle 8.5 ergeben sich die in Bild 8.12 dargestellten Ergebnisse. Im Vergleich zu den Verteilungsformen aus den experimentellen Untersuchungen (Bild 8.6 und 8.8) sind die Ergebnisse geradliniger, da kein granularer Bodenkörper mit Imperfektionen, Reibung sowie Verdichtungs- und Auflockerungseffekten betrachtet wird.

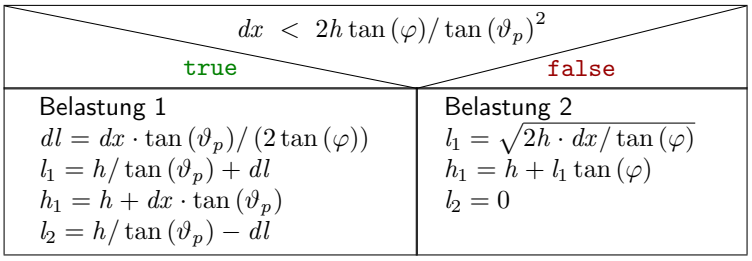


Bild 8.11: Bei den geometrischen Zusammenhängen der Erstbelastung wird mit der Bedingung $dx < 2h \tan(\varphi) / \tan(\vartheta_p)^2$ unterschieden, ob nach der Verformung ein horizontales Plateau vorhanden ist oder nicht (siehe auch Bild H.3). Dabei ist l_2 die horizontale Länge des angehobenen Plateaus, h_1 die maximale Höhe und l_1 die horizontale Entfernung vom Schieber bis zum unteren Ende der Böschung.

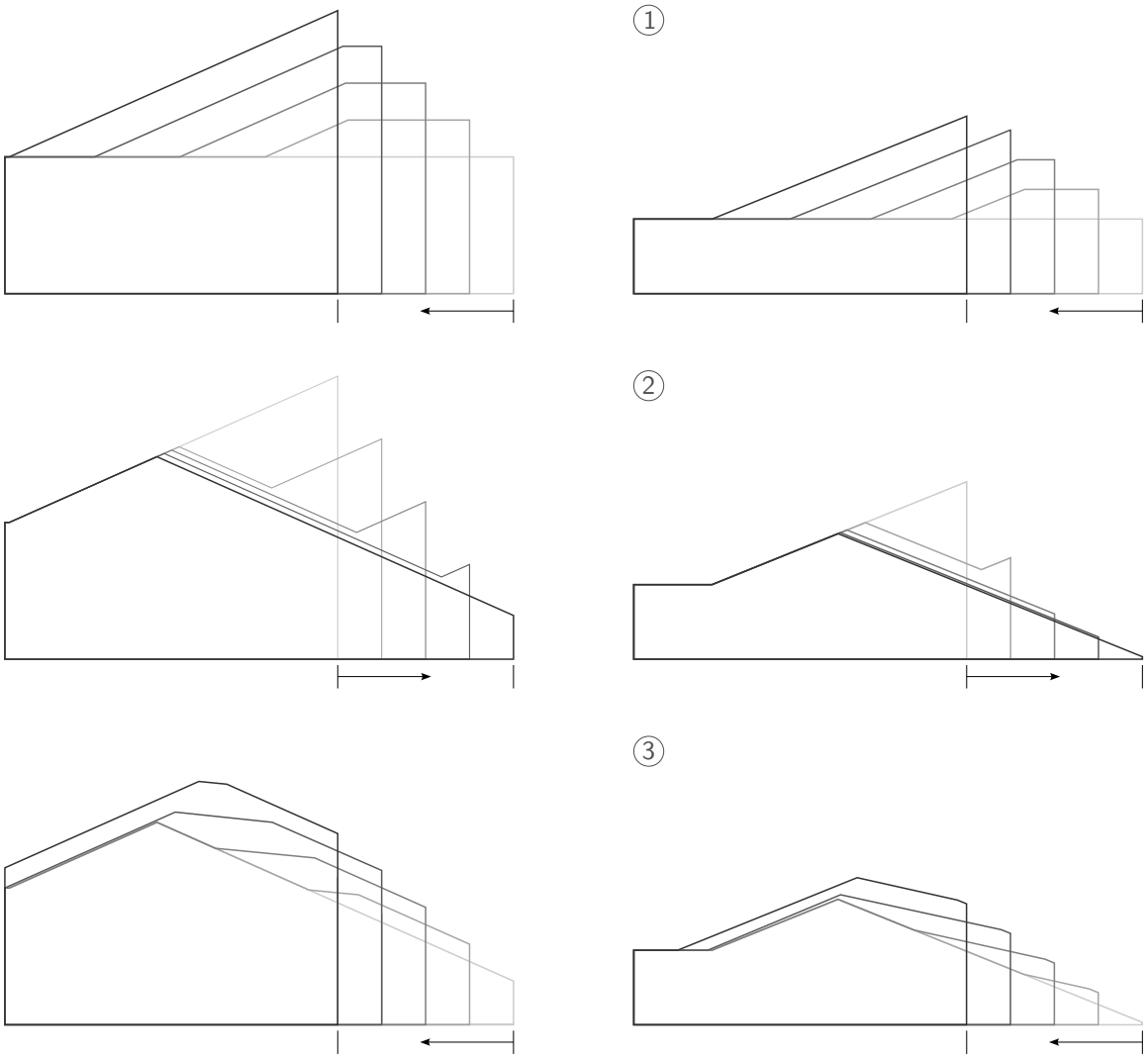


Bild 8.12: Idealisierte Verteilungsformen der Be-, Ent- und Wiederbelastungsphase für einen Bodenkörper mit einer Länge von 573 mm und einer Anfangshöhe $h_{\mathrm{ini}} = 154\,\mathrm{mm}$ für die Glaskugelmischung (links) bzw. $h_{\mathrm{ini}} = 84\,\mathrm{mm}$ für Hamburger Sand (rechts).

8.4 Planung der numerischen Simulationen

Für die numerischen Untersuchungen wird Abaqus verwendet. Typischerweise werden Modelle in Abaqus als `.cae`-Dateien mit einem Journal als `.jnl`-Datei gespeichert. Im Allgemeinen können die `cae`-Dateien nicht in älteren Abaqus-Versionen eingelesen werden und müssen vor der Nutzung in einer neueren Version entsprechend konvertiert werden. Da alle relevanten Aufgaben in Abaqus über Python-Befehle gesteuert werden können, die größtenteils im Journal dokumentiert werden, bietet sich für eine hohe Kompatibilität eine skriptbasierte Vorgehensweise an.

abapys

abapys[3] und das grafische Frontend basierend auf dem *SimpleScriptGenerator*[4] sind im Rahmen dieser Arbeit erstellt worden, um die Modellerstellung und -auswertung von geotechnischen Abaqus-Modellen mithilfe von Python-Skripten zu vereinfachen. Typische Arbeitsvorgänge und Teilaufgaben werden in abapys als fertige Funktionen zur Verfügung gestellt. Dadurch kann Abaqus effizienter bedient und Arbeitsabläufe können in kompakten Skripten gespeichert werden.

abapys ist eine betriebssystemunabhängige Bibliothek von Python 2.7-Befehlen, die für Abaqus 6.14.1 bis Abaqus 2020 getestet worden ist. Eine Hilfsfunktion ist aus Effizienzgründen als C++-Bibliothek ausgegliedert. Die C++-Bibliothek selbst ist ebenfalls betriebssystemunabhängig und muss lediglich für das Zielsystem kompiliert werden. abapys ist dafür gedacht, aus Abaqus heraus mit dem Python-Interpreter der Abaqus-Installation benutzt zu werden.

Modellbeschreibung

Die numerischen Modelle orientieren sich an den Randbedingungen der experimentellen Untersuchungen, die in Abschnitt 8.1 beschrieben werden. Sie können alle über ein parametrisiertes Python-Skript erstellt werden (siehe Anhang F).

Der Bodenkörper hat eine Länge von 573 mm und eine Höhe von 84 mm für Hamburger Sand bzw. eine Höhe von 154 mm für die Glaskugelmischung. Über die Breite wird nur ein 1 cm breiter Streifen statt des 30 cm breiten Bodenkörpers betrachtet. Der Anfangszustand ist in Bild 8.13 dargestellt.

Der Versuchsbehälter wird am linken und unteren Rand durch einen räumlich fixierten Starrkörper abgebildet. Auch der Schieber am rechten Rand wird als Starrkörper modelliert. Zwischen beiden Starrkörpern und dem Bodenkörper ist jeweils ein reibungsbehafteter Kontakt mit einem Reibungskoeffizienten $\mu_R = 0{,}22$ definiert. Im Gegensatz dazu besteht kein Kontakt und keine Interaktion zwischen Schieber und Starrkörper.

[3] abapys ©D. Zobel: »https://github.com/d-zo/abapys
Der Name abapys setzt sich aus Abaqus und Python zusammen.

[4] SimpleScriptGenerator ©D. Zobel: »https://github.com/d-zo/SimpleScriptGenerator

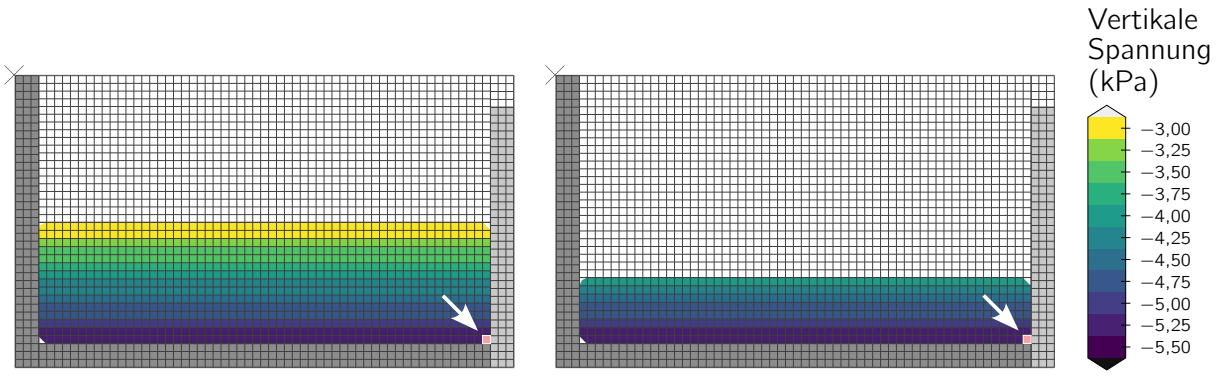


Bild 8.13: Numerisches Modell der Erddruckversuche mit der Glaskugelmischung (links) sowie Hamburger Sand (rechts). Die Gitterweite entspricht ungefähr 10 mm. Der Versuchsbehälter wird im linken und unteren Bereich durch dunkelgraue Elemente symbolisiert, die starr und räumlich fixiert sind. Am rechten Rand der Modelle ist jeweils der Schieber zu sehen, der nur mit dem Bodenkörper und nicht mit dem Versuchsbehälter interagiert. Während der Bodenkörper mit einer Anfangsporenzahl von 0,575 (Glaskugelmischung) bzw. 0,686 (Hamburger Sand) initialisiert wird, hat das markierte Element die Anfangsporenzahl e_{c0}, um dort die Bildung einer Scherfuge zu begünstigen.

Der Bodenkörper selbst ist als Euler-Material mit quaderförmigen `EC3D8R`-Elementen diskretisiert worden. Dabei wird nur der Teil des Bodenkörpers im Versuchsbehälter mit Bodenmaterial initialisiert, der der Anfangshöhe entspricht. Die Kantenlänge der Elemente entspricht ungefähr 1 cm. Abaqus/Explizit kann mit der *gekoppelten Euler-Lagrange-Methode (CEL)* die Verformung des Bodenkörpers über den Füllgrad der einzelnen Elemente mit Bodenmaterial darstellen.[5] In jedem Element werden die Spannungen über den gemittelten Spannungszustand im jeweiligen Elementvolumen beschrieben. Im Ausgangszustand wird vertikal eine mit der Tiefe zunehmende Spannung von mindestens 0,1 kPa ab Oberkante des Euler-Materials angenommen und horizontal ein K_0-Zustand.

Die Anfangsporenzahl für die Simulation mit Hamburger Sand (siehe Tabelle 6.2 auf S. 119) beträgt $e_{\text{ini}} = 0{,}686$. Für die Glaskugelmischung wird $e_{\text{ini}} = 0{,}575$ verwendet. Beide Werte entsprechen jeweils den in Abschnitt 8.1 erwähnten durchschnittlichen Einbaudichten der experimentellen Versuche mit dem entsprechenden Material.

Die horizontale Bewegung des Schiebers um ±198 mm wird nach der Initialisierung des Modells in drei Simulationsschritten ausgeführt. Jeder Simulationsschritt der expliziten Analyse dauert 5 s und es werden jeweils 1000 Zwischenergebnisse pro Simulationsschritt ausgegeben. In der xMat sind `setting_initial_substep_scale` auf 10^{-5} und `setting_very_small_stress` für das elastische Ersatzmodell auf $\sigma_{\text{el}} = -10^{-5}$ gesetzt.

Die Anpassung von σ_{el} ist erforderlich, da die Druckspannungen an der freien Oberfläche des Bodenkörpers durch die Verformung sehr klein werden. Da sich das Materialverhalten mit dem elastischen Ersatzmodell verändert, beeinflusst σ_{el} die Form des Bodenkörpers.

[5] Mehr zur gekoppelten Euler-Lagrange-Methode findet sich bspw. in Noh (1963) und Benson (1992), S. 366–368. Qiu u. a. (2011) zeigt Anwendungsbeispiele der CEL-Methode in der Geotechnik auf.

8.5 Auswertung der numerischen Simulationen

In den numerischen Simulationen zeigt sich mit den in Tabelle 6.2 und 6.4 aufgelisteten Materialparametern ein völlig anderes Verformungsverhalten, als in den experimentellen Untersuchungen beobachtet worden ist. Moderate Optimierungen der Materialparameter haben nur einen geringen Effekt und erst mit der Anpassung der Granulathärte h_S und des Exponenten n_H beider Parametersätze auf $h_S = 5{,}8 \cdot 10^6\,\mathrm{kPa}$ und $n_H = 0{,}28$ zeigen die numerischen Simulationen ein vergleichbares Verhalten. Da im Folgenden nur die grundsätzliche Eignung der xMat für FE-Analysen von Interesse ist, werden die Ergebnisse mit den modifizierten Parametersätzen für beide Materialien vorgestellt.

Die Scherfugen im Bodenkörper werden über die Invariante der deviatorischen Dehnung ε_q abgeschätzt. Dabei wird ε_q wie folgt bestimmt

$$\begin{aligned}\varepsilon_q &= \sqrt{\frac{2}{3}\boldsymbol{\varepsilon}^{\mathrm{dev}} : \boldsymbol{\varepsilon}^{\mathrm{dev}}} \\ &= \sqrt{\frac{4}{9}\left(\varepsilon_{11}^2 + \varepsilon_{22}^2 + \varepsilon_{33}^2 - \varepsilon_{11}\varepsilon_{22} - \varepsilon_{22}\varepsilon_{33} - \varepsilon_{11}\varepsilon_{33}\right) + \frac{4}{3}\left(\varepsilon_{12}^2 + \varepsilon_{23}^2 + \varepsilon_{13}^2\right)} \quad .\end{aligned}$$

Glaskugelmischung

Bild 8.14 zeigt die Kontur des Bodenkörpers und die Scherfugen der numerischen Simulation mit den modifizierten Parametern der Glaskugelmischung. Die Ergebnisse haben nach den Anpassungen von h_S und n_H eine große Ähnlichkeit mit den Ergebnissen der experimentellen Untersuchungen aus Bild 8.6, die zum Vergleich eingezeichnet sind.

Die Horizontalkraft am Schieber ist in Bild 8.15 zu sehen. Für einen Vergleich mit den Modellversuchen ist das Ergebnis mit einem Faktor von 30 skaliert worden, da im numerischen Modell nur ein 1 cm breiter Streifen betrachtet wird. Die resultierenden Kräfte sind in einem ähnlichen Bereich wie die Modellversuche in Bild 8.5, aber es können einige deutliche Unterschiede erkannt werden:

Zum einen sind die Kräfte am Schieber in der Simulation kleiner und starten auf einem niedrigeren Niveau. In der Simulation werden Gewicht und Reibung des Schiebers auf den Führungsschienen nicht berücksichtigt, was ungefähr 70 N entspricht (siehe Tabelle 8.1).

Zum anderen zeigt der Verlauf der Horizontalkraft in Bild 8.15 einen deutlichen Sprung in der Belastungsphase bei ungefähr 140 mm. Dort verringert sich der horizontale Widerstand des Bodenkörpers, da ein Knick in der vorher geradlinigen Scherfuge hin zur Horizontalen entsteht, wie unten links in Bild 8.14 zu sehen ist. Ein Teil des Rauschens und kleinere Sprünge, wie bei ungefähr 60 mm in der Belastungsphase oder 40 mm, 95 mm und 150 mm in der Wiederbelastungsphase, lassen sich ebenfalls durch Änderungen am Verlauf der Scherfuge erklären.

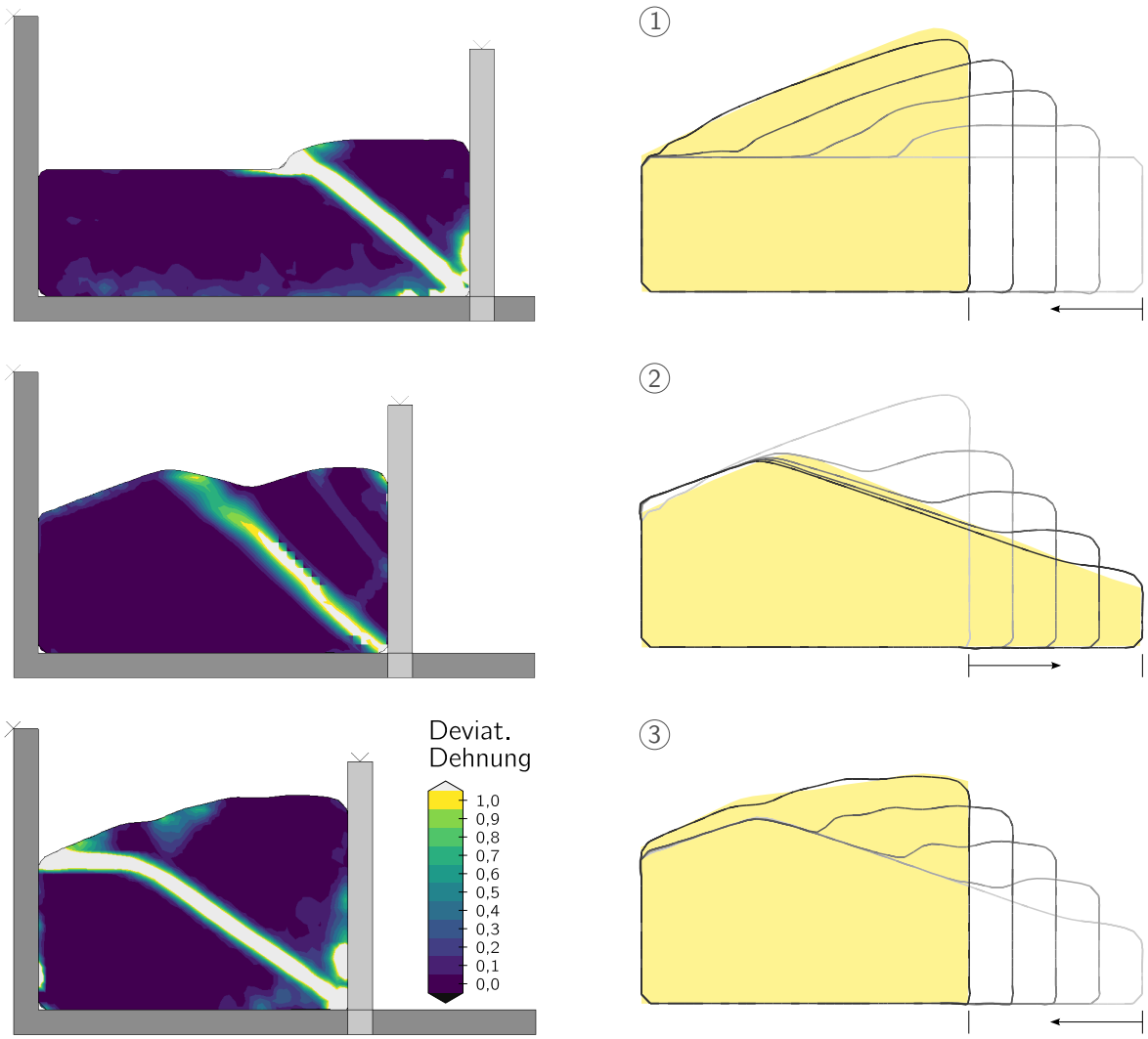


Bild 8.14: Invariante der deviatorischen Dehnung und Form des Bodenkörpers der numerischen Simulation mit modifizierten Parametern der Glaskugelmischung. Zum Vergleich ist auf den rechten Seiten in gelb jeweils die Form des Bodenkörpers der experimentellen Untersuchung am Ende der jeweiligen Belastungsphase eingezeichnet.

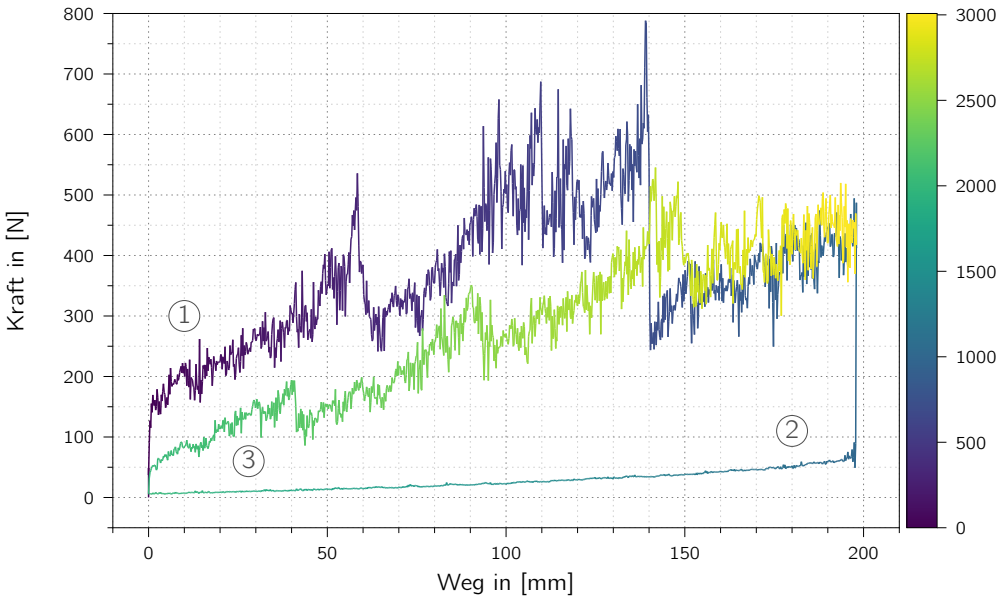


Bild 8.15: Horizontale Kraft am Schieber in der numerischen Simulation mit modifizierten Parametern der Glaskugelmischung. Die dargestellten Werte sind mit Faktor 30 multipliziert worden, da in der Simulation nur ein 1 cm breiter Streifen betrachtet wird. Die drei Belastungsphasen mit jeweils 1000 Schritten sind über die Farbgebung der Kurve symbolisiert.

Hamburger Sand

Die Ergebnisse der Simulation mit den modifizierten Parametern für Hamburger Sand sind in Bild 8.16 und 8.17 dargestellt. Im Gegensatz zur Glaskugelmischung ist generell eine geringere Übereinstimmung zu den experimentellen Untersuchungen erkennbar. So ist die freie Böschung in der Belastungsphase und vor allem am Ende der Wiederbelastungsphase der Simulation steiler als in den Experimenten.

Auch bei der Horizontalkraft am Schieber in Bild 8.17 sind deutliche Unterschiede zu sehen: Zum einen sind die Kräfte am Schieber in der Simulation kleiner, starten auf einem niedrigeren Niveau und sind von mehreren Sprüngen im Kraftverlauf geprägt, v. a. bei ungefähr 10 mm, 40 mm und 60 mm der Belastungsphase. Wie bei der Simulation der Glaskugelmischung werden Gewicht und Reibung des Schiebers auf den Führungsschienen nicht berücksichtigt. Zusätzlich hat auch der in den Modellversuchen am Schieber vorbeigedrückte Sand in der Simulation keinen Einfluss auf die Horizontalkraft. Einer der größten Unterschiede zu den experimentellen Untersuchungen mit Hamburger Sand ist, dass das Material in der Simulation vorwiegend nach oben weggedrückt und dabei stellenweise aufgelockert wird.

Bei diesen Versuchen ist der horizontale Widerstand des Bodenkörpers relativ gering, da sich in der Belastungsphase und der Wiederbelastungsphase frühzeitig eine zur Horizontalen gekrümmte Scherfuge einstellt (siehe Bild 8.16).

Für alle numerischen Simulationen in diesem Abschnitt muss beachtet werden, dass die dargestellten Ergebnisse abhängig von der Geschwindigkeit des Schiebers sind. Es sind Simulationen mit einer Schrittlänge zwischen 1 und 24 s durchgeführt worden, wobei das Verhalten nur im Bereich von 6 s bis 12 s reproduziert werden kann. Bei größeren Schrittlängen und somit kleineren Geschwindigkeiten stellt sich die Verformung des Bodenkörpers nicht am Schieber, sondern am linken Rand des Versuchsbehälters ein.

Umgekehrt führen größere Geschwindigkeiten des Schiebers dazu, dass sich der Bodenkörper aufgrund der Trägheit bei der ersten Belastungsumkehr vom Schieber ablöst. Überprüfungen mit der halben Elementlänge resultieren in einem vergleichbaren Verformungsverhalten für die Glaskugelmischung und leichte Abweichungen für Hamburger Sand.

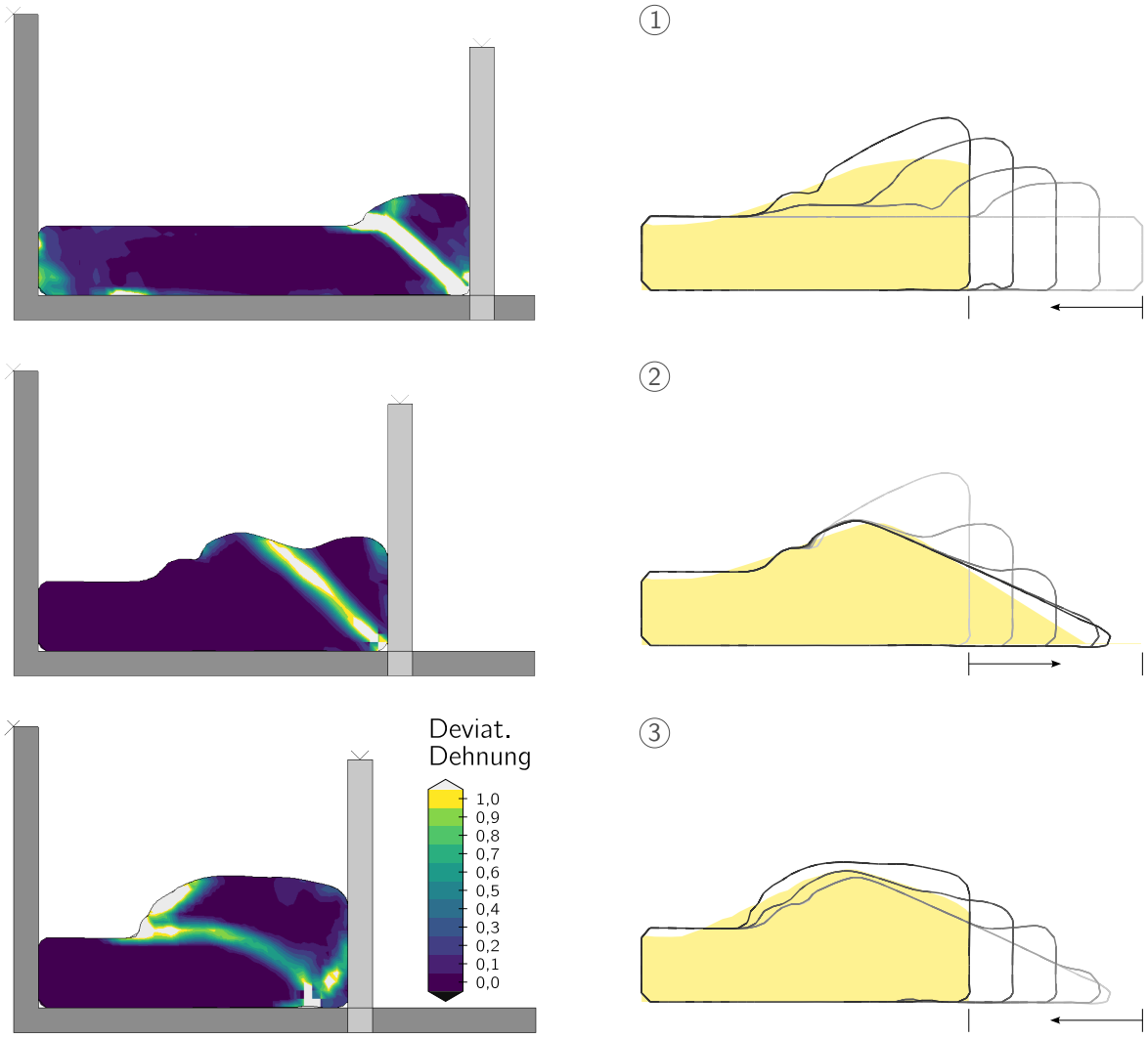


Bild 8.16: Invariante der deviatorischen Dehnung und Form des Bodenkörpers der numerischen Simulation mit modifizierten Parametern für Hamburger Sand. Zum Vergleich ist auf den rechten Seiten in gelb jeweils die Form des Bodenkörpers der experimentellen Untersuchung am Ende der jeweiligen Belastungsphase eingezeichnet.

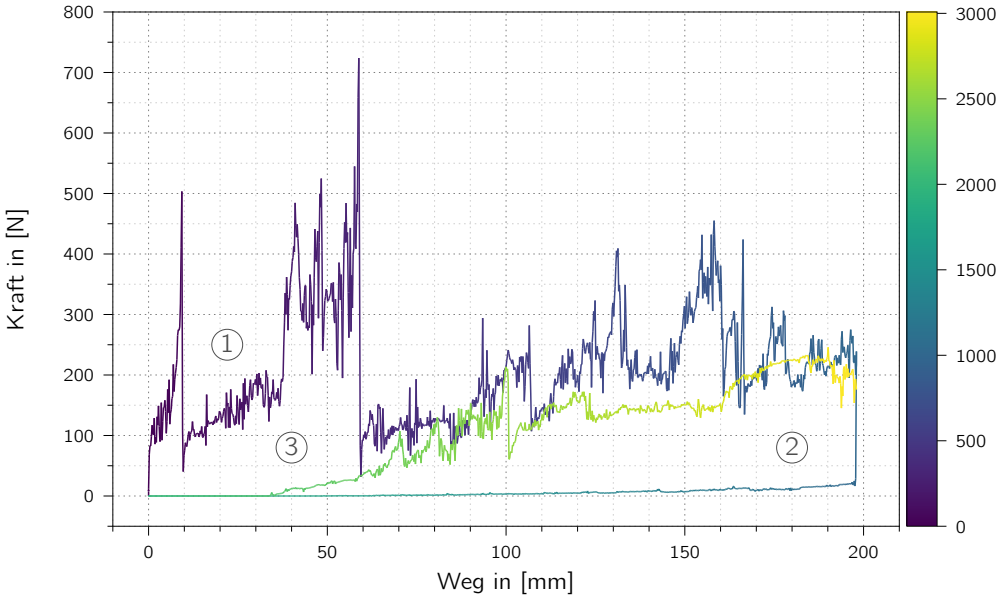


Bild 8.17: Horizontale Kraft am Schieber in der numerischen Simulation mit modifizierten Parametern für Hamburger Sand. Die dargestellten Werte sind mit Faktor 30 multipliziert worden, da in der Simulation nur ein 1 cm breiter Streifen betrachtet wird. Die drei Belastungsphasen mit jeweils 1000 Schritten sind über die Farbgebung der Kurve symbolisiert.

9 Zusammenfassung und Ausblick

Dealing with failure is easy: Work hard to improve.
Success is also easy to handle: You've solved the wrong problem.
Work hard to improve.

Perlis (1982), S. 12

Zusammenfassung

Im Rahmen dieser Arbeit ist die Containerstruktur xMat vorgestellt und deren Funktionsweise überprüft worden. Kapitel 4 hat aufgezeigt, wie die Benutzerroutine aufgebaut und die modulare Struktur realisiert worden ist. Zusammen mit Kapitel 5 ist demonstriert worden, wie die Stoffmodelle, Integrationsverfahren und Schnittstellen miteinander interagieren und erweitert werden können.

Die einzelnen Komponenten sind in Kapitel 7 mit Referenzdaten aus der Literatur verglichen worden. Es konnte gezeigt werden, dass mit der xMat Ergebnisse von Ödometer- und Triaxialversuchen reproduziert werden können. Auch eine vergleichbare Nutzung aus verschiedenen Programmen und Programmiersprachen führt jeweils auf das gleiche Ergebnis, was sowohl das Funktionieren, als auch die Unabhängigkeit der Schnittstellen demonstriert.

Wie Kapitel 7 aufzeigt, ist die xMat für einfache Untersuchungen einsetzbar, aber für FE-Analysen noch nicht ausreichend getestet.

Während der Bearbeitung sind neben diversen Skripten mehrere eigene Programme und Bibliotheken entstanden, die im jeweils passenden Abschnitt kurz vorgestellt worden sind: ElementSimulator[1], MPO[2], miniSoilLAB[3], abapys[4] und SimpleScriptGenerator[5].

Ausblick

Die experimentellen Untersuchungen in Kapitel 8 zeigen noch Verbesserungspotenzial bei der Durchführung. Für weitere Versuche ist eine Abdichtung zwischen Schieber und den umliegenden Wänden sowie der Bodenplatte überlegenswert. Außerdem wären auch zusätzliche Druckmessungen zwischen Schieber und Bodenkörper hilfreich, um daraus Informationen zum Belastungszustand abzuleiten.

[1] ElementSimulator ©D. Zobel: »https://github.com/d-zo/ElementSimulator
[2] MPO ©D. Zobel: »https://github.com/d-zo/MPO
[3] miniSoilLAB ©D. Zobel: »https://github.com/d-zo/miniSoilLAB
[4] abapys ©D. Zobel: »https://github.com/d-zo/abapys
[5] SimpleScriptGenerator ©D. Zobel: »https://github.com/d-zo/SimpleScriptGenerator

Auch an der xMat gibt es noch verschiedene Verbesserungsmöglichkeiten. Aktuell wird das Integrationsverfahren fest im Modul `General_Settings` vorgegeben. Mehr Flexibilität bietet die Übergabe einer Statusvariable zur Auswahl des Integrationsverfahrens. Das hat aber auch den Nachteil, dass eine Nummer statt einer Bezeichnung die Verständlichkeit und Anpassbarkeit negativ beeinflussen kann. Wie in der Benutzerroutine `ABQS02` ist die Angabe von Toleranzen sinnvoll und ggf. auch die Wahl der Fehlernorm bei der Integration. Eine vielversprechende Erweiterung wäre die automatische Erkennung, welches Integrationsverfahren sich für ein gewähltes Stoffmodell und übergebene Parameter am besten eignet.

Zusätzlich kann der Einsatz von weiteren, expliziten Integrationsverfahren, wie `odex` und `dop853`, geprüft werden (siehe Hairer, Nørsett u. a., 2008). Implizite Integrationsverfahren sind im Rahmen dieser Arbeit nicht untersucht worden und bieten weiteres Potenzial.

Im Rahmen der Arbeit sind vor allem einfache Überprüfungen durchgeführt worden, aber weitere Tests und der Einsatz in FE-Analysen sind notwendig, um die Tauglichkeit der xMat zu verbessern.

Außerdem sollte die derzeitige Umsetzung der `direct_variables` überdacht werden, um eine bessere Trennung von Stoffmodellen und Integrationsverfahren zu ermöglichen. Die aktuelle Implementierung von `PlaxisInformationPool` macht Anpassungen an Stoffmodellen an zwei Stellen erforderlich, statt an einer einzigen. Wie in Abschnitt 4.4 aufgezeigt gibt es Argumente für und gegen die aktuelle Wahl, was genauer untersucht werden sollte.

Die Definition der Schnittstellen `xmat_console` und `Dot_Values` ist frei gewählt worden und hat sich für einfache Versuche als ausreichend erwiesen. Komplexere Berechnungen erfordern weitere Anpassungen und auch eine Überarbeitung der Schnittstellen.

Literatur

Adams, J. C., W. S. Brainerd, R. S. Hendrickson, R. E. Maine, J. T. Martin und B. T. Smith (2009). *The Fortran 2003 Handbook. The Complete Syntax, Features and Procedures*. Springer. ISBN: 978-1-846-28378-9. DOI: »10.1007/978-1-84628-746-6.

Altenbach, H. (2018). *Kontinuumsmechanik. Einführung in die materialunabhängigen und materialabhängigen Gleichungen*. 4. Aufl. Springer. DOI: »10.1007/978-3-662-57504-8.

Anderson, E., Z. Bai, C. H. Bischof, S. Blackford, J. Demmel, J. J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney und D. C. Sorensen (1999). *LAPACK Users' Guide*. 3. Aufl. Society for Industrial und Applied Mathematics. ISBN: 0-898-71447-8. URL: »https://www.netlib.org/lapack/lug/.

Atkinson, J. H., D. Richardson und S. E. Stallebrass (1990). „Effect of recent stress history on the stiffness of overconsolidated soil". In: *Géotechnique* 40.53, S. 531–540. DOI: »10.1680/geot.1990.40.4.531.

Backus, G. (1970). „A Geometrical Picture of Anisotropic Elastic Tensors". In: *Reviews of Geophysics and Space Physics* 8.3, S. 633–671. DOI: »10.1029/RG008i003p00633.

Bardet, J. P. (1990). „Lode Dependences for Isotropic Pressure-Sensitive Elastoplastic Materials". In: *Journal of Applied Mechanics* 57, S. 498–506. DOI: »10.1115/1.2897051.

Batista, D., P. Royis und T. Doanh (2003). „Time-Integration of a Hypoplastic Constitutive Equation in Finite Element Modelling". In: *Mathematical and Computer Modelling* 37, S. 615–621. DOI: »10.1016/S0895-7177(03)00055-4.

Bauer, E. (1992). „Zum mechanischen Verhalten granularer Stoffe unter vorwiegend ödometrischer Beanspruchung". Dissertation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC036923656. Heft 130.

Bauer, E. (1996). „Calibration of a Comprehensive Hypoplastic Model for Granular Materials". In: *Soils and Foundations* 36.1, S. 13–26. DOI: »10.3208/sandf.36.13.

Benson, D. J. (1992). „Computational methods in Lagrangian and Eulerian hydrocodes". In: *Computer Methods in Applied Mechanics and Engineering* 99 (2–3), S. 235–394. DOI: »10.1016/0045-7825(92)90042-I.

Bentley Systems (2016). *User-Defined Soil Models*. Plaxis. URL: »https://communities.bentley.com/products/geotech-analysis/w/plaxis-soilvision-wiki/45468/plaxis-user-defined-soil-models.

Benz, T. (2007). „Small-Strain Stiffness of Soils and its Numerical Consequences“. Dissertation. Institut für Geotechnik, Stuttgart. URL: »https://www.igs.uni-stuttgart.de/institut/publikationen/mitteilungen/.

Benz, T. und K. Bergholz (2017). „Geotechnische Materialmodelle im Spannungsfeld zwischen Forschung und Praxis“. In: *BAW Mitteilungen* 101, S. 73–85. URL: »https://hdl.handle.net/20.500.11970/104393.

Bergholz, K. (2020). „An extended bounding surface model for the application to general stress paths in sand“. Dissertation. TU Dresden. URL: »https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-726001.

Bernal, J. D. (1964). „The Bakerian Lecture 1962. The Structure of Liquids“. In: *Proceedings of the Royal Society of London.* Mathematical and Physical Sciences 280.1382, S. 299–322. URL: »https://www.jstor.org/stable/2415872.

Bienen, B., S. Stanier, C. Vulpe und D. Mašín (2014). *Interface enabling constitutive models coded as user materials to be employed in explicit analysis.* Research report. Centre for Offshore Foundation Systems. URL: »https://soilmodels.com/download/bienen-b-stanier-s-vulpe-c-and-masin-d-2014-interface-enabling/.

Bode, M., W. Fellin, D. Mašín, G. Medicus und A. Ostermann (2020). „An intergranular strain concept for material models formulated as rate equations“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 44, S. 1003–1018. DOI: »10.1002/nag.3043.

Bornath, T. und G. Walter (2020). *Messunsicherheiten – Grundlagen. Für das Physikalische Praktikum.* Springer. ISBN: 978-3-658-29384-0. DOI: »10.1007/978-3-658-29385-7.

Butcher, J. C. (2016). *Numerical Methods for Ordinary Differential Equations.* 3. Aufl. Wiley. ISBN: 978-1-119-12150-3. DOI: »10.1002/9781119121534.

Cekerevac, C., S. Girardin, G. Klubertanz und L. Laloui (2006). „Calibration of an elasto-plastic constitutive model by a constrained optimisation procedure“. In: *Computers and Geotechnics* 33, S. 432–443. DOI: »10.1016/j.compgeo.2006.07.009.

Chambon, R., J. Desrues, W. Hammad und R. Charlier (1994). „CLoE, a new rate-type constitutive model for geomaterials theoretical basis and implementation“. In: *Numerical and Analytical Methods in Geomechanics* 18.4, S. 253–278. DOI: »10.1002/nag.1610180404.

Collatz, L. (1950). *Numerische Behandlung von Differentialgleichungen.* 2. Aufl. Springer. ISBN: 978-3-662-22249-2. DOI: »10.1007/978-3-662-22248-5.

Dafalias, Y. F. (1986). „Bounding Surface Plasticity. I: Mathematical Foundation and Hypoplasticity“. In: *Journal of Engineering Mechanics* 112.9, S. 966–987. DOI: »10.1061/(ASCE)0733-9399(1986)112:9(966).

Dafalias, Y. F. und M. T. Manzari (2004). „Simple Plasticity Sand Model Accounting for Fabric Change Effects“. In: *ASCE*. doi: »10.1061/(ASCE)0733-9399(2004)130%3A6(622).

Dassault Systèmes (2015). *Abaqus 6.14-2*. url: »http://www.3ds.com/support/certified-hardware/simulia-system-information/.

Dassault Systèmes (2020). *2020 SIMULIA User Assistance. Abaqus Online Documentation*. url: »https://help.3ds.com.

Desrues, J., E. Andò, P. Bésuelle, G. Viggiani, L. Debove, P. Charrier und J. B. Toni (2017). „Localisation Precursors in Geomaterials?“ In: *Bifurcation and Degradation of Geomaterials with Engineering Applications–Proceedings of the 11th International Workshop on Bifurcation and Degradation in Geomaterials*. Hrsg. von E. Papamichos, P. Papanastasiou, E. Pasternak und A. Dyskin. Geomechanics and Geoengineering. Springer, S. 3–10. isbn: 978-3-319-56396-1. doi: »10.1007/978-3-319-56397-8_1.

Desrues, J., E. Andò, F. A. Mevoli Ando, L. Debove und G. Viggiani (2018). „How does strain localise in standard triaxial tests on sand: Revisiting the mechanism 20 years on“. In: *Mechanics Research Communications* 92, S. 142–146. doi: »10.1016/j.mechrescom.2018.08.007.

Deutsche Gesellschaft für Geotechnik e.V. (2014). *Empfehlungen des Arbeitskreises Numerik in der Geotechnik*. Wiley. isbn: 978-3-433-03080-6. doi: »10.1002/9783433604489.

Dijkstra, E. W. (1982). *Selected writings on computing*. Springer. isbn: 978-1-461-25697-7. doi: »10.1007/978-1-4612-5695-3.

DIN 1319-3:1996. *Grundlagen der Meßtechnik – Auswertung von Messungen einer einzelnen Meßgröße – Meßunsicherheit*. Normenausschuss Bauwesen. Beuth Verlag. doi: »10.31030/7204542.

DIN 18126:2022. *Baugrund, Untersuchung von Bodenproben – Bestimmung der Dichte nichtbindiger Böden bei lockerster und dichtester Lagerung*. Normenausschuss Bauwesen. Beuth Verlag. doi: »10.31030/3371603.

DIN EN ISO 17892-3:2016. *Geotechnische Erkundung und Untersuchung, Laborversuche an Bodenproben – Teil 3: Bestimmung der Korndichte*. Normenausschuss Bauwesen. Beuth Verlag. doi: »10.31030/2337616.

DIN EN ISO 17892-4:2017. *Geotechnische Erkundung und Untersuchung, Laborversuche an Bodenproben – Teil 4: Bestimmung der Korngrößenverteilung*. Normenausschuss Bauwesen. Beuth Verlag. doi: »10.31030/2362539.

DIN EN ISO 17892-5:2017. *Geotechnische Erkundung und Untersuchung, Laborversuche an Bodenproben – Teil 5: Oedometerversuch mit stufenweiser Belastung*. Normenausschuss Bauwesen. Beuth Verlag. doi: »10.31030/2579350.

DIN EN ISO 17892-9:2018. *Geotechnische Erkundung und Untersuchung, Laborversuche an Bodenproben – Teil 9: Konsolidierte triaxiale Kompressionsversuche an wassergesättigten Böden*. Normenausschuss Bauwesen. Beuth Verlag. DOI: »10.31030/2791792.

Fellin, W., M. Mittendorfer und A. Ostermann (2009). „Adaptive integration of constitutive rate equations“. In: *Computers and Geotechnics* 36.5, S. 698–708. DOI: »10.1016/j.compgeo.2008.11.006.

Fellin, W. und A. Ostermann (2002). „Consistent tangent operators for constitutive rate equations“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 26 (12), S. 1213–1233. DOI: »10.1002/nag.242.

Fellin, W. und A. Ostermann (2013). „The critical state behaviour of barodesy compared with the Matsuoka-Nakai failure criterion“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 37.3, S. 299–308. DOI: »10.1002/nag.1111.

Fuentes Lacouture, W. M. (2014). „Contributions in Mechanical Modelling of Fill Materials“. Dissertation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC422654051. Heft 179.

Goldberg, D. (1991). „What Every Computer Scientist Should Know About Floating-Point Arithmetic“. In: *ACM Computing Surveys* 23.1, S. 5–48. DOI: »10.1145/103162.103163. URL: »https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html.

Goldscheider, M. (1976). „Grenzbedingung und Fliessregel von Sand“. In: *Mechanics Research Communications* 3.6, S. 463–468. DOI: »10.1016/0093-6413(76)90037-9.

Golub, G. H. und C. F. Van Loan (2013). *Matrix Computations*. 4. Aufl. Johns Hopkins University Press. ISBN: 978-1-421-40794-4. DOI: »10.56021/9781421407944.

Grabe, J. (2004). *Bodenmechanik und Grundbau*. 3. Aufl. Arbeitsbereich Geotechnik und Baubetrieb, TUHH. ISBN: 3-936-31008-4. URL: »https://www.tuhh.de/gbt/publikationen/veroeffentlichungsreihe.html.

Gudehus, G. (1996). „A Comprehensive Constitutive Equation for Granular Materials“. In: *Soils and Foundations* 36.1, S. 1–12. DOI: »10.3208/sandf.36.1.

Gudehus, G. und D. Mašín (2009). „Graphical representation of constitutive equations“. In: *Géotechnique* 59.2, S. 147–151. DOI: »10.1680/geot.2007.00155.

Hairer, E., C. Lubich und G. Wanner (2006). *Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations*. 2. Aufl. Bd. 31. Springer Series in Computational Mathematics. Springer. ISBN: 978-3-540-30663-4. DOI: »10.1007/3-540-30666-8.

Hairer, E., S. P. Nørsett und G. Wanner (2008). *Solving Ordinary Differential Equations I. Nonstiff Problems*. 2. Aufl. Bd. 8. Springer Series in Computational Mathematics. Springer. ISBN: 978-3-540-56670-0. DOI: »10.1007/978-3-540-78862-1.

Hairer, E. und G. Wanner (2002). *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*. 2. Aufl. Bd. 14. Springer Series in

Computational Mathematics. Springer. ISBN: 978-3-642-05220-0. DOI: »10.1007/978-3-642-05221-7.

Hajihassani, M., D. J. Armaghani und R. Kalatehjari (2018). „Applications of Particle Swarm Optimization in Geotechnical Engineering: A Comprehensive Review“. In: *Geotechnical and Geological Engineering* 36, S. 705–722. DOI: »10.1007/s10706-017-0356-z.

Halilovič, M., M. Vrh und B. Štok (2009). „NICE–An explicit numerical scheme for efficient integration of nonlinear constitutive equations“. In: *Mathematics and Computers in Simulation* 80.2. DOI: »10.1016/j.matcom.2009.06.030.

Hamming, R. W. (1997). *The Art of Doing Science and Engineering. Learing to Learn*. CRC Press. ISBN: 978-9-056-99501-0. DOI: »10.1201/9781482283198.

Heeres, O. M. und R. de Borst (2000). „Implicit Integration of Hypoplastic Models“. In: *European Congress on Computational Methods in Applied Sciences and Engineering*. Barcelona, Spanien. DOI: »10.1007/978-3-642-57018-6_22.

Helnwein, P. (2001). „Some remarks on the compressed matrix representation of symmetric second-order and fourth-order tensors“. In: *Computer methods in applied mechanics and engineering* 190, S. 2753–2770. DOI: »10.1016/S0045-7825(00)00263-2.

Helwany, S. (2007). *Applied Soil Mechanics. With Abaqus Applications*. 3. Aufl. Wiley. ISBN: 978-0-471-79107-2. DOI: »10.1002/9780470168097.

Herle, I. (1997). „Hypoplastizität und Granulometrie einfacher Korngerüste“. Dissertation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC063943395. Heft 142.

Herle, I. und G. Gudehus (Sep. 1999). „Determination of parameters of a hypoplastic constitutive model from properties of grain assemblies“. In: *Mechanics of Cohesive-Frictional Materials* 4.5, S. 461–486. DOI: »10.1002/(SICI)1099-1484(199909)4:5<461::AID-CFM71>3.0.CO;2-P.

Hettler, A. (2008). „Erddruck“. In: *Grundbau-Taschenbuch: Teil 1: Geotechnische Grundlagen*. Hrsg. von K. J. Witt. 7. Aufl. John Wiley & Sons, S. 289–395. DOI: »10.1002/9783433600221.ch6.

Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. 2. Aufl. Society for Industrial und Applied Mathematics. ISBN: 978-0-898-71521-7. DOI: »10.1137/1.9780898718027.

Hodge Jr., P. G. (1961). „On Isotropic Cartesion Tensors“. In: *The American Mathematical Monthly* 68.8, S. 793–795. DOI: »10.2307/2311997.

Hopkins, A. B. und F. H. Stillinger (2012). „Densest binary sphere packings“. In: *Physical Review* 85.021130, S. 1–19. DOI: »10.1103/PhysRevE.85.021130.

Hügel, H. M. (1995). „Prognose von Bodenverformungen“. Dissertation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC050769189. Heft 136.

IEEE Std 754:2019. *Standard for Radix-Independent Floating-Point Arithmetic*. Floating-Point Working Group. The Institute of Electrical und Electronics Engineers, Inc. DOI: »10.1109/IEEESTD.2019.8766229.

Itskov, M. (2019). *Tensor Algebra and Tensor Analysis for Engineers*. 5. Aufl. Mathematical Engineering. Springer. ISBN: 978-3-319-98805-4. DOI: »10.1007/978-3-319-98806-1.

JCGM 100:2008. *Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement*. BIPM. URL: »https://www.iso.org/sites/JCGM/GUM-introduction.htm.

Ji, X., X. Kong, D. Zou, Y. Sang, C. Zhou, J. Liu und J. Zhao (2021). „Measurement of membrane penetration in triaxial specimen through digital image correlation“. In: *Acta Geotechnica* 16, S. 1–19. DOI: »10.1007/s11440-020-00998-6.

Karcher, C. (2003). „Tagebaubedingte Deformationen im Lockergestein“. Dissertation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC105702560. Heft 160.

Knuth, D. E. (1998). *The Art of Computer Programming. Seminumerical Algorithms*. 3. Aufl. Bd. 2. Addison-Wesley. ISBN: 978-0-201-89684-8. URL: »https://www-cs-faculty.stanford.edu/~knuth/taocp.html.

Kolymbas, D. (1977). „A Rate-Dependent Constitutive Equation for Soils“. In: *Mechanics Research Communications* 4 (6), S. 367–372. DOI: »10.1016/0093-6413(77)90056-8.

Kolymbas, D. (1988). „Eine konstitutive Theorie für Böden und andere körnige Stoffe“. Habilitation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC014483947. Heft 109.

Kolymbas, D. (2011a). „Barodesy: a new hypoplastic approach“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 36, S. 1220–1240. DOI: »10.1002/nag.1051.

Kolymbas, D. (2011b). *Geotechnik: Bodenmechanik, Grundbau und Tunnelbau*. 3. Aufl. Springer. ISBN: 978-3-642-20481-4. DOI: »10.1007/978-3-642-20482-1.

Kolymbas, D. (2015). „Introduction to barodesy“. In: *Géotechnique* 65, S. 52–65. DOI: »10.1680/geot.14.P.151.

Kolymbas, D. (2021). „Barodesy with a new concept for critical void ratio“. In: *geotechnik*. DOI: »10.1002/gete.202000037.

Kolymbas, D. (2022). *A Primer on Theoretical Soil Mechanics*. Cambridge University Press. ISBN: 978-1-009-21034-8. DOI: »10.1017/9781009210348.

Kolymbas, D. und I. Herle (2008). „Stoffgesetze für Böden“. In: *Grundbau-Taschenbuch: Teil 1: Geotechnische Grundlagen*. Hrsg. von K. J. Witt. 7. Aufl. John Wiley & Sons, S. 243–287. DOI: »10.1002/9783433600221.ch5.

Kolymbas, D., I. Herle und P.-A. von Wolffersdorff (1995). „Hypoplastic Constitutive Equation with Internal Variables“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 19, S. 415–436. doi: »10.1002/nag.1610190604.

Kolymbas, D. und G. Medicus (2016). „Genealogy of hypoplasticity and barodesy“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 40.18, S. 2532–2550. doi: »10.1002/nag.2546.

Lewis, M. (2017). *The Undoing Project: A Friendship that Changed the World*. isbn: 978-0-393-35477-5. url: »https://wwnorton.com/books/9780393354775.

Lubich, C. und M. Roche (1990). „Rosenbrock methods for differential-algebraic systems with solution-dependent singular matrix multiplying the derivative“. In: *Computing* 43, S. 325–342. doi: »10.1007/BF02241653.

Machaček, J., P. Staubach, C. E. Grandas Tavera, T. Wichtmann und H. Zachert (2022). „On the automatic parameter calibration of a hypoplastic soil model“. In: *Acta Geotechnica* 17. doi: »10.1007/s11440-022-01669-4.

Markus, A. (2012). *Modern Fortran in Practice*. isbn: 978-1-107-60347-9. url: »www.cambridge.org/9781107603479.

Martin, R. C. (2009). *Clean Code. A Handbook of Agile Software Craftsmanship*. Prentice Hall. isbn: 978-0-132-35088-4. url: »https://www.pearson.com/us/higher-education/program/Martin-Clean-Code-A-Handbook-of-Agile-Software-Craftsmanship/PGM63937.html.

Mašín, D. (2005). „A hypoplastic constitutive model for clays“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 29, S. 311–336. doi: »10.1002/nag.416.

Mašín, D. (2012). „Hypoplastic Cam-clay model“. In: *Géotechnique* 62.6, S. 549–553. doi: »10.1680/geot.11.T.019.

Mašín, D. (2019). *Modelling of Soil Behaviour with Hypoplasticity. Another Approach to Soil Constitutive Modelling*. Hrsg. von W. Wu. Geomechanics and Geoengineering. Springer Verlag. isbn: 978-3-030-03975-2. doi: »10.1007/978-3-030-03976-9.

McConnell, S. (2004). *Code Complete. A practical handbook of software construction*. 2. Aufl. Microsoft Press. isbn: 978-0-735-61967-8. url: »https://www.microsoftpressstore.com/store/code-complete-9780735619678.

Medicus, G. (2014). „Barodesy and its Application for Clay“. Dissertation. Universität Innsbruck. url: »https://resolver.obvsg.at/urn:nbn:at:at-ubi:1-1346.

Medicus, G., W. Fellin und D. Kolymbas (2012). „Barodesy for clay“. In: *Géotechnique Letters* 2, S. 173–180. doi: »10.1680/geolett.12.00037.

Medicus, G., W. Fellin und F. Schranz (2018). „Konzepte der Barodesie“. In: *Bautechnik* 95 (9), S. 620–638. doi: »10.1002/bate.201800015.

Mendez, F. J., A. Pasculli, M. A. Mendez und N. Sciarra (2021). „Calibration of a hypoplastic model using genetic algorithms“. In: *Acta Geotechnica*. DOI: »10.1007/s11440-020-01135-z.

Metcalf, M., J. Reid und M. Cohen (2011). *Modern Fortran Explained*. 4. Aufl. Oxford University Press. ISBN: 978-0-199-60141-7. URL: »https://global.oup.com/academic/product/modern-fortran-explained-9780199601417.

Mittendorfer, M. (2010). „A modular finite element setting for nonlinear constitutive models: design and implementation“. Dissertation. Universität Innsbruck.

Miura, K., K. Maeda und S. Toki (1997). „Method of Measurement for the Angle of Repose of Sands“. In: *Soils and Foundations* 37.2, S. 89–96. DOI: »10.3208/sandf.37.2_89.

Möhrke, P. und B.-U. Runge (2020). *Arbeiten mit Messdaten. Eine praktische Kurzeinführung nach GUM*. Springer. ISBN: 978-3-662-60659-9. DOI: »10.1007/978-3-662-60660-5.

Nicholson, P. G., R. B. Seed und H. A. Anwar (1993). „Elimination of membrane compliance in undrained triaxial testing. I. Measurement and evaluation“. In: *Canadian Geotechnical Journal* 30, S. 727–738. DOI: »10.1139/t93-065.

Niemunis, A. (2003). „Extended hypoplastic models for soils“. Habilitation. Ruhr-Universität Bochum. URL: »https://www.bgu.ruhr-uni-bochum.de/bgu/publikationen/schriftenreihe.html.de.

Niemunis, A. und I. Herle (Okt. 1997). „Hypoplastic model for cohesionless soils with elastic strain range“. In: *Mechanics of Cohesive-Frictional Materials* 2.4, S. 279–299. DOI: »10.1002/(SICI)1099-1484(199710)2:4<279::AID-CFM29>3.0.CO;2-8.

Niemunis, A., T. Wichtmann und T. Triantafyllidis (2005). „A high-cycle accumulation model for sand“. In: *Computers and Geotechnics* 32.4, S. 245–263. DOI: »10.1016/j.compgeo.2005.03.002.

Noh, W. F. (1963). *CEL: A Time Dependent Two Space Dimensional, Coupled Eulerian-Lagrange Code*. Techn. Ber. Lawrence Radiation Laboratory. DOI: »10.2172/4621975.

Perlis, A. J. (1982). „Epigrams on Programming“. In: *ACM SIGPLAN Notices* 17.9, S. 7–13. DOI: »10.1145/947955.1083808.

Press, W. H., S. A. Teukolsky, W. T. Vetterling und B. P. Flannery (1997). *Numerical Recipes in Fortran 77. The Art of Scientific Computing*. 2. Aufl. ISBN: 978-0-521-43064-7. URL: »http://www.cambridge.org/9780521430647.

Qiu, G., S. Henke und J. Grabe (2011). „Application of a Coupled Eulerian–Lagrangian approach on geomechanical problems involving large deformations“. In: *Computers and Geotechnics* 38 (1), S. 30–39. DOI: »10.1016/j.compgeo.2010.09.002.

Roscoe, K. H. und J. B. Burland (1968). „On the generalized stress-strain behaviour of wet clay. Papers for a Conference held in Cambridge March 1986“.

In: *Engineering Plasticity*. Hrsg. von J. Heyman und F. A. Leckie. Cambridge University Press, S. 535–609. ISBN: 978-0-521-05254-2. URL: »https://www.cambridge.org/us/academic/subjects/engineering/solid-mechanics-and-materials/engineering-plasticity.

Rosenbrock, H. H. (1963). „Some general implicit processes for the numerical solution of differential equations“. In: *The Computer Journal* 5 (4), S. 329–330. DOI: »10.1093/comjnl/5.4.329.

Rouson, D., J. Xia und X. Xu (2011). *Scientific Software Design. The Object-Oriented Way*. Cambridge University Press. URL: »www.cambridge.org/9780521888134.

Sadd, M. H. (2014). *Elasticity - Theory, Applications, and Numerics*. 3. Aufl. Academic Press. ISBN: 978-0-124-08136-9. DOI: »10.1016/C2012-0-06981-5.

Schanz, T., P. A. Vermeer und P. G. Bonnier (1999). „The hardening soil model: Formulation and verification“. In: *Beyond 2000 in Computational Geotechnics*. Hrsg. von R. B. J. Brinkgreve. Taylor & Francis, S. 281–296. DOI: »10.1201/9781315138206-27.

Schneider-Muntau, B., G. Medicus, J. Desrues, E. Andò und G. Viggiani (2020). „Investigation of Uncertainty in Strength Parameter Identification“. In: *Challenges and Innovations in Geomechanics–Proceedings of the 16th International Conference of IACMAG*. Hrsg. von M. Barla, A. Di Donna und D. Sterpi. Bd. 125. Lecture Notes in Civil Engineering. Springer, S. 277–284. DOI: »10.1007/978-3-030-64514-4_23.

Schranz, F. (2018). „Applications and developments of Barodesy“. Dissertation. Universität Innsbruck. URL: »https://resolver.obvsg.at/urn:nbn:at:at-ubi:1-26846.

Scott, G. D. (1960). „Packing of Spheres: Packing of Equal Spheres“. In: *Nature* 188, S. 908–909. DOI: »10.1038/188908a0.

Severance, C. und K. Dowd (2010). *High Performance Computing*. Connexions. URL: »https://open.umn.edu/opentextbooks/textbooks/560.

Simo, J. C. und T. J. R. Hughes (1998). *Computational Inelasticity*. Hrsg. von J. E. Marsden, L. Sirovich und S. Wiggins. Bd. 7. Interdisciplinary Applied Mathematics. Springer. ISBN: 0-387-97520-9. DOI: »10.1007/b98904.

Tamagnini, C., G. Viggiani und R. Chambon (2000). „A review of two different approaches to hypoplasticity“. In: *Constitutive Modelling of Granular Materials*. Hrsg. von D. Kolymbas. Springer Verlag, S. 107–145. DOI: »10.1007/978-3-642-57018-6_5.

Tamagnini, C., G. Viggiani, R. Chambon und J. Desrues (2000). „Evaluation of different strategies for the integration of hypoplastic constitutive equations: Application to the CLoE model“. In: *Mechanics of Cohesive-Frictional Materials* 5, S. 263–289. DOI: »10.1002/(SICI)1099-1484(200005)5:4%3C263::AID-CFM93%3E3.0.CO;2-Q.

Testo Industrial Services GmbH (2020). *Messunsicherheitsfibel. Praxisgerechte Bestimmung von Messunsicherheiten nach GUM*. URL: »https://www.testotis.de/know-how/fibeln-lexika/.

The MathWorks, Inc. (2021). *MATLAB Online documentation*. URL: »https://www.mathworks.com/help/matlab/index.html.

Thielicke, W. und E. J. Stamhuis (2014). „PIVlab. Towards User-Friendly, Affordable and Accurate Digital Particle Image Velocimetry in MATLAB“. In: *Journal of Open Research Software* 2.1. DOI: »10.5334/jors.bl. URL: »https://pivlab.blogspot.com/.

Truesdell, C. (1984). *An Idiot's Fugitive Essays on Science. Methods, Criticism, Training, Circumstances*. Springer. ISBN: 978-1-461-38187-7. DOI: »10.1007/978-1-4613-8185-3.

Truesdell, C. und W. Noll (2004). *The Non-Linear Field Theories of Mechanics*. 3. Aufl. Srpinger. ISBN: 978-3-662-10388-3. DOI: »10.1007/978-3-662-10388-3.

von Wolffersdorff, P.-A. (1996). „A hypoplastic relation for granular materials with a predefined limit state surface“. In: *Mechanics of Cohesive-Frictional Materials* 1.3, S. 251–271. DOI: »10.1002/(SICI)1099-1484(199607)1:3<251::AID-CFM13>3.0.CO;2-3.

Vrh, M., M. Halilovič und B. Štok (2010). „Improved explicit integration in plasticity“. In: *International Journal for Numerical Methods in Engineering* 81.7, S. 910–938. DOI: »10.1002/nme.2737.

Wagner, W. und A. Pruss (2002). „The IAPWS Formulation 1995 for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use“. In: *Journal of Physical and Chemical Reference Data* 31.2, S. 387–535. DOI: »10.1063/1.1461829.

Walz, B. (2006). „Möglichkeiten und Grenzen bodenmechanischer 1g-Modellversuche“. In: *Entwicklungen in der Bodenmechanik, Bodendynamik und Geotechnik. Festschrift zum 60. Geburtstag von Univ.-Professor Dr.-Ing. habil. Stavros A. Savidis*. Hrsg. von F. Rackwitz. Springer, S. 63–78. ISBN: 978-3-540-27425-4. DOI: »10.1007/3-540-27438-3_5.

Weifner, T. und D. Kolymbas (2007). „A hypoplastic model for clay and sand“. In: *Acta Geotechnica* 2. DOI: »10.1007/s11440-007-0031-2.

Wichtmann, T. (2016). „Soil behaviour under cyclic loading – experimental observations, constitutive description and applications“. Habilitation. Universität Fridericiana Karlsruhe. URL: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC480167184. Heft 181.

Wichtmann, T., W. Fuentes und T. Triantafyllidis (2019). „Inspection of three sophisticated constitutive models based on monotonic and cyclic tests on fine sand: Hypoplasticity vs. Sanisand vs. ISA“. In: *Soil Dynamics and Earthquake Engineering* 124, S. 172–183. DOI: »10.1016/j.soildyn.2019.05.001.

Wichtmann, T. und T. Triantafyllidis (2016). „An experimental data base for the development, calibration and verification of constitutive models for sand with focus to cyclic loading. Part I: tests with monotonic loading and stress cycles“. In: *Acta Geotechnica* 11.4, S. 739–761. doi: »10.1007/s11440-015-0402-z.

Wilson, G., D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White und P. Wilson (2014). „Best Practices for Scientific Computing“. In: *PLOS Computational Biology* 12 (1). doi: »10.1371/journal.pbio.1001745.

Wood, D. M. (1990). *Soil behaviour and critical state soil mechanics*. Cambridge University Press. isbn: 978-0-521-33249-1. doi: »10.1017/CBO9781139878272.

Wu, W. (1992). „Hypoplastizität als mathematisches Modell zum mechanischen Verhalten granularer Stoffe“. Dissertation. Universität Fridericiana Karlsruhe. url: »https://primo.bibliothek.kit.edu/permalink/f/smllk5/KITSRC031680062. Heft 129.

Wu, W. und E. Bauer (1994). „A Simple Hypoplastic Constitutive Model For Sand“. In: *International Journal for Numerical and Analytical Methods in Geomechanics* 18 (12), S. 833–862. doi: »10.1002/nag.1610181203.

Wu, W., J. Lin und X. Wang (2017). „A basic hypoplastic constitutive model for sand“. In: *Acta Geotechnica* 12, S. 1373–1382. doi: »10.1007/s11440-017-0550-4.

Zienkiewicz, O. C., R. L. Taylor und D. D. Fox (2014). *The Finite Element Method. For Solid and Structural Mechanics*. 7. Aufl. Elsevier. isbn: 978-1-85617-634-7. doi: »10.1016/C2009-0-26332-X.

Häufig verwendete Symbole

Skalare, Vektoren, Matrizen und S4-Tensoren

c	[kPa]	Kohäsion	23
C_1, C_2, C_3, C_4	[-]	Materialparameter (`Hypo-Wu92`)	26
c_2, c_3, c_4	[-]	Exponent (Materialparameter `Baro-Ko15` und `Baro-Ko21`)	34
c_5	[-]	Faktor und Exponent in den barodetischen Stoffmodellen	34
c_6, c_7	[-]	Term (Materialparameter `Baro-Sc18`)	35
c_e	[-]	Verhältnis Porenzahlen (Materialparameter `Baro-Sc18`)	35
C_{cC}, C_{cF}	[-]	Kompressionsbeiwerte	31
d_{ini}	[mm]	Anfangsdurchmesser	13
D_{ref}	[-]	Referenzdehnungsrate	23
d_{20}	[-]	Kleinster Siebdurchmesser mit 20 % der Masse Durchgang	122
dt	[s]	Zeitschritt	19
E	[kPa]	Elastizitätsmodul	23
e	[-]	Porenzahl	9
e_{ini}	[-]	Anfangsporenzahl	52
$e_{\mathrm{min}}, e_{\mathrm{max}}$	[-]	minimale und maximale Porenzahl	10
e_{c0}	[-]	Porenzahl im kritischen Zustand	28
e_{d0}	[-]	Porenzahl bei dichtester Lagerung	28
e_{i0}	[-]	Porenzahl bei lockerster isotroper Lagerung	28
F	[kN]	Axialkraft	13
f	[-]	Faktor in den barodetischen Stoffmodellen	33
$\hat{f}$	[-]	relativer Fehler des Integrationsschritts	40
$f_{\mathrm{min}}, f_{\mathrm{max}}$	[-]	min. und max. Faktor bei der Anpassung der Schrittweite	40
f_{MC}	[-]	Grenzbedingung nach Mohr-Coulomb	23
f_{MN}	[-]	Grenzbedingung nach Matsuoka-Nakai	23
f_b, f_e	[-]	Faktoren im Stoffmodell `Hypo-VW96`	28
f_d	[-]	Faktor in der hypoplastischen Modellgleichung	26
g	[-]	Faktor in den barodetischen Stoffmodellen	33
h	[-]	Faktor in den barodetischen Stoffmodellen	33
h, h_{sub}	[s]	Zeitinkrement und Subinkrement	37
h_{ini}	[mm]	Anfangshöhe	13
h_s	[kPa]	Granulathärte (Materialparameter `Hypo-VW96`)	28
i, j, k, l	[-]	Laufindizes	6

I_1, I_2, I_3	[-]	Invarianten (einer Matrix)	8
J_1, J_2, J_3	[-]	Invarianten des Deviators (einer Matrix)	21
k_1, k_2	[-]	Term (Materialparameter `Baro-Ko21`)	36
K_r	[-]	Referenzsteifigkeit (Materialparameter `Baro-Sc18`)	35
m	[g]	Gewicht	9
m_R	[-]	Faktor für umgekehrte Lastrichtung (Materialparameter IG)	30
m_S	[g]	Trockenmasse	9
m_T	[-]	Faktor für orthogonale Lastrichtung (Materialparameter IG)	30
n_H	[-]	Exponent (Materialparameter `Hypo-VW96`)	28
p	[-]	Ordnung eines Integrationsverfahrens	38
p	[kPa]	mittlerer Druck	22
q	[kPa]	Deviatorspannung	22
$R_{\max}$	[-]	Elastischer Bereich (Materialparameter IG)	30
S	[-]	Sicherheitsfaktor bei der Anpassung der Schrittweite	40
s	[-]	Stufe eines Integrationsverfahrens	39
S_M	[-]	Normalisierte Änderungsrate (Membrandurchdringungseffekt)	122
t_A, t_R	[-]	Absolute und relative Toleranz in einem Integrationsschritt	40
u	[-]	Standardunsicherheit	9
u	[kPa]	Porenwasserdruck	11
V	[cm^3]	Volumen	9
v_{Schieb}	[mm/min]	Geschwindigkeit des Schiebers	168
V_P	[cm^3]	Volumen des Porenraums	9
V_S	[cm^3]	Feststoffvolumen	9
$y, \hat{y}$	[-]	Funktionswert und Abschätzung von ebenjenem	37
α_H	[-]	Pyknotropischer Exponent (Materialparameter `Hypo-VW96`)	28
β_H	[-]	Barotropischer Exponent (Materialparameter `Hypo-VW96`)	28
β_R	[-]	Exponent (Materialparameter IG)	30
χ	[-]	Exponent (Materialparameter IG)	30
δ_{ij}	[-]	Kronecker-Delta-Symbol	7
δV_m	[-]	Membrananschmiegungsfaktor	121
ΔV_{Rp}	[mm^3]	Volumenanteil der Randporen	122
κ	[-]	Verhältnis Steifigkeiten (Materialparameter `Baro-Sc18`)	35
κ_1, κ_2	[-]	Faktor (Materialparameter `Baro-Ko21`)	36
λ	[-]	Eigenwert (einer Matrix)	8
ν	[-]	Querdehnzahl	23
ϕ	[-]	Packungsdichte einer Kugelmischung	121
ψ	[Grad]	Dilatanzwinkel	23
ρ	[g/cm^3]	Dichte	10
ρ^χ	[-]	Interpolationsterm intergranulare Dehnung	29
ρ_d	[g/cm^3]	Trockendichte	10
ρ_S	[g/cm^3]	Korndichte	10
σ	[kPa]	(eindimensionale) Spannung	11

σ_r	[kPa]	Zelldruck, radiale Spannung	13
θ, θ^*	[Grad]	Lode-Winkel	22
ε	[-]	(eindimensionale) Dehnung	19
ε_q	[-]	Invariante der deviatorischen Dehnung	182
ε_V	[-]	Volumetrische Dehnung	121
φ_c	[Grad]	kritischer Reibungswinkel	23
ϑ_a, ϑ_p	[Grad]	aktiver und passiver Gleitwinkel	176
ξ	[-]	Exponent (Materialparameter `Baro-Sc18`)	35
μ	[-]	Mittelwert	9
σ	[-]	Standardabweichung	9
$\mathbf{D}$	[-]	Dehnungsrate	19
$\mathbf{F}$	[-]	Deformationsgradient	18
$\mathbf{h}$	[-]	intergranulare Dehnung (IG)	29
$\mathbf{I}$	[-]	Einheitsmatrix	7
$\mathbf{N}$	[kPa]	Teil des hypoplastischen Materialtensors	26
${}^{I}\mathbf{P}, {}^{II}\mathbf{P}$	[kPa]	Erster und zweiter Piola-Kirchhoff-Spannungstensor	20
$\mathbf{R}$	[-]	$\mathbf{R}$-Funktion der Barodesie	33
$\mathbf{T}$	[kPa]	allgemeiner Spannungszustand	11
$\mathbf{T}^K$	[kPa]	Kirchhoff-Spannungstensor	20
$\mathring{\mathbf{T}}, \mathring{\mathbf{T}}^G, \mathring{\mathbf{T}}^J$	[kPa]	objektive Spannungsraten	20
$\mathbf{W}$	[-]	Drehmatrix	19
$\boldsymbol{\varepsilon}$	[-]	Dehnungstensor	19
$\mathcal{C}$	[kPa]	Allgemeiner Materialtensor	22
$\mathcal{C}^{\mathrm{el}}$	[kPa]	elastischer Materialtensor	23
$\mathcal{I}^{\mathrm{id}}, \mathcal{I}^{\mathrm{sym}}, \mathcal{I}^{\mathrm{tr}}$	[-]	Spezielle S4-Tensoren zur Abbildung von Matrizen	8
$\mathcal{L}$	[kPa]	Teil des hypoplastischen Materialtensors	26
$\mathcal{M}$	[kPa]	hypoplastischer Materialtensor	26
$\mathcal{J}$	[kPa]	Jacobi-Matrix (tangentialer Steifigkeitstensor)	46

Operatoren und sonstige Symbole

:	Doppelte Kontraktion	6
$\otimes$	Dyadisches Produkt	6
$\hat{\square}$	Dimensionslose Variante (einer Matrix)	7
$\det(\square)$	Determinante (einer Matrix)	7
$\mathrm{dev}(\square)$	Deviator (einer Matrix)	7
$\ln(\square)$	Natürlicher Logarithmus	6
$\log_{10}(\square)$	Dekadischer Logarithmus	6
$e^{\square}$	Exponentialfunktion für Skalare oder Matrizen	8
$\vec{\square}$	Richtung (einer Matrix)	7
$\mathrm{tr}(\square)$	Spur (einer Matrix)	7

Anhang

Der Anhang umfasst die folgenden Anlagen:

A Mögliche Visualisierung von S4-Tensoren und Rechenoperationen,

B Sensitivitätsstudie,

C Explizite Runge-Kutta-Methoden,

D Dokumentation der xMat-Funktionalitäten,

E Materialparameter und Statusvariablen der Benutzerroutinen,

F Schnittstellen zur Nutzung der xMat,

G Zur Bestimmung der Unsicherheit von Laborversuchen,

H Eigene Abschätzungen,

I Daten der Laborversuche und

J xMat Referenz.

A Mögliche Visualisierung von S4-Tensoren und Rechenoperationen

Die folgenden Überlegungen und Visualisierungen sind im Allgemeinen nur für S4-Tensoren gültig, d. h. Tensoren vierter Ordnung mit je drei Dimensionen und einer Standardbasis im euklidischen Vektorraum.

Für die Darstellung von $\mathcal{T}_{\mathrm{MAT66}}$ der folgenden Übersichten gelten die in Kapitel 4 beschriebenen Annahmen, d. h. die Komponenten ergeben sich zu

$$\begin{pmatrix}
T_{1111} & T_{1122} & T_{1133} & \frac{T_{1112}+T_{1121}}{2} & \frac{T_{1123}+T_{1132}}{2} & \frac{T_{1113}+T_{1131}}{2} \\
T_{2211} & T_{2222} & T_{2233} & \frac{T_{2212}+T_{2221}}{2} & \frac{T_{2223}+T_{2232}}{2} & \frac{T_{2213}+T_{2231}}{2} \\
T_{3311} & T_{3322} & T_{3333} & \frac{T_{3312}+T_{3321}}{2} & \frac{T_{3323}+T_{3332}}{2} & \frac{T_{3313}+T_{3331}}{2} \\
\frac{T_{1211}+T_{2111}}{2} & \frac{T_{1222}+T_{2122}}{2} & \frac{T_{1233}+T_{2133}}{2} & \frac{T_{1212}+T_{1221}+T_{2112}+T_{2121}}{4} & \frac{T_{1223}+T_{1232}+T_{2123}+T_{2132}}{4} & \frac{T_{1213}+T_{1231}+T_{2113}+T_{2131}}{4} \\
\frac{T_{2311}+T_{3211}}{2} & \frac{T_{2322}+T_{3222}}{2} & \frac{T_{2333}+T_{3233}}{2} & \frac{T_{2312}+T_{2321}+T_{3212}+T_{3221}}{4} & \frac{T_{2323}+T_{2332}+T_{3223}+T_{3232}}{4} & \frac{T_{2313}+T_{2331}+T_{3213}+T_{3231}}{4} \\
\frac{T_{1311}+T_{3111}}{2} & \frac{T_{1322}+T_{3122}}{2} & \frac{T_{1333}+T_{3133}}{2} & \frac{T_{1312}+T_{1321}+T_{3112}+T_{3121}}{4} & \frac{T_{1323}+T_{1332}+T_{3123}+T_{3132}}{4} & \frac{T_{1313}+T_{1331}+T_{3113}+T_{3131}}{4}
\end{pmatrix}$$

Diese Matrixrepräsentation mit 6×6 Einträgen unterscheidet sich von der in der Kontinuumsmechanik üblicheren Notation, in der die Einträge in der vierten, fünften und sechsten Spalte mit einem Faktor zwei skaliert sind. Der Faktor ist nötig, damit die Multiplikation der Matrixrepräsentation mit den Schubspannungen τ_{ij} auf die Scherungen $\gamma_{ij} = \varepsilon_{ij} + \varepsilon_{ji}$ führt, während hier im Allgemeinen ε_{ij} bzw. $(\varepsilon_{ij} + \varepsilon_{ji})/2$ für die Scherungen verwendet wird.

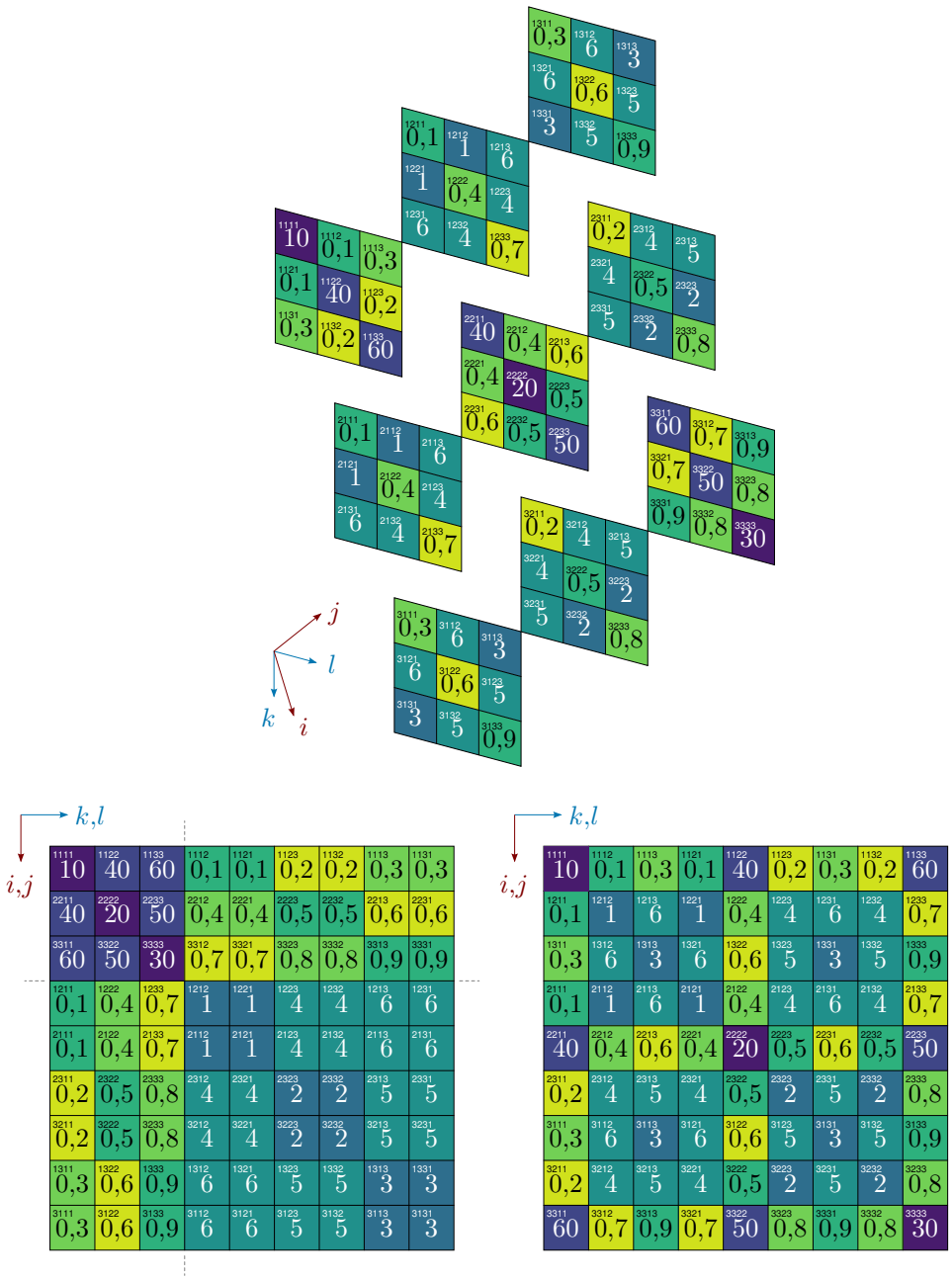
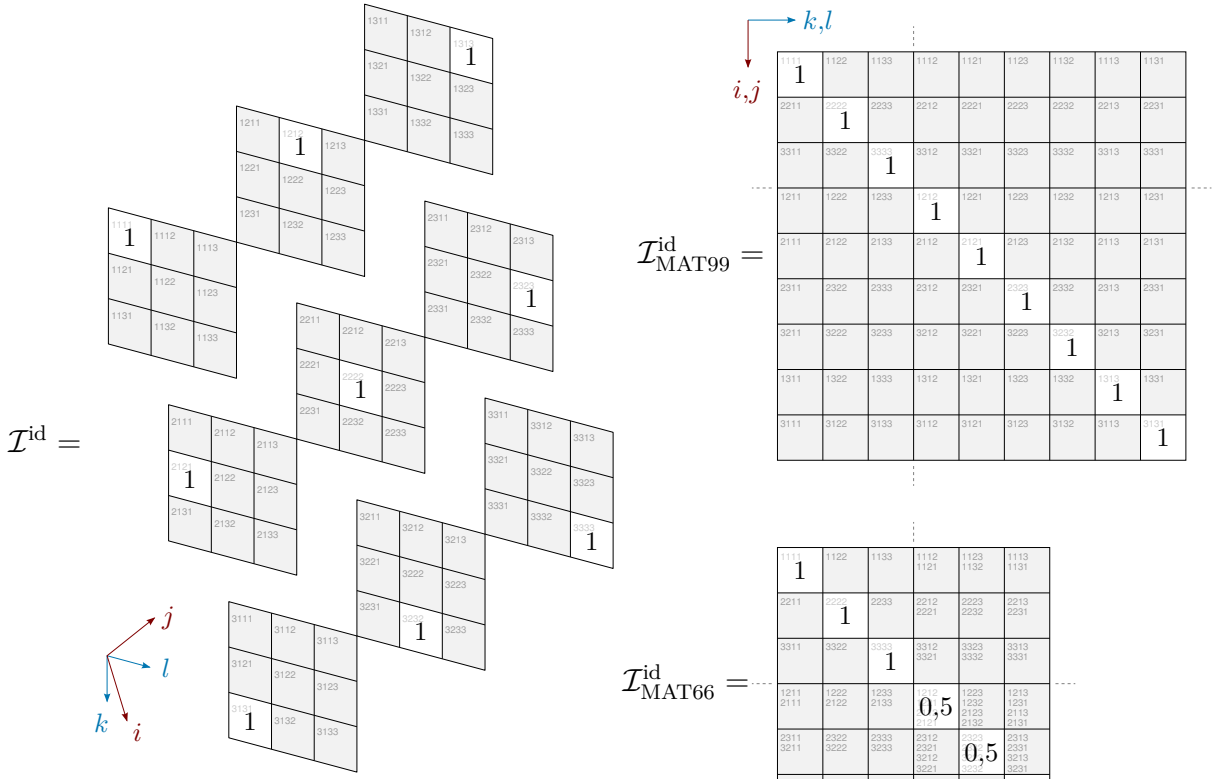


Bild A.1: Mögliche Visualisierung eines total symmetrischen S4-Tensors in Form von Matrizen. Während die Darstellung im Allgemeinen beliebig ist, ist eine der beiden Repräsentationen in der unteren Reihe häufig anzutreffen. Links unten ist eine Sortierung in Haupt- und Nebenkomponenten dargestellt, um eingeschränkte Matrixoperationen mit der zweidimensionalen Tensorrepräsentation `MAT99` durchführen zu können. Die Darstellung rechts unten entspricht einem S4-Tensor im Speicher eines Computers, bei dem jeweils $i \cdot j$-Komponenten pro Spalte ausgegeben werden. Zahlen und Farben sind frei gewählt, um die unterschiedlichen Strukturen und mögliche Symmetrien aufzuzeigen. Während total symmetrische S4-Tensoren, wie dargestellt, eine symmetrische Matrixdarstellung haben (siehe bspw. Helnwein, 2001, S. 2760), ist die Matrixrepräsentation von S4-Tensoren im Allgemeinen nicht symmetrisch.

Tabelle A.1: Gleiche Tensoreinträge aufgrund von Symmetrien anhand der Indizes der Einträge. Vollständig symmetrische Tensoren „total symmetry“ haben maximal 21 unterschiedliche Einträge (hier durch 21 Zeilen dargestellt). Innerhalb einer Zeile sind die dazugehörigen Tensoreinträge bei vollständig symmetrischen Tensoren identisch (bspw. $\mathcal{T}_{1112} = \mathcal{T}_{1121} = \mathcal{T}_{1211} = \mathcal{T}_{2111}$). Liegt nur eine „minor symmetry“ vor, so sind die Spalten 1-4 und die Spalten 1^*-4^* als getrennte Blöcke zu sehen (max. 36 unterschiedliche Einträge und bspw. $\mathcal{T}_{1112} = \mathcal{T}_{1121}$, aber im Allgemeinen nicht $\mathcal{T}_{1112} = \mathcal{T}_{1211}$). Bei einer „major symmetry“ hat jeder Wert höchstens einen Symmetriepartner (Spalte mit und ohne Stern in der gleichen Zeile, also bspw. $\mathcal{T}_{1112} = \mathcal{T}_{1211}$) und es gibt maximal 45 unterschiedliche Einträge im Tensor.

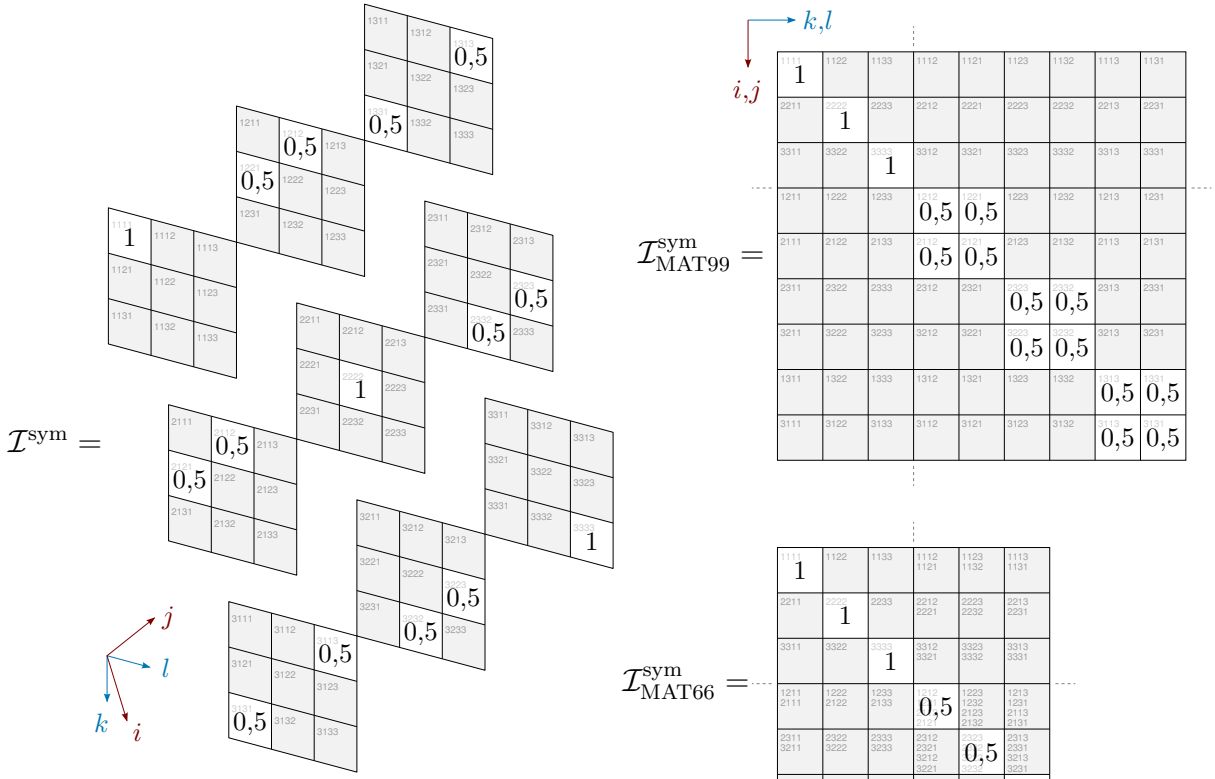
Schema	1	2	3	4	1^*	2^*	3^*	4^*
a-a-a-a	1111							
	2222							
	3333							
a-a-b-b (1)	1122				2211			
	2233				3322			
	3311				1133			
a-a-a-b	1112	1121			1211	2111		
	1113	1131			1311	3111		
	2221	2212			2122	1222		
	2223	2232			2322	3222		
	3331	3313			3133	1333		
	3332	3323			3233	2333		
a-a-b-c	1123	1132			2311	3211		
	2231	2213			3122	1322		
	3312	3321			1233	2133		
a-a-b-b (2)	1221	2112	1212	2121				
	2332	3223	2323	3232				
	3113	1331	3131	1313				
a-b-a-c	1213	2113	1231	2131	1312	1321	3112	3121
	2321	3221	2312	3212	2123	2132	1223	1232
	3132	1332	3123	1323	3231	3213	2331	2313



$$I^{\mathrm{id}}_{ijkl} = \delta_{ik} \otimes \delta_{jl}$$

$$\mathcal{I}^{\mathrm{id}} : \mathbf{A} = \mathbf{A}$$

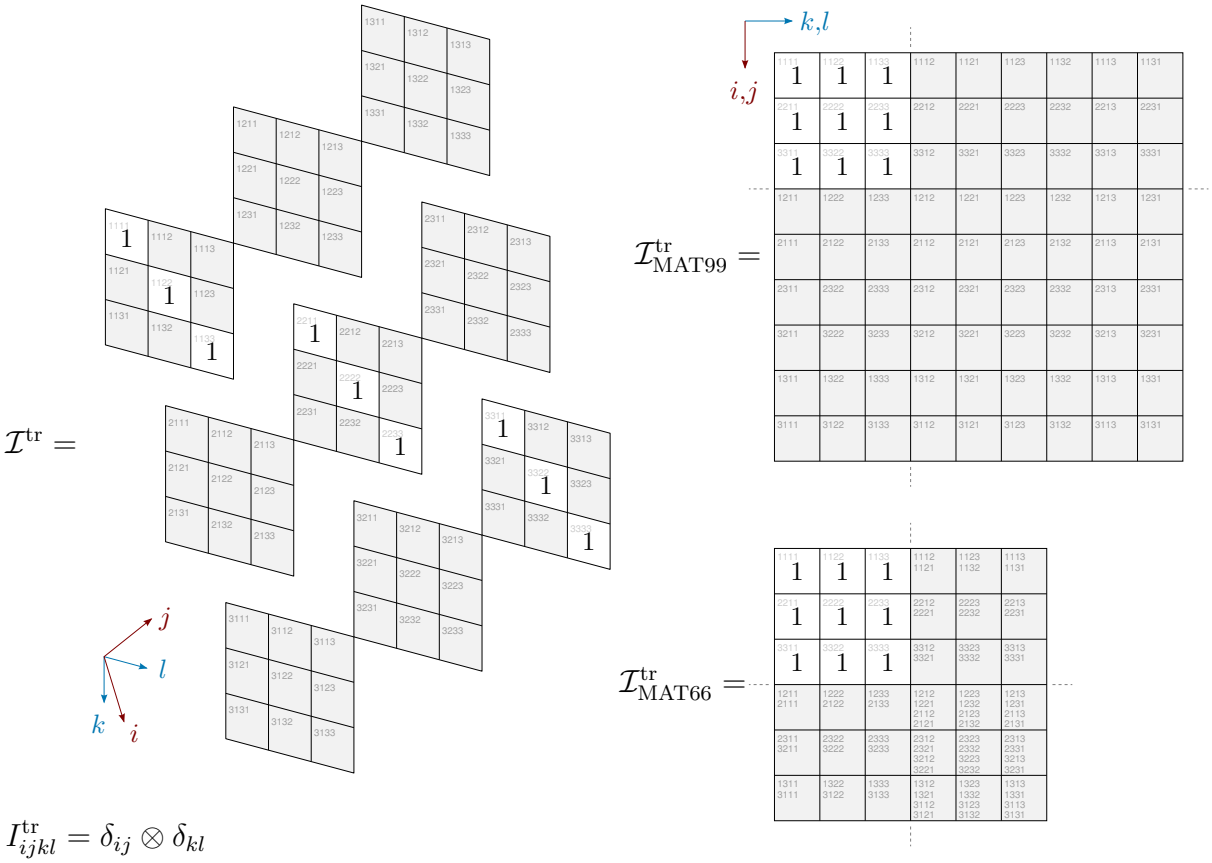
Bild A.2: Einheitstensor vierter Stufe für identische Abbildungen.



$$I^{\text{sym}}_{ijkl} = \frac{1}{2}\left(\delta_{ik} \otimes \delta_{jl} + \delta_{il} \otimes \delta_{jk}\right)$$

$$\mathcal{I}^{\text{sym}} : \mathbf{A} = \frac{1}{2}\left(\mathbf{A} + \mathbf{A}^T\right) = \text{sym}(\mathbf{A})$$

Bild A.3: Einheitstensor vierter Stufe für symmetrische Abbildungen.



$$I^{\mathrm{tr}}_{ijkl} = \delta_{ij} \otimes \delta_{kl}$$

$$\mathcal{I}^{\mathrm{tr}} : \mathbf{A} = \mathbf{B} \quad \text{mit} \quad B_{ij} = \mathrm{tr}(\mathbf{A}) \cdot \delta_{ij}$$

Bild A.4: S4-Tensor für eine Diagonalmatrix mit der Spur der Matrix in den Diagonalelementen.

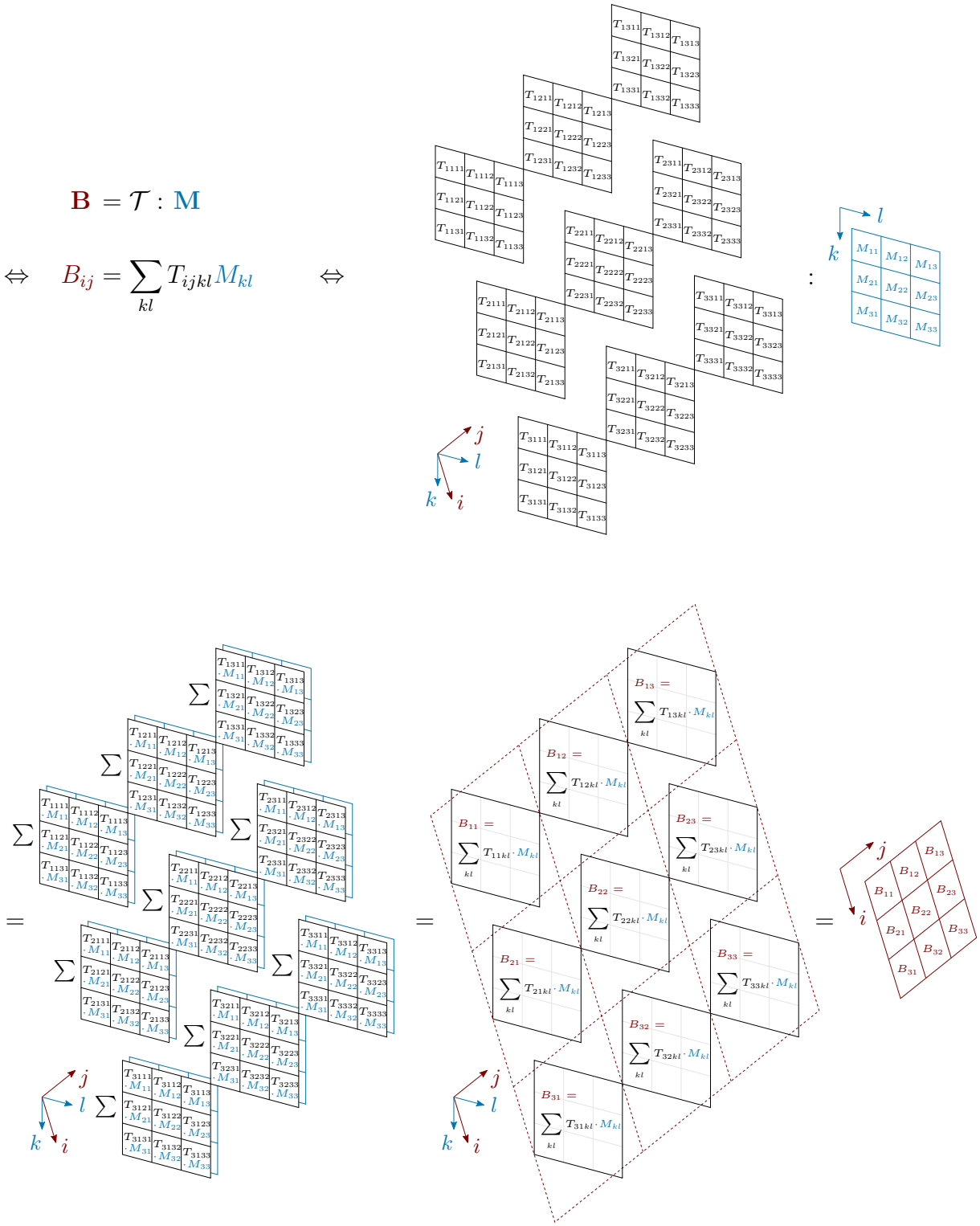


Bild A.5: Doppelte Kontraktion eines S4-Tensors mit einer Matrix.

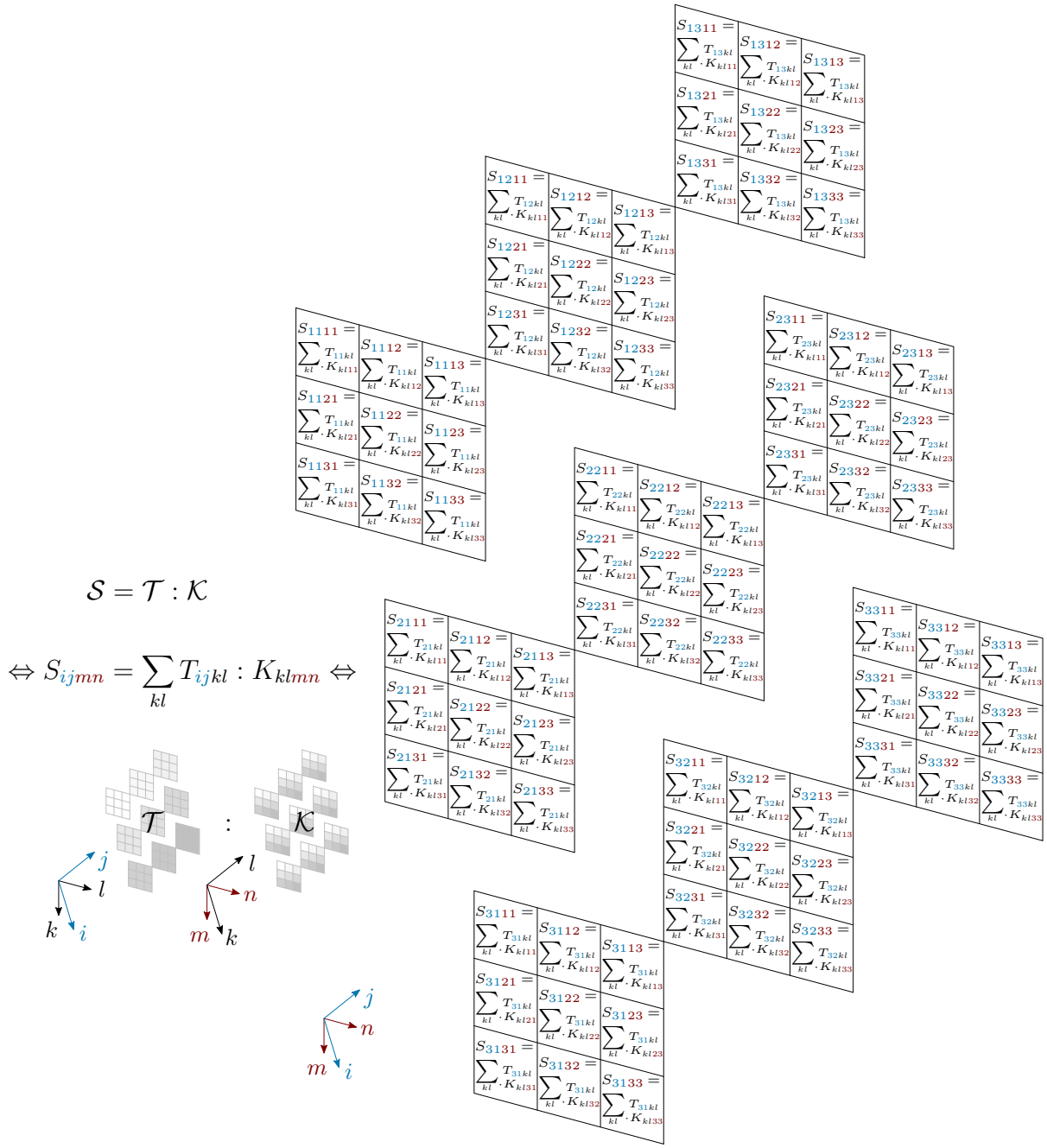


Bild A.6: Doppelte Kontraktion von zwei S4-Tensoren.

B Sensitivitätsstudie

Die Materialparameter der Stoffmodelle haben einen unterschiedlichen Einfluss auf das Bodenverhalten bei Nachrechnungen. Um diesen Effekt zu visualisieren, wird für die implementierten Stoffmodelle eine Sensitivitätsstudie, wie in Herle (1997), S. 69–76, durchgeführt.

Es werden jeweils Triaxialversuche mit einem Zelldruck von $-100\,\mathrm{kPa}$, einer axialen Stauchung von 20 %, $D_{\mathrm{ref}} = -1$ und $dt = 10^{-4}$ berechnet sowie Ödometerversuche mit $D_{\mathrm{ref}} = -1$, $dt = 10^{-5}$ und den Drücken $-1\,\mathrm{kPa}$ und $-500\,\mathrm{kPa}$. In jeder Untersuchung werden als Referenz die Parameter für Hostun-Sand aus Tabellen 6.9 bis 6.12 auf S. 125 verwendet und für das Stoffmodell `Hypo-Wu92` die Parameter für dicht gelagerten Sand aus Tabelle 6.8. Anschließend wird in weiteren Berechnungen jeder Parameter einzeln um ±20 % variiert. Die Ergebnisse sind für alle Stoffmodelle in den Bildern B.1 bis B.10 zu sehen.

Bei der aufgezeigten Sensitivitätsstudie muss beachtet werden, dass sie nur einen qualitativen Eindruck vermitteln kann. Wie bereits in Abschnitt 6.5 erwähnt, kann die Anpassung eines einzelnen Parameters ohne Rücksicht auf die Abhängigkeiten zu anderen Parametern zu einem inkonsistenten Parametersatz führen, für den die Zusammenhänge aus der Bestimmung der Materialparameter nicht mehr erfüllt sind. Die Variation um 20 % ist ebenfalls beliebig gewählt und berücksichtigt nicht, wie groß der typische oder gültige Wertebereich für einzelne Parameter ist.

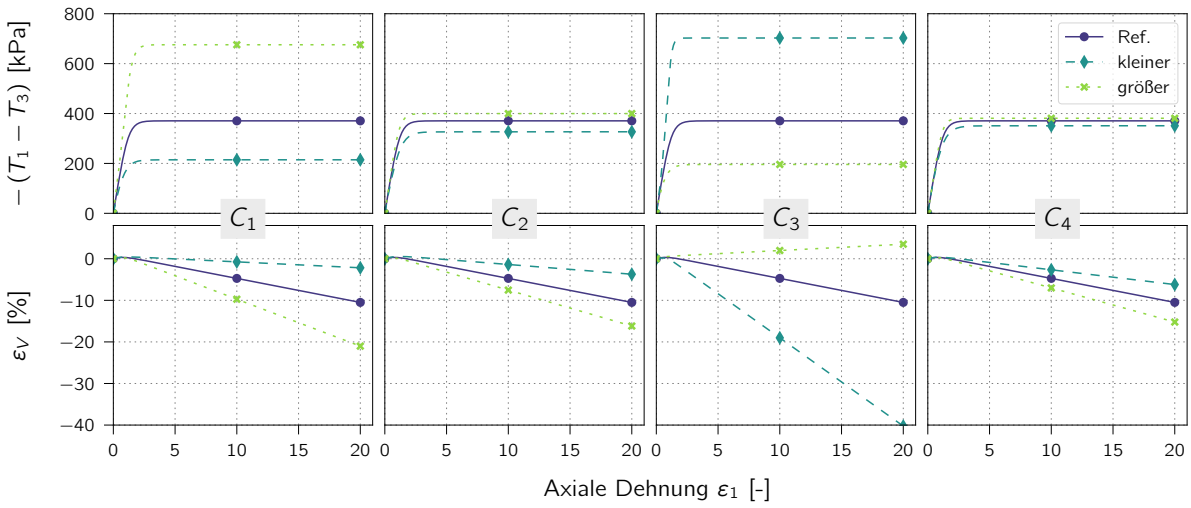


Bild B.1: Triaxialversuch mit dem Stoffmodell `Hypo-Wu92` und Variation der Parameter für dichten Sand aus Tabelle 6.8 auf S. 125 sowie einer Anfangsporenzahl von 0,53.

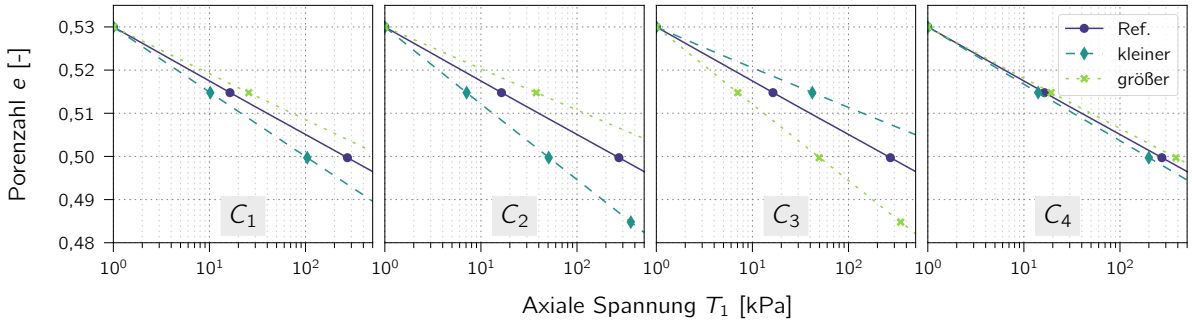


Bild B.2: Ödometerversuch mit dem Stoffmodell `Hypo-Wu92` und Variation der Parameter für dichten Sand aus Tabelle 6.8 auf S. 125 sowie einer Anfangsporenzahl von 0,53.

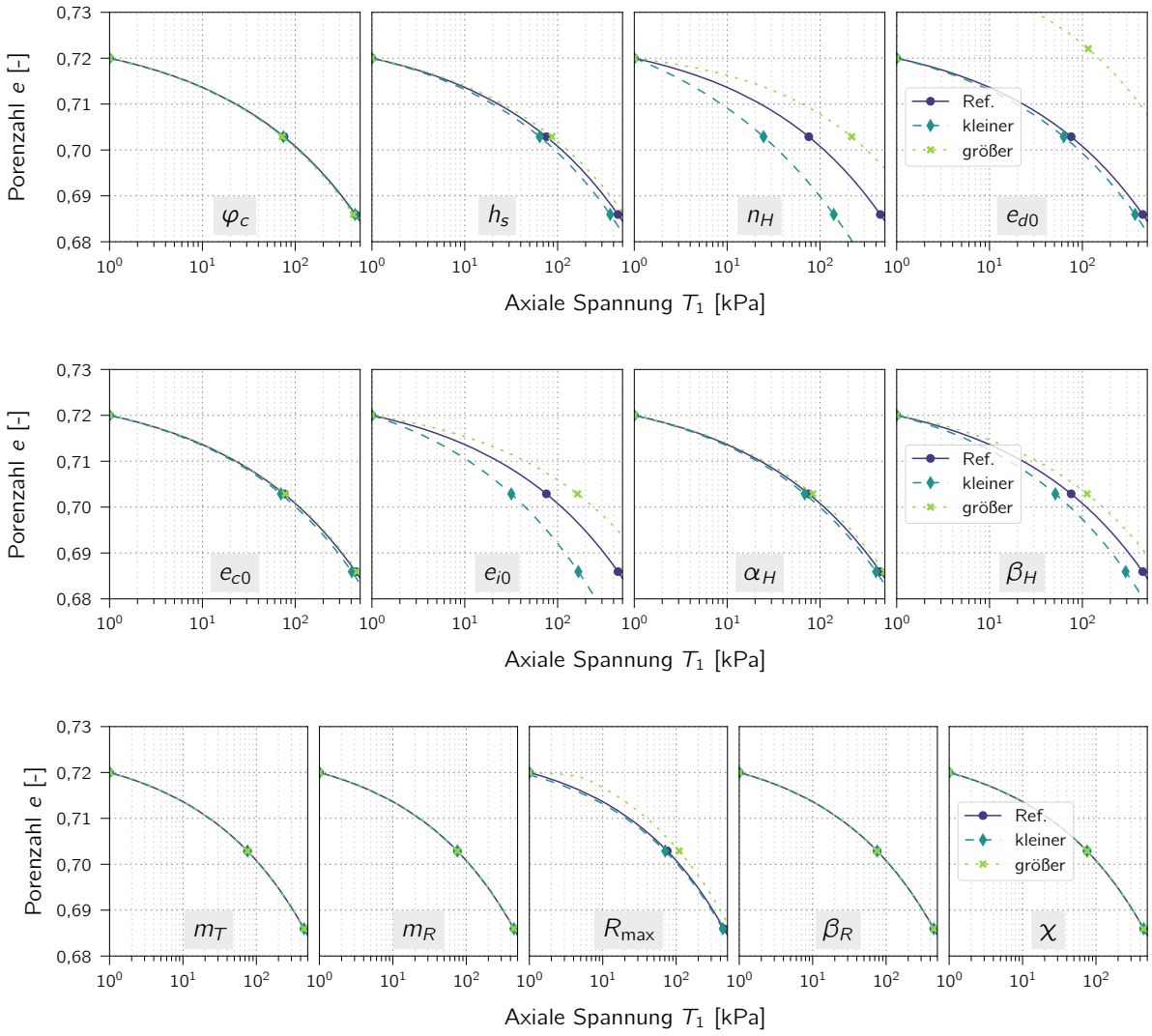


Bild B.3: Ödometerversuch mit dem Stoffmodell `Hypo-VW96` und Variation der Parameter für Hostun-Sand aus Tabelle 6.9 auf S. 125 sowie einer Anfangsporenzahl von 0,72.

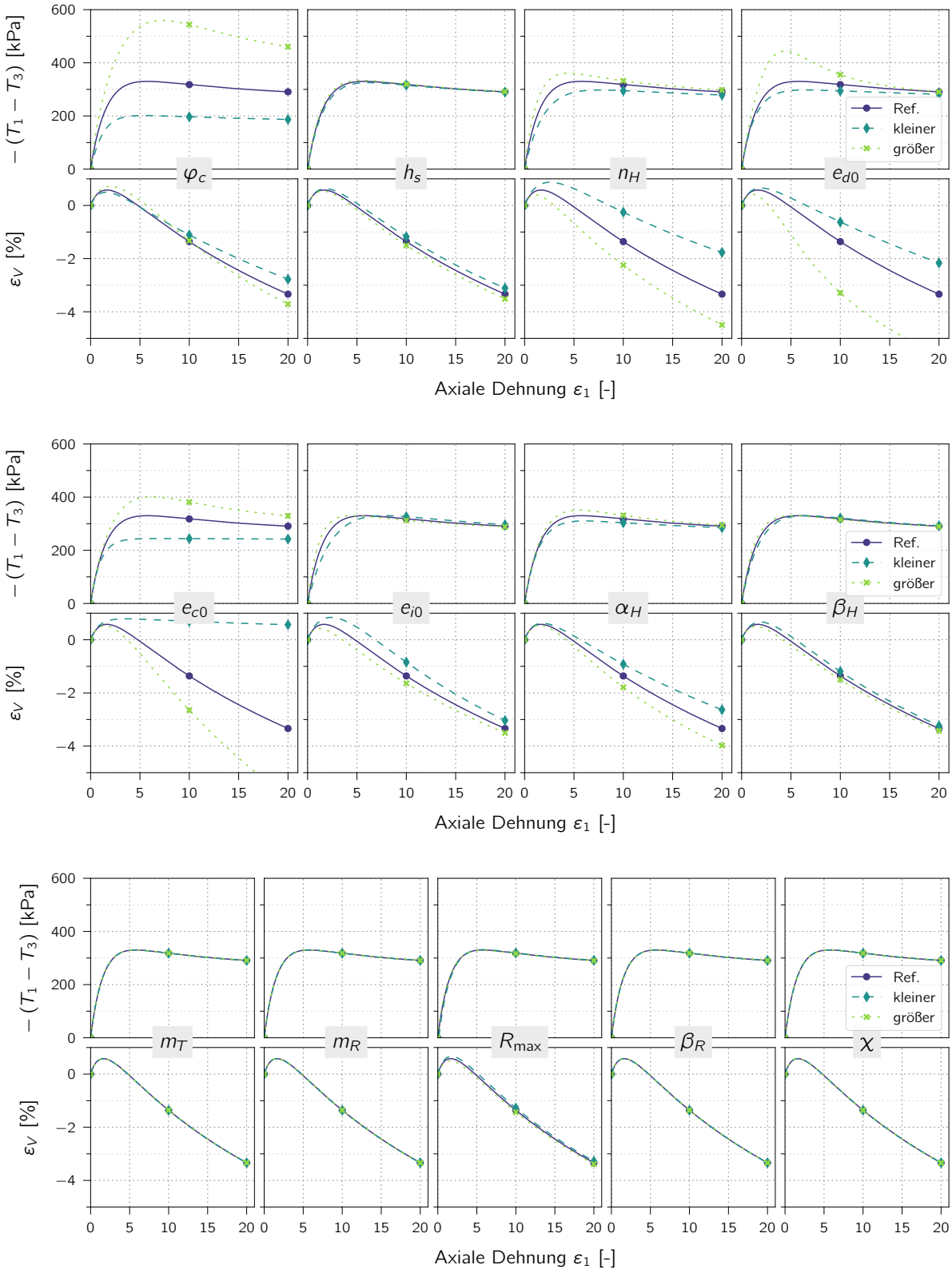


Bild B.4: Triaxialversuch mit dem Stoffmodell `Hypo-VW96` und Variation der Parameter für Hostun-Sand aus Tabelle 6.9 auf S. 125 sowie einer Anfangsporenzahl von 0,72 und voll ausgebildeter intergranularer Dehnung in Belastungsrichtung.

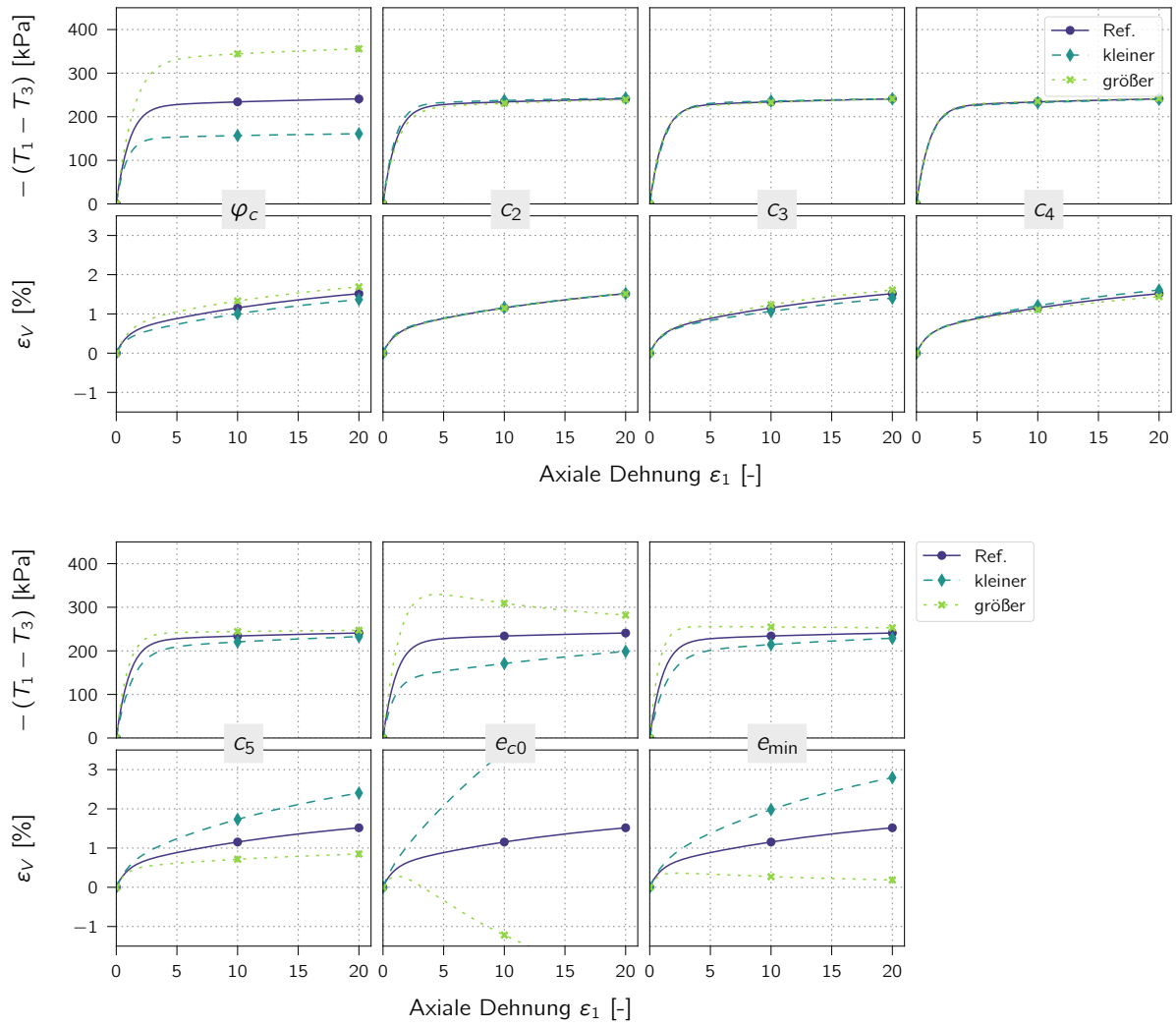


Bild B.5: Triaxialversuch mit dem Stoffmodell `Baro-Ko15` und Variation der Parameter für Hostun-Sand aus Tabelle 6.10 auf S. 125 sowie einer Anfangsporenzahl von 0,8.

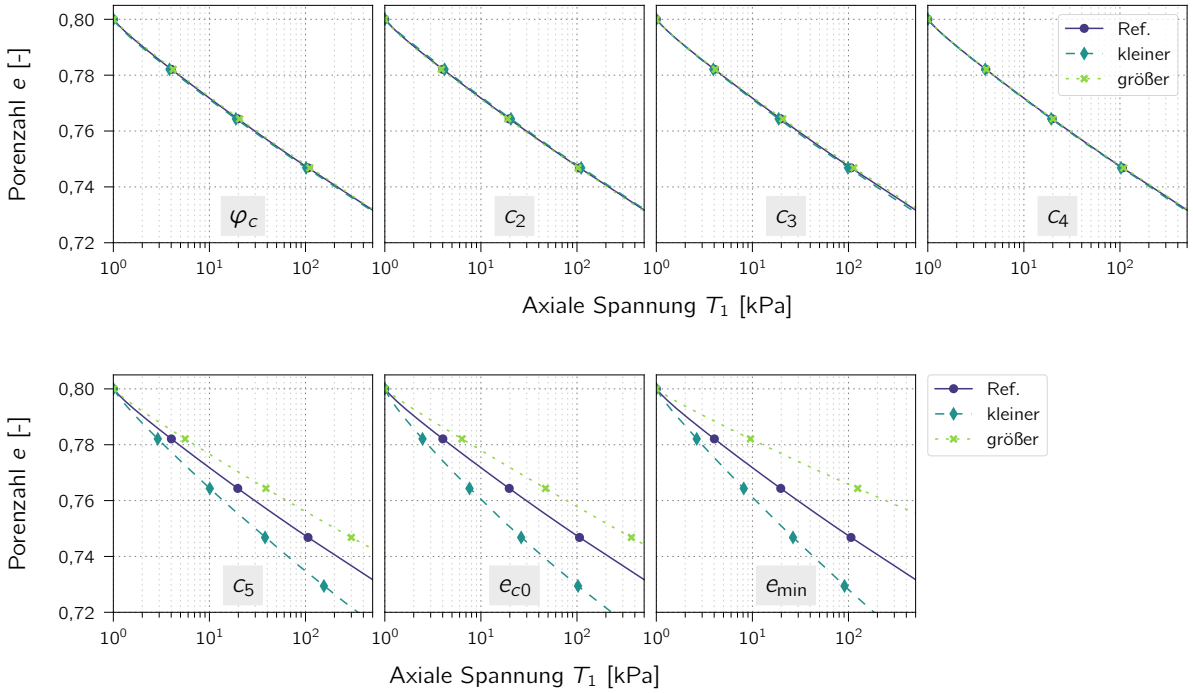


Bild B.6: Ödometerversuch mit dem Stoffmodell `Baro-Ko15` und Variation der Parameter für Hostun-Sand aus Tabelle 6.10 auf S. 125 sowie einer Anfangsporenzahl von 0,8.

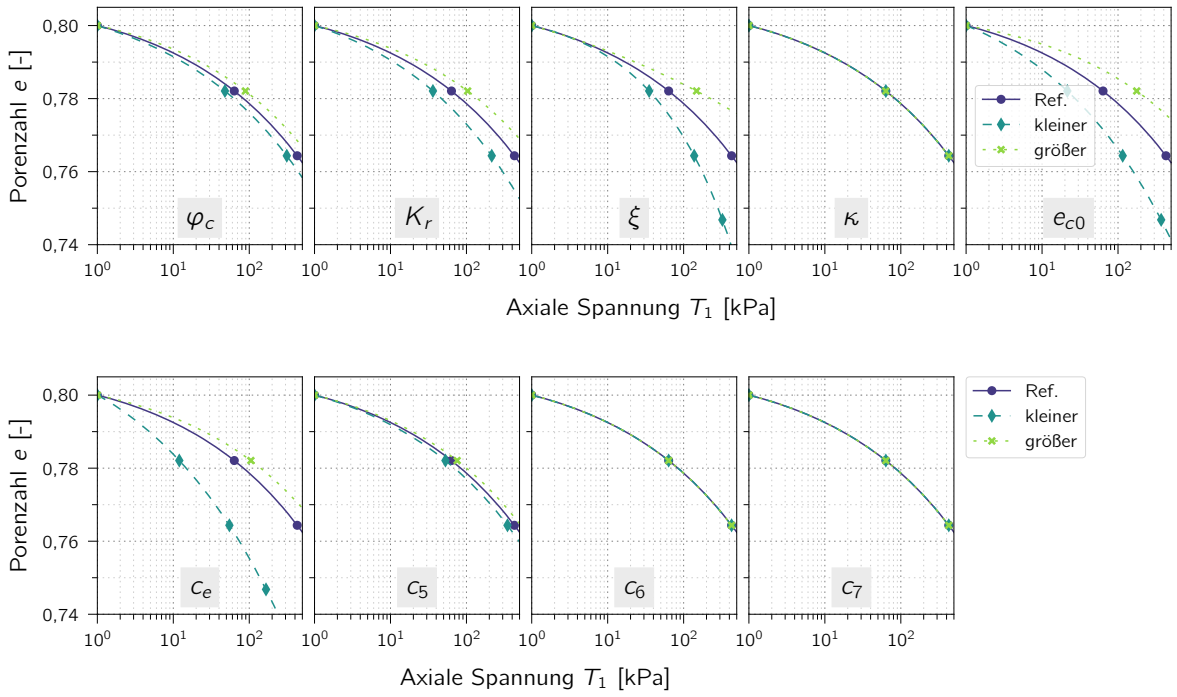


Bild B.7: Ödometerversuch mit dem Stoffmodell `Baro-Sc18` und Variation der Parameter für Hostun-Sand aus Tabelle 6.11 auf S. 125 sowie einer Anfangsporenzahl von 0,8.

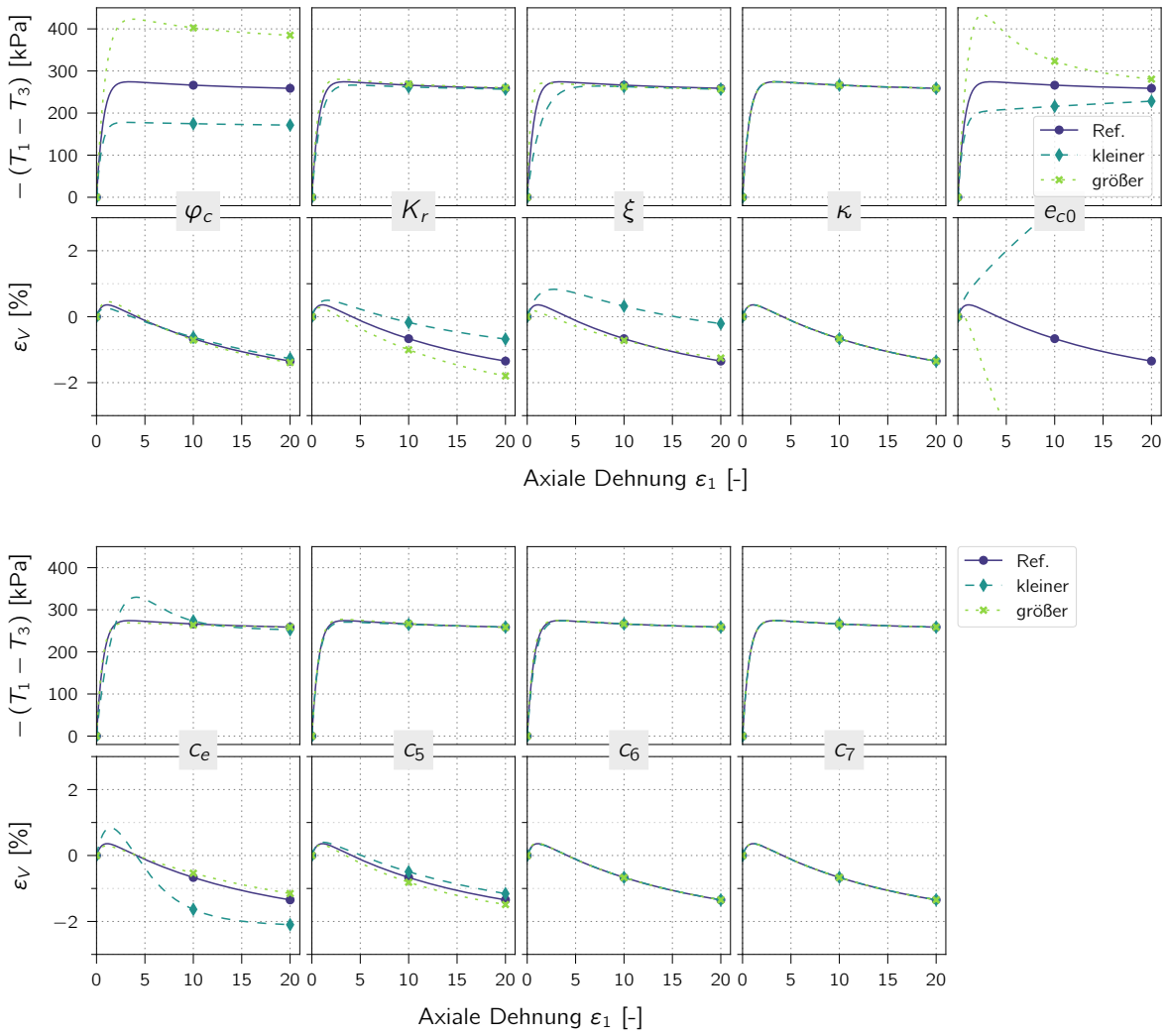


Bild B.8: Triaxialversuch mit dem Stoffmodell `Baro-Sc18` und Variation der Parameter für Hostun-Sand aus Tabelle 6.11 auf S. 125 sowie einer Anfangsporenzahl von 0,8.

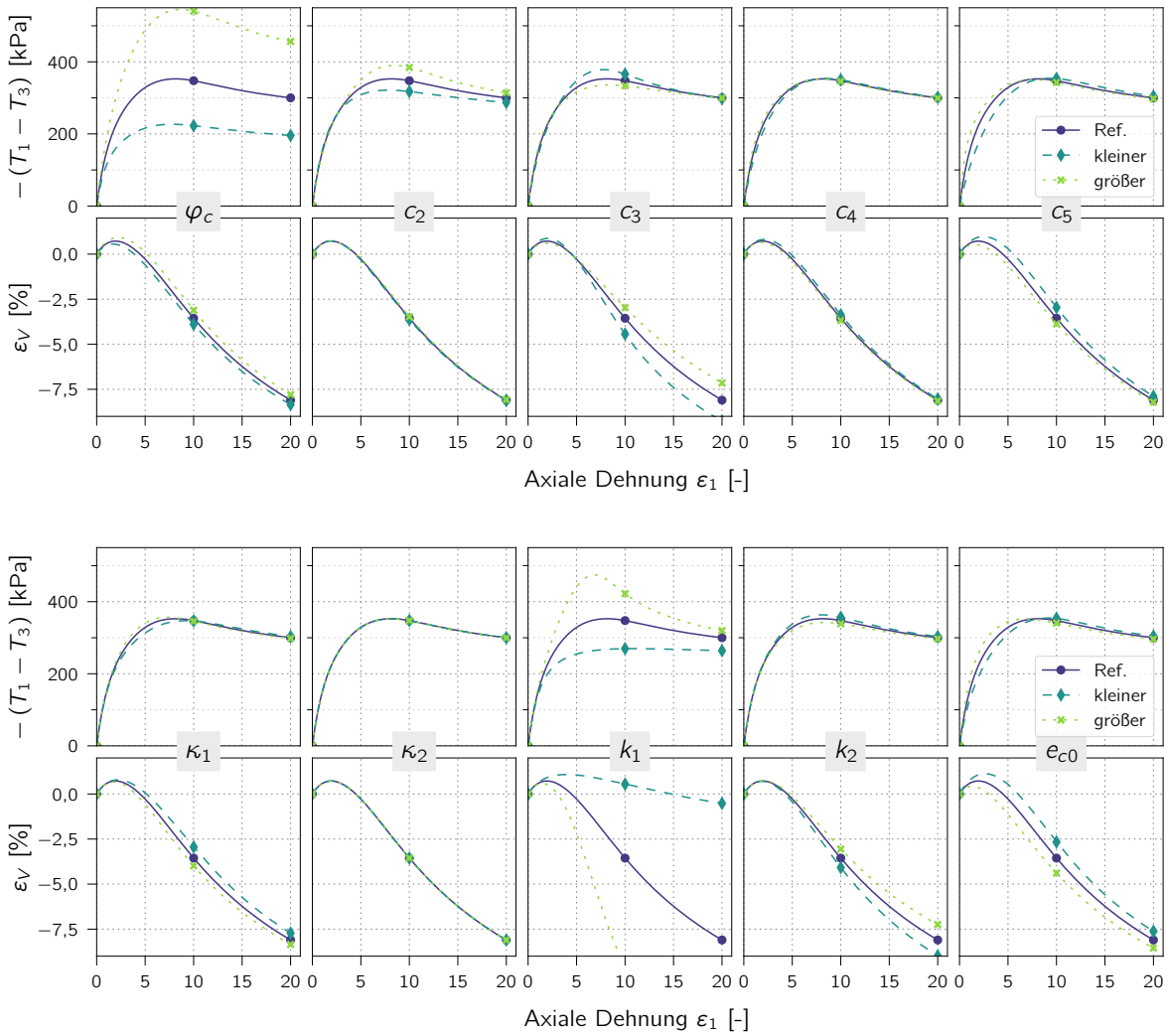


Bild B.9: Triaxialversuch mit dem Stoffmodell `Baro-Ko21` und Variation der Parameter für Hostun-Sand aus Tabelle 6.12 auf S. 125 sowie einer Anfangsporenzahl von 0,66. Da κ_2 den Wert null hat, findet für diesen Wert keine Variation statt.

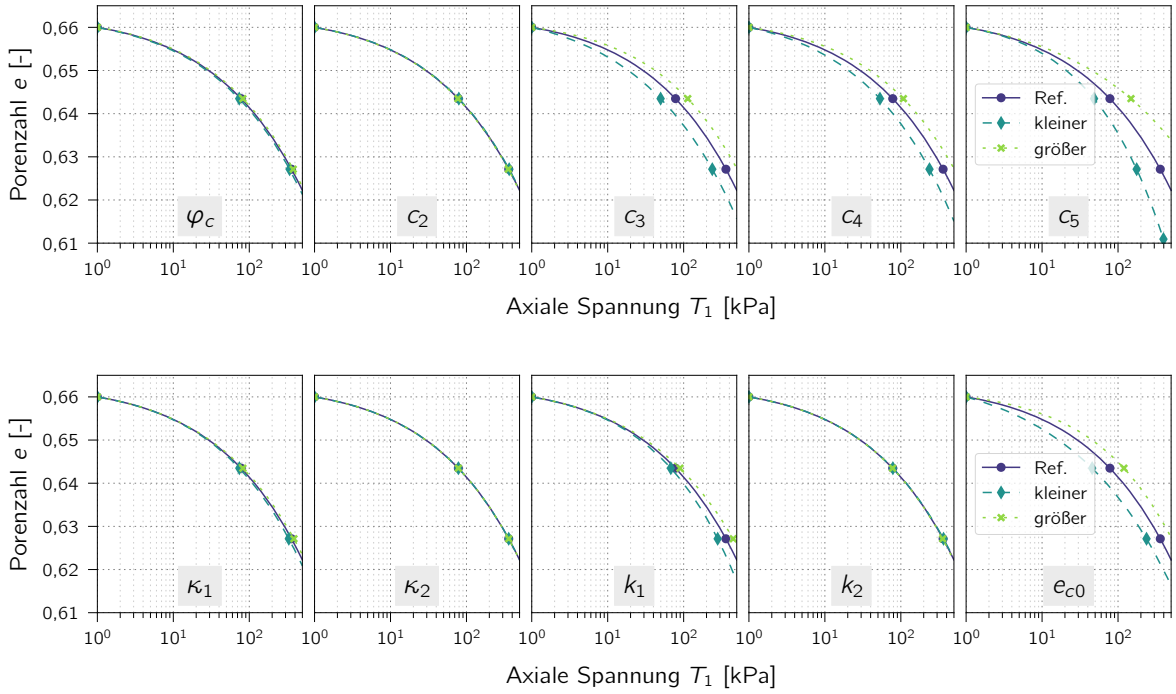


Bild B.10: Ödometerversuch mit dem Stoffmodell `Baro-Ko21` und Variation der Parameter für Hostun-Sand aus Tabelle 6.12 auf S. 125 sowie einer Anfangsporenzahl von 0,66. Da κ_2 den Wert null hat, findet für diesen Wert keine Variation statt.

C Explizite Runge-Kutta-Methoden

Explizites Euler-Verfahren

Beim expliziten Euler-Verfahren wird der Funktionswert der Differentialgleichung am Startpunkt für das ganze Intervall als konstant angenommen. Daraus folgt die Beziehung

$$\hat{y}_{n+1} = y_n + hk_1 \quad \text{mit} \qquad\qquad k_1 = f(x_n, y_n) \quad ,$$

oder in Form eines Butcher-Tableaus

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \quad .$$

```
dot_state = Evaluate_DGL(cur_state)
new_state = cur_state + dt*dot_state
```

Wie bereits in Abschnitt 2.4 erwähnt, ist $\hat{y}_{n+1}$ eine fehlerbehaftete Abschätzung der tatsächlichen Lösung y_{n+1}. Bild C.1 demonstriert die Akkumulation des Gesamtfehlers.

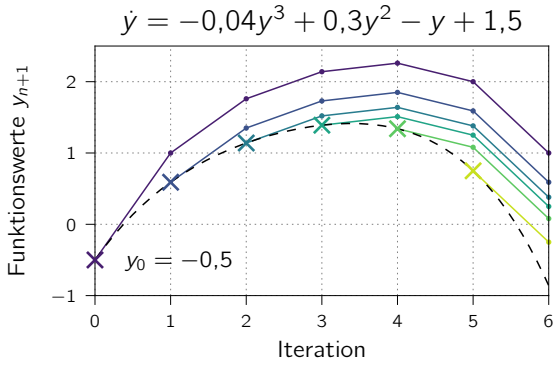


Bild C.1: Fehlerakkumulation durch die numerische Integration mit dem expliziten Euler-Verfahren. Gestrichelt dargestellt ist die Lösung $y(t)$ einer DGL $\dot{y}$ mit dem Anfangswert y_0. Für diskrete Zeitpunkte wird die exakte Lösung der DGL (Tangente) bestimmt und bis zum nächsten Zeitpunkt extrapoliert. Ausgehend von verschiedenen Startpunkten kann gesehen werden, wie sich der Gesamtfehler aus den vorherigen Schritten aufsummiert.

Explizites Euler-Verfahren mit Richardson-Extrapolation

In dieser Variante wird nicht nur die Informationen der Differentialgleichung am Startpunkt genutzt, sondern auch am Mittelpunkt der ersten Extrapolation (siehe Punkt 2b in Bild C.2). Dazu wird zwar eine zweite Auswertung der Differentialgleichung benötigt, aber die zusätzlichen Informationen liefern i. d. R. nicht nur eine bessere Abschätzung für x_{n+1} (Punkt 4a), sondern können auch mit der sog. Richardson-Extrapolation zur Abschätzung des Fehlers genutzt werden. Untersuchungen zur Integration von hypoplastischen Stoffroutinen sind bspw. in Fellin, Mittendorfer u. a. (2009) aufgezeigt. Es gilt

$$\hat{y}_{n+\frac{1}{2}} = y_n + \frac{h}{2} k_1 \quad \text{mit} \qquad k_1 = f(x_n, y_n) \quad \text{und}$$

$$\hat{y}_{n+1} = y_n + h k_2 \quad \text{mit} \qquad k_2 = f\left(x_n + \frac{h}{2}, \hat{y}_{n+\frac{1}{2}}\right) \quad .$$

Daraus ergibt sich das Butcher-Tableau

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} \quad .$$

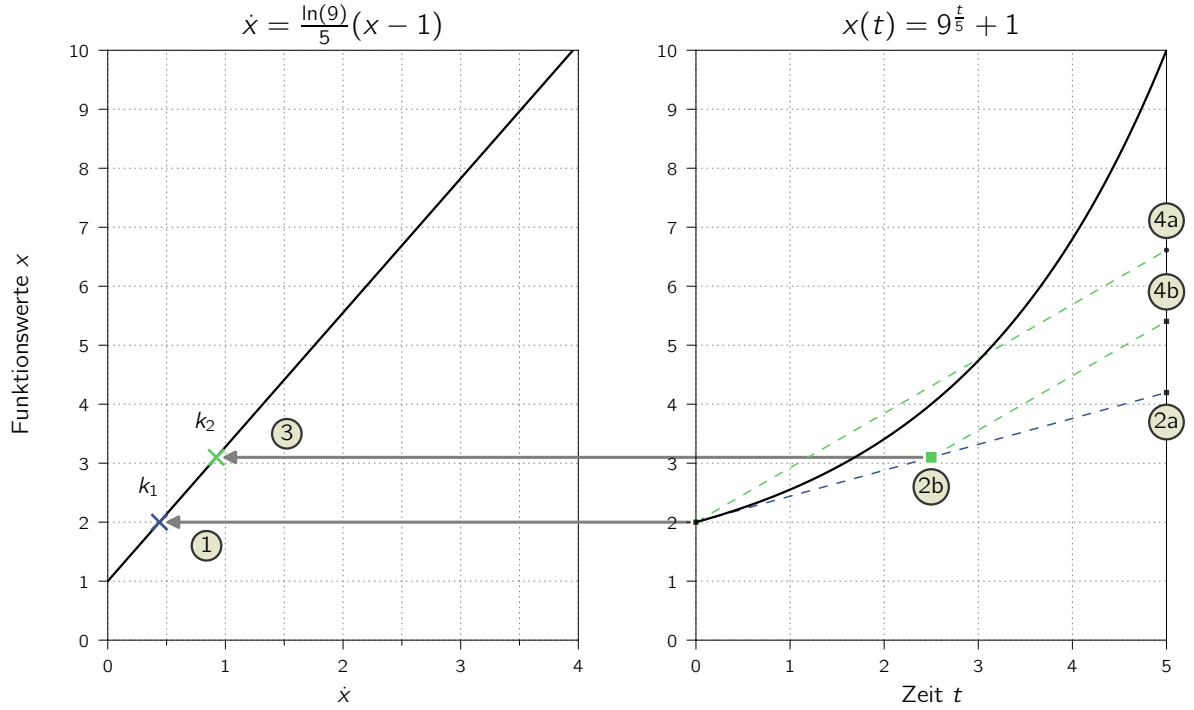


Bild C.2: Links ist der Phasenraum der Differentialgleichung $\dot{x} = \ln(9)\,(x-1)\,/5$ dargestellt, rechts eine Lösung $x(t) = 9^{\frac{t}{5}} + 1$ über dieselben Funktionswerte. Die Arbeitsschritte für das explizite Euler-Verfahren sind 1-2a und für das explizite Euler-Verfahren mit Richardson-Extrapolation 1-2b-3-4a (sowie 4b für Fehlerschätzung).

```
dot_state_k1 = Evaluate_DGL(cur_state)
dot_state_k2 = Evaluate_DGL(cur_state + dt*0.5_dp*dot_state_k1)

new_state = cur_state + dot_state_k2*dt
extra_state = cur_state + dt*(0.5_dp*dot_state_k1+0.5_dp*dot_state_k2)
```

Bogacki-Shampine (Runge-Kutta 2/3)

Mit dem Verfahren nach Bogacki-Shampine sind vier Auswertungen der Differentialgleichung erforderlich. Es wird die Lösung mit einem Verfahren der Ordnung 2 berechnet und der Fehler mit einem Verfahren der Ordnung 3 abgeschätzt. Wenn der Integrationsschritt akzeptiert wird, kann die letzte Auswertung der Differentialgleichung im nächsten Integrationsschritt wiederverwendet werden, was die Anzahl der tatsächlichen Auswertungen reduziert (FSAL-Eigenschaft). Die Funktionsweise ist an einem Beispiel in Bild C.3 dargestellt.

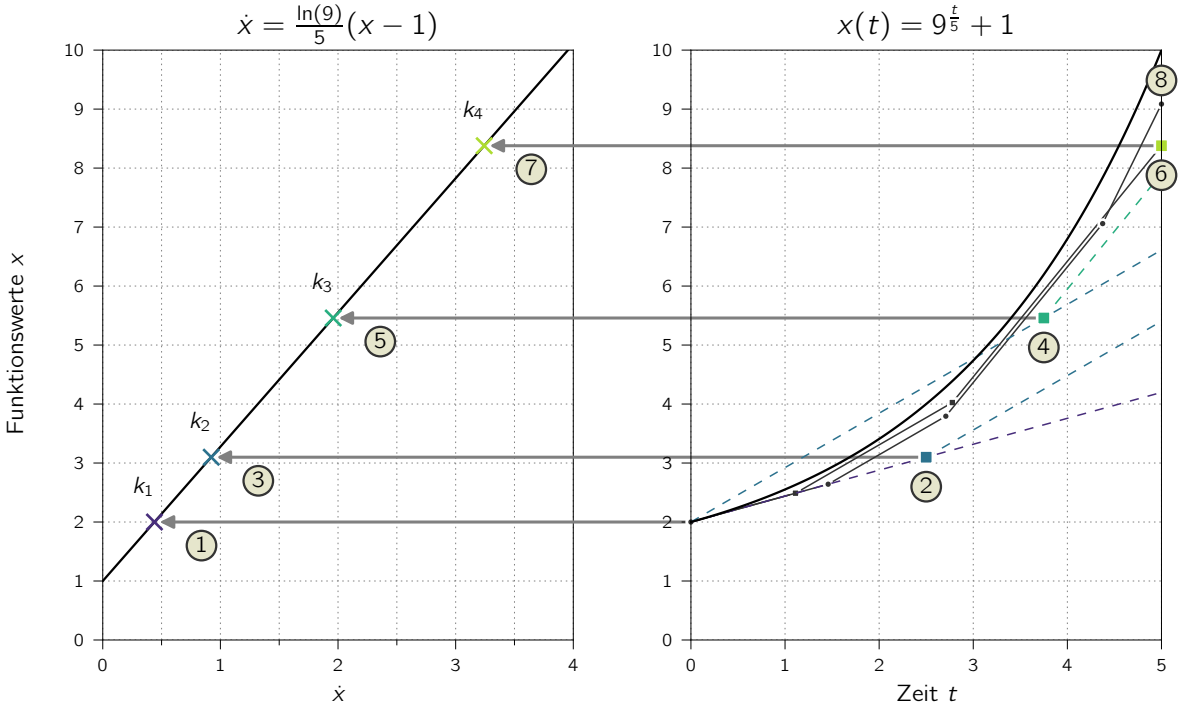


Bild C.3: Links ist der Phasenraum der Differentialgleichung $\dot{x} = \ln(9)\,(x-1)\,/5$ dargestellt, rechts eine Lösung $x(t) = 9^{\frac{t}{5}} + 1$ über dieselben Funktionswerte. Die Arbeitsschritte für das Verfahren nach Bogacki-Shampine sind mit aufsteigender Zahlenfolge dargestellt. Schritt 6 symbolisiert das Ergebnis des Verfahrens und Schritt 8 wird für die Fehlerschätzung genutzt.

Es gilt

$$\hat{y}_{n+1} = y_n + \frac{2}{9}hk_1 + \frac{1}{3}hk_2 + \frac{4}{9}hk_3 \quad \text{mit} \qquad k_1 = f(t_n, y_n) \quad ,$$
$$k_2 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \quad \text{und} \qquad k_3 = f(t_n + \frac{3}{4}h, y_n + \frac{3}{4}hk_2) \quad \text{sowie}$$
$$\hat{y}_{p,n+1} = y_n + \frac{7}{24}hk_1 + \frac{1}{4}hk_2 + \frac{1}{3}hk_3 + \frac{1}{8}hk_4 \quad \text{mit} \quad k_4 = f(t_n + h, \hat{y}_{n+1})$$

und somit

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{3}{4} & 0 & \frac{3}{4} & & \\ 1 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \\ \hline & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \\ & \frac{7}{24} & \frac{1}{4} & \frac{1}{3} & \frac{1}{8} \end{array} .$$

```
dot_state_k1 = Evaluate_DGL(cur_state)
dot_state_k2 = Evaluate_DGL(cur_state + dt*0.5_dp*dot_state_k1)
dot_state_k3 = Evaluate_DGL(cur_state + dt*0.75_dp*dot_state_k2)

new_state = cur_state + dt*(2.0_dp/9.0_dp*dot_state_k1 + 1.0_dp/3.0_dp*dot_state_k2 &
          + 4.0_dp/9.0_dp*dot_state_k3)
dot_state_k4 = Evaluate_DGL(new_state)
extra_state = cur_state + dt*(7.0_dp/24.0_dp*dot_state_k1 + 1.0_dp/4.0_dp*dot_state_k2 &
            + 1.0_dp/3.0_dp*dot_state_k3 + 1.0_dp/8.0_dp*dot_state_k4)
```

Simpsonregel (Runge-Kutta 2/3)

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

```
dot_state_k1 = Evaluate_DGL(cur_state)
dot_state_k2 = Evaluate_DGL(cur_state + dt*0.5_dp*dot_state_k1)
dot_state_k3 = Evaluate_DGL(cur_state + dt*(-dot_state_k1 + 2.0_dp*dot_state_k2))

new_state = cur_state + dt*(dot_state_k1/6.0_dp + 2.0_dp/3.0_dp*dot_state_k2 + dot_state_k3/6.0_dp)
extra_state = cur_state + dt*dot_state_k2
```

Runge-Kutta-Fehlberg 4/5

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	0

```
dot_state_k1 = Evaluate_DGL(cur_state)
dot_state_k2 = Evaluate_DGL(cur_state + dt*0.25_dp*dot_state_k1)
dot_state_k3 = Evaluate_DGL(cur_state + dt*(3.0_dp/32.0_dp*dot_state_k1 &
             + 9.0_dp/32.0_dp*dot_state_k2))
dot_state_k4 = Evaluate_DGL(cur_state + dt*(1932.0_dp/2197.0_dp*dot_state_k1 &
             - 7200.0_dp/2197.0_dp*dot_state_k2 + 7296.0_dp/2197.0_dp*dot_state_k3))
dot_state_k5 = Evaluate_DGL(cur_state + dt*(439.0_dp/216.0_dp*dot_state_k1 - 8.0_dp*dot_state_k2 &
             + 3680.0_dp/513.0_dp*dot_state_k3 - 845.0_dp/4104.0_dp*dot_state_k4))
dot_state_k6 = Evaluate_DGL(cur_state + dt*(-8.0_dp/27.0_dp*dot_state_k1 + 2.0_dp*dot_state_k2 &
             - 3544.0_dp/2565.0_dp*dot_state_k3 + 1859.0_dp/4104.0_dp*dot_state_k4 &
             - 11.0_dp/40.0_dp*dot_state_k5))

new_state = cur_state + dt*(25.0_dp/216.0_dp*dot_state_k1 + 1408.0_dp/2565.0_dp*dot_state_k3 &
          + 2197.0_dp/4104.0_dp*dot_state_k4 - 0.2_dp*dot_state_k5)
extra_state = cur_state + dt*(16.0_dp/135.0_dp*dot_state_k1 + 6656.0_dp/12825.0_dp*dot_state_k3 &
            + 28561.0_dp/56430.0_dp*dot_state_k4 - 9.0_dp/50.0_dp*dot_state_k5 &
            + 2.0_dp/55.0_dp*dot_state_k6)
```

Runge-Kutta 4/5 (Cash-Karp)

0						
$\frac{1}{5}$	$\frac{1}{5}$					
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				
$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			
1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		
$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	
	$\frac{37}{378}$	0	$\frac{250}{621}$	$\frac{125}{594}$	0	$\frac{512}{1771}$
	$\frac{2825}{27648}$	0	$\frac{18575}{48384}$	$\frac{13525}{55296}$	$\frac{277}{14336}$	$\frac{1}{4}$

```
dot_state_k1 = Evaluate_DGL(cur_state)
dot_state_k2 = Evaluate_DGL(cur_state + dt*0.2_dp*dot_state_k1)
dot_state_k3 = Evaluate_DGL(cur_state + dt*(3.0_dp/40.0_dp*dot_state_k1 &
             + 9.0_dp/40.0_dp*dot_state_k2))
dot_state_k4 = Evaluate_DGL(cur_state + dt*(3.0_dp/10.0_dp*dot_state_k1 &
             - 9.0_dp/10.0_dp*dot_state_k2 + 6.0_dp/5.0_dp*dot_state_k3))
dot_state_k5 = Evaluate_DGL(cur_state + dt*(-11.0_dp/54.0_dp*dot_state_k1 &
             + 5.0_dp/2.0_dp*dot_state_k2 - 70.0_dp/27.0_dp*dot_state_k3 &
             + 35.0_dp/27.0_dp*dot_state_k4))
dot_state_k6 = Evaluate_DGL(cur_state + dt*(1631.0_dp/55296.0_dp*dot_state_k1 &
             + 175.0_dp/512.0_dp*dot_state_k2 + 575.0_dp/13824.0_dp*dot_state_k3 &
             + 44275.0_dp/110592.0_dp*dot_state_k4 + 253.0_dp/4096.0_dp*dot_state_k5))

new_state = cur_state + dt*(37.0_dp/378.0_dp*dot_state_k1 + 250.0_dp/621.0_dp*dot_state_k3 &
          + 125.0_dp/594.0_dp*dot_state_k4 + 512.0_dp/1771.0_dp*dot_state_k6)
extra_state = cur_state + dt*(2825.0_dp/27648.0_dp*dot_state_k1 + 18575.0_dp/48384.0_dp*dot_state_k3 &
            + 13525.0_dp/55296.0_dp*dot_state_k4 + 277.0_dp/14336.0_dp*dot_state_k5 &
            + 0.25_dp*dot_state_k6)
```

Runge-Kutta 4/5 (Dormand-Prince)

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

```
dot_state_k1 = Evaluate_DGL(cur_state)
dot_state_k2 = Evaluate_DGL(cur_state + dt*0.2_dp*dot_state_k1)
dot_state_k3 = Evaluate_DGL(cur_state + dt*(3.0_dp/40.0_dp*dot_state_k1 &
             + 9.0_dp/40.0_dp*dot_state_k2))
dot_state_k4 = Evaluate_DGL(cur_state + dt*(44.0_dp/45.0_dp*dot_state_k1 &
             - 56.0_dp/15.0_dp*dot_state_k2 + 32.0_dp/9.0_dp*dot_state_k3))
dot_state_k5 = Evaluate_DGL(cur_state + dt*(19372.0_dp/6561.0_dp*dot_state_k1 &
             - 25360.0_dp/2187.0_dp*dot_state_k2 + 64448.0_dp/6561.0_dp*dot_state_k3 &
             - 212.0_dp/729.0_dp*dot_state_k4))
dot_state_k6 = Evaluate_DGL(cur_state + dt*(9017.0_dp/3168.0_dp*dot_state_k1 &
             - 355.0_dp/33.0_dp*dot_state_k2 + 46732.0_dp/5247.0_dp*dot_state_k3 &
             + 49.0_dp/176.0_dp*dot_state_k4 - 5103.0_dp/18656.0_dp*dot_state_k5))

new_state = cur_state + dt*(35.0_dp/384.0_dp*dot_state_k1 + 500.0_dp/1113.0_dp*dot_state_k3 &
          + 125.0_dp/192.0_dp*dot_state_k4 - 2187.0_dp/6784.0_dp*dot_state_k5 &
          + 11.0_dp/84.0_dp*dot_state_k6)
dot_state_k7 = Evaluate_DGL(new_state)
```

```
18 extra_state = cur_state + dt*(5179.0_dp/57600.0_dp*dot_state_k1 &
19             + 7571.0_dp/16695.0_dp*dot_state_k3 + 393.0_dp/640.0_dp*dot_state_k4 &
20             - 92097.0_dp/339200.0_dp*dot_state_k5 + 187.0_dp/2100.0_dp*dot_state_k6 &
21             + 1.0_dp/40.0_dp*dot_state_k7)
```

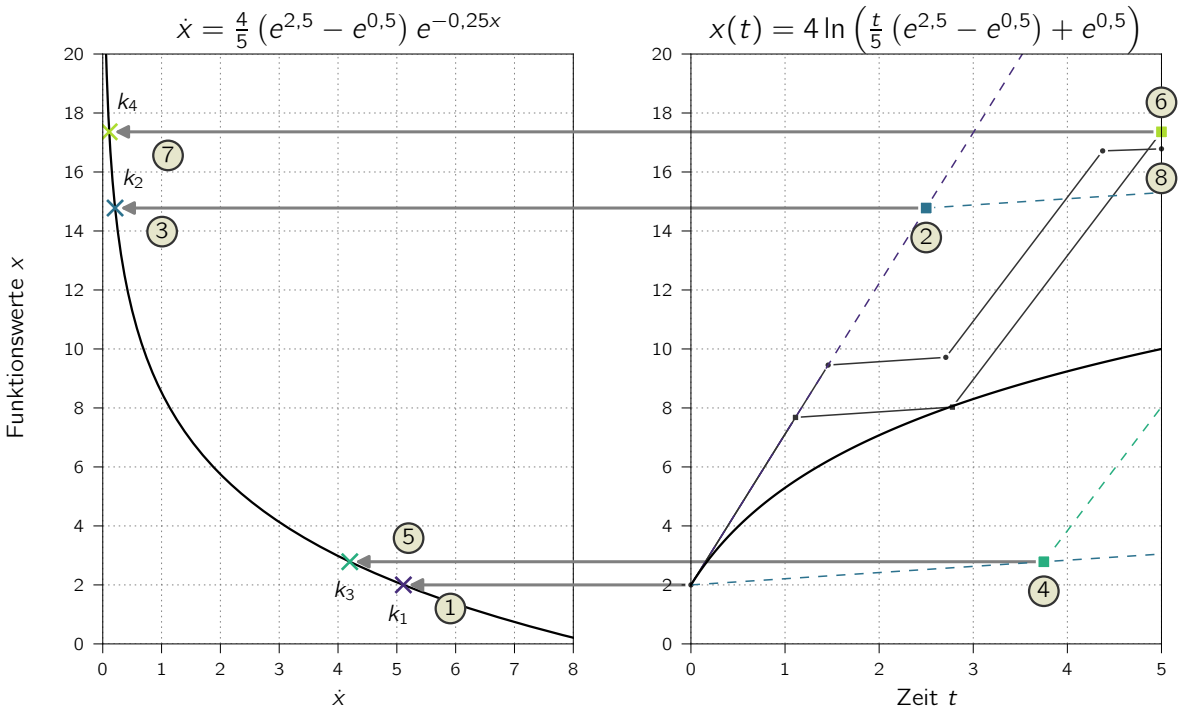


Bild C.4: Für zu große Zeitschritte kann auch mit einem expliziten Runge-Kutta-Verfahren höherer Ordnung der Unterschied zwischen dem geschätzten und dem tatsächlichen Fehler groß sein. Insbesondere bei steifen Problemen eignen sich (semi-)implizite Integrationsverfahren besser.

D Dokumentation der xMat-Funktionalitäten

Allgemeine Einstellungen (Modul `General_Settings`)

Ergänzend zu den einstellbaren `setting_...`-Parametern aus Abschnitt 4.2.1 sind in diesem Abschnitt die Hilfsfunktionen des Moduls `General_Settings` dokumentiert. Für die Ausgabe von Nachrichten oder das Beenden der Berechnung stehen die folgenden Funktionen zur Verfügung:

- `subroutine Write_Message(message`$^{\text{in}}$`)` um eine Nachricht in die Logdatei oder Textausgabe zu schreiben,
- `subroutine Write_Warning(message`$^{\text{in}}$`)` um eine Warnung in die Logdatei oder Textausgabe zu schreiben und
- `subroutine Write_Error_And_Exit(message`$^{\text{in}}$`)` um eine Fehlermeldung in die Logdatei oder Textausgabe zu schreiben und die Berechnung zu beenden.

In diesem Modul ist auch die folgende Funktion definiert:

- `elemental function Is_Nan(number`$^{\text{in}}$`)` um zu überprüfen, ob es sich bei der übergebenen Zahl um eine ungültige Zahl handelt, die als NaN (*„Not a Number“*) bezeichnet wird – wie das Ergebnis von $0/0$. Jede Rechenoperation mit einem NaN erzeugt wieder ein NaN.

Außerdem werden die folgenden Hilfsroutinen für die Schnittstellen bereitgestellt:

- `pure subroutine Estimate_Components(nel`$^{\text{in}}$`, num_dimensions`$_{\text{out}}$`, num_shear`$_{\text{out}}$`)` zur Initialisierung der Komponenten auf Basis der Anzahl an Gesamtkomponenten. Erwartet werden $3 \leq \text{nel} \leq 6$ Gesamtkomponenten, mit der vordefinierten Aufteilung $(2,1), (3,1), (3,2)$ und $(3,3)$.
- `function Check_Input_Dimensions(num_dimensions`$^{\text{in}}$`, num_shear`$^{\text{in}}$`)` zur Überprüfung, ob die übergebene Anzahl an Einzelkomponenten von der xMat unterstützt wird. Als Seiteneffekt werden die globalen Werte `global_num_direct_components` und `global_num_shear_components` angepasst. Gültige Eingaben sind $(2,1)$, $(3,1)$, $(3,2)$ und $(3,3)$.
- `pure function Uppercase(text`$^{\text{in}}$`)` zur Umwandlung aller Kleinbuchstaben in einer Zeichenkette in Großbuchstaben. Wird in den Schnittstellen genutzt, um die Bezeichnungen von Stoffmodellen einheitlich anzupassen.

- `pure function C_To_Fortran_String(length`$^{\text{in}}$`, c_string`$^{\text{in}}$`)` zum Umwandeln von C-Strings (`char*`) der angegebenen Länge in eine Fortran `character`-Variable.

Kontrollausgabe (Modul `Debug`)

Dieses Modul stellt `function Formatval(name`$^{\text{in}}$`, variable`$^{\text{in}}$`)` zur Verfügung, um Variablen formatiert mit dem angegebenen Namen auszugeben. Ursprünglich als Hilfe beim Fehlerfinden (*„Debugging“*) entwickelt, hat sich diese Funktion aktuell vielerorts in den Aufrufen von `Write_Message()`, `Write_Warning()` und `Write_Error_And_Exit()` zur formatierten Ausgabe von Variablen etabliert.

Innerhalb des Moduls ist `Formatval` als Schnittstelle definiert, die je nach Variablentyp eine passende Ausgabefunktion aufruft. Unterstützt werden

- skalare Größen vom Typ `logical`, `integer` oder `real(dp)` und
- Vektoren, Matrizen und S4-Tensoren vom Typ `real(dp)`.

Am Anfang des Moduls sind Parameter zur Formatierung der Ausgabe definiert, um bspw. die Zeichenlänge und das Format anzupassen. Es sollten keine Änderungen notwendig sein, aber wenn Anpassungen gewünscht sind, muss auf die vorhandenen Abhängigkeiten der Parameter geachtet werden:

- `debug_format` und `alternative_format` müssen die gleiche Länge bzw. Anzahl an Zeichen haben und eine Ausgabe der gleichen Länge `onenumberlength` erzeugen. Aktuell werden dafür die Formate `f12.x` und `es12.x` genutzt und
- falls `mat_identifierlength` oder `tens_identifierlength` geändert werden, muss die lokale Variable `identifier` in der jeweiligen Funktionen `Format_dp_dim2()` bzw. `Format_dp_dim4()` angepasst werden. Beide Werte sind als Modulparameter definiert, um eine explizite Berechnung der Länge der Zeichenkette in den Funktionen zu ermöglichen.

Mathematische Hilfsfunktionen (Modul `Math_Operations`)

In diesem Modul finden sich Parameter und Funktionen, die alle notwendigen mathematischen Operationen der implementierten Stoffmodelle bereitstellen. Da die xMat als Komplettlösung ohne zusätzliche externe Abhängigkeiten entwickelt worden ist, sind viele mathematische Operationen dafür neu geschrieben und getestet worden.[1] Intern arbeitet die xMat mit einer von mehreren möglichen Repräsentationen von Matrizen und S4-Tensoren, auf die alle Hilfsfunktionen abgestimmt sind. Standardmäßig werden S4-Tensoren als 6×6-Matrizen repräsentiert, aber ein Wechsel ist in der xMat-Entwicklungsumgebung möglich (siehe Abschnitt 5.2).

[1] Falls existierende und bewährte Bibliotheken wie bspw. LAPACK genutzt werden sollen, ist dies mit der xMat mit ein wenig Aufwand ebenfalls möglich.

In diesem Modul sind verschiedene Konstanten definiert, wie

- `const_root2` ($\sqrt{2}$),
- `const_root3` ($\sqrt{3}$) und
- `const_root6` ($\sqrt{6}$).

Zusätzlich werden für die gewählte Repräsentation die Einheitsmatrix `const_identity2d` und die folgenden beiden S4-Tensoren definiert:

- `const_identity4d_sym` als $\mathcal{I}^{\text{sym}}$ mit $I^{\text{sym}}_{ijkl} = \frac{1}{2}\left(\delta_{ik} \otimes \delta_{jl} + \delta_{il} \otimes \delta_{jk}\right)$ und
- `const_identity4d_tr` als $\mathcal{I}^{\text{tr}}$ mit $I^{\text{tr}}_{ijkl} = \delta_{ij} \otimes \delta_{kl}$.

Zur eindeutigen Zuordnung von Matrix- und Tensoreinträgen in unterschiedlichen Repräsentationen werden intern `ref_indices` und ggf. `ref_elements` genutzt.

Mathematische Funktionen unabhängig von der gewählten Repräsentation

Die folgenden Funktionen werden für die Ermittlung von Eigenwerten und Eigenvektoren einer Matrix genutzt:

- `subroutine Eigendecomposition(mat`$^{\text{in}}$`, nel`$^{\text{in}}$`, eigenvalues`$_{\text{out}}$`, eigenvectors`$_{\text{out}}$`)` führt eine Eigenwertzerlegung (QL-Zerlegung) der übergebenen quadratischen Matrix durch. Falls die QL-Zerlegung nicht erfolgreich ist, wird die gesamte Berechnung beendet. Anderenfalls werden die berechneten Eigenwerte und Eigenvektoren zurückgegeben,
- `pure subroutine Tridiagonal_Matrix(matrix`$^{\text{in}}_{\text{out}}$`, nel`$^{\text{in}}$`, trans_mat`$_{\text{out}}$`)` erstellt eine Tridiagonalmatrix und überschreibt die übergebene quadratische Matrix damit. Die Vorgehensweise orientiert sich an Press u. a. (1997), S. 462–469, sowie Golub und Van Loan (2013), S. 458 f. Außerdem wird die Transformationsmatrix mit den durchgeführten Householder-Transformationen zurückgegeben,
- `pure subroutine QL_Decomposition(matrix`$^{\text{in}}_{\text{out}}$`, nel`$^{\text{in}}$`, eigenvec_mat`$_{\text{out}}$`, success`$_{\text{out}}$`)` führt angelehnt an Press u. a. (1997), S. 469–474, die QL-Zerlegung der übergebenen quadratischen Tridiagonalmatrix durch (siehe auch Golub und Van Loan, 2013, S. 461–464). Die übergebene Matrix wird mit einer Diagonalmatrix mit den Eigenwerten überschrieben und es wird eine Matrix mit den Eigenvektoren zurückgegeben. Ein Wahrheitswert gibt an, ob die Zerlegung erfolgreich gewesen ist oder nicht und
- `pure function Givens_CS(refval`$^{\text{in}}$`, zeroval`$^{\text{in}}$`)` berechnet eine Givens-Rotation, bei der durch eine Drehung in der Spalte mit dem Diagonalwert `refval` der andere gegebene Wert zu null wird. Givens-Rotationen werden im Zusammenhang mit der QL-Zerlegung genutzt und geben eine 2×2-Rotationsmatrix zurück.

Die nächsten drei Funktionen sind für die LU-Zerlegung und der Bestimmung der Inversen einer Matrix:

- `pure subroutine LU_Decomposition(matrix`$^{\text{in}}_{\text{out}}$`, nel`$^{\text{in}}$`, trans_mat`$_{\text{out}}$`, success`$_{\text{out}}$`)` führt angelehnt an Press u. a. (1997), S. 36–40, die LU-Zerlegung der übergebenen quadratischen Matrix $\mathbf{A}$ in der Form $\mathbf{PA} = \mathbf{LU}$ durch (siehe auch Golub und Van Loan, 2013, S. 111–114 und S. 125–130). Die übergebene Matrix wird mit der zusammengesetzten LU-Matrix überschrieben ($\mathbf{L}+\mathbf{U}-\mathbf{I}$) und die Permutationsmatrix $\mathbf{P}$ wird gesondert zurückgegeben. Ein Wahrheitswert gibt an, ob die Zerlegung erfolgreich gewesen ist,
- `pure subroutine LU_Solve(lu_mat`$^{\text{in}}$`, nel`$^{\text{in}}$`, trans_mat`$^{\text{in}}$`, b_vec`$^{\text{in}}_{\text{out}}$`)` kann nach einer erfolgreichen LU-Zerlegung mit der LU-Matrix und der Permutationsmatrix aus `LU_Decomposition()` genutzt werden, um das Gleichungssystem $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ nach $\boldsymbol{x}$ für beliebige rechte Seiten $\boldsymbol{b}$ zu lösen. Die Lösung überschreibt den Eingabevektor und
- `pure subroutine Inverse_Internal(matrix`$^{\text{in}}$`, inv_matrix`$_{\text{out}}$`, nel`$^{\text{in}}$`, success`$_{\text{out}}$`)` nutzt intern die LU-Zerlegung, um die Inverse der übergebenen quadratischen Matrix zu berechnen. Ein Wahrheitswert gibt an, ob die Inverse erfolgreich berechnet werden konnte oder nicht.

In der xMat werden mehrere Größen (Spannungen, Jacobi-Matrix und Statusvariablen) zwischenzeitlich in einem einzigen großen Vektor gespeichert. Zum Schreiben und Entnehmen der Werte werden die beiden folgenden Funktionen genutzt:

- `pure function Pack_States(stress`$^{\text{in}}$`, jacobian`$^{\text{in}}$`, statevariables`$^{\text{in}}$`)` wird dafür genutzt, um einen Vektor aus einer Matrix (Spannungen), einem S4-Tensor (Jacobi-Matrix) und einem Vektor (Statusvariablen) zu erzeugen und
- `pure subroutine Unpack_States(input_states`$^{\text{in}}$`, stress`$_{\text{out}}$`, jacobian`$_{\text{out}}$`, statevariables`$_{\text{out}}$`)` ist das Gegenstück zu `Pack_States()`, um die gespeicherte Matrix, S4-Tensor und Vektor wieder aus dem übergebenen Vektor zu extrahieren.

Weitere Hilfsfunktionen der xMat sind:

- `pure function Value_In_Interval(value`$^{\text{in}}$`, limits`$^{\text{in}}$`)` zur Überprüfung, ob sich eine Variable in einem angegebenen Intervall befindet (bspw. gültige Grenzen von Materialparametern),
- `subroutine Abort_If_Not_In_Interval(name`$^{\text{in}}$`, number`$^{\text{in}}$`, limits`$^{\text{in}}$`)` um zusätzlich zur vorangegangenen Überprüfung die aktuelle Berechnung abzubrechen, wenn die Überprüfung fehlgeschlagen ist,
- `elemental function Sqrt_Of_Sum_Of_Squares(a`$^{\text{in}}$`, b`$^{\text{in}}$`)` zur elementweisen Berechnung von $\sqrt{a_i^2 + b_i^2}$ für die Vektoren $\boldsymbol{a}$, $\boldsymbol{b}$ und
- `elemental function Nonzero_Division(val`$^{\text{in}}$`, fac`$^{\text{in}}$`)` zur elementweisen Berechnung von `val/fac` für `fac` $\neq 0$. Für `fac` $= 0$ wird `val` zurückgegeben.

Mathematische Funktionen mit Abhängigkeit zur gewählten Repräsentation

Die folgenden Funktionen implementieren viele der in Abschnitt 2.1 vorgestellten Rechenoperationen:

- `pure function Trace(mat`$^{\text{in}}$`)` zur Berechnung von $\mathrm{tr}(\mathbf{M})$,
- `pure function Dimensionless(mat`$^{\text{in}}$`)` zur Berechnung von $\hat{\mathbf{M}}$,
- `pure function Deviatoric_Part(mat`$^{\text{in}}$`)` zur Berechnung von $\mathbf{M}^{\text{dev}}$,
- `pure function Determinant(mat`$^{\text{in}}$`)` zur Berechnung von $\det(\mathbf{M})$,
- `pure function Double_Contraction22(mata`$^{\text{in}}$`, matb`$^{\text{in}}$`)` berechnet $\mathbf{A} : \mathbf{B}$,
- `pure function Double_Contraction42(tens`$^{\text{in}}$`, mat`$^{\text{in}}$`)` berechnet $\mathcal{T} : \mathbf{M}$,
- `pure function Double_Contraction44(tensa`$^{\text{in}}$`, tensb`$^{\text{in}}$`)` berechnet $\mathcal{A} : \mathcal{B}$,
- `pure function Dyadic_Product22(mata`$^{\text{in}}$`, matb`$^{\text{in}}$`)` berechnet $\mathbf{A} \otimes \mathbf{B}$,
- `pure function Matrix_Rotation(mat`$^{\text{in}}$`, rot_mat33`$^{\text{in}}$`)` zur Berechnung von $\mathbf{RMR}^T$,
- `pure function Norm(mat`$^{\text{in}}$`)` zur Berechnung von $||\mathbf{M}||$ und
- `function Matrix_Exponential(mat`$^{\text{in}}$`)` zur Berechnung von $e^{\mathbf{M}}$.

Ergänzend dazu werden die folgenden Rechenoperationen definiert:

- `pure function Squared(mat`$^{\text{in}}$`)` zur Berechnung von $\mathbf{M}^2$,
- `pure function Is_Diagonal(mat`$^{\text{in}}$`)` überprüft, ob die übergebene Matrix eine Diagonalmatrix ist,
- `pure subroutine Inverse_Tensor(tensor`$^{\text{in}}$`, inv_tensor`$_{\text{out}}$`, success`$_{\text{out}}$`)` berechnet mit `Inverse_Internal()` die Inverse der Matrix-Repräsentation des übergebenen Tensors. Die Größe des übergebenen Tensors ist durch die gewählte Repräsentation definiert, aber der tatsächlich genutzte Bereich mit `global_num_direct_components` und `global_num_shear_components`. Deshalb müssen zuerst die tatsächlich genutzten Bereiche extrahiert und nach dem Invertieren wieder den richtigen Positionen in der Ausgabematrix zugeordnet werden. Ein Wahrheitswert gibt an, ob die Inverse erfolgreich berechnet werden konnte und
- `pure function Tensor_Partialtrace(tens`$^{\text{in}}$`)` berechnet $\left(J_{1111} + J_{2222} + J_{3333}\right)/3$.

Die folgenden Funktionen dienen der Verarbeitung und Umwandlung von Vektoren, Matrizen und Tensoren intern oder zwischen der Größe der internen Repräsentation und der vom aufrufenden Programm genutzten Größe `num_dimensions+num_shear`:

- `pure function Export_Matrix(mat`$^{\text{in}}$`, num_dimensions`$^{\text{in}}$`, num_shear`$^{\text{in}}$`)` wandelt eine Matrix aus der internen Repräsentation in die erwartete Größe des aufrufenden Programms um,
- `pure function Export_Tensor(tens`$^{\text{in}}$`, num_dimensions`$^{\text{in}}$`, num_shear`$^{\text{in}}$`)` wandelt einen Tensor aus der internen Repräsentation in die erwartete Größe des aufrufenden Programms um und

- `pure function Import_Matrix(mat`$^{\text{in}}$`, num_dimensions`$^{\text{in}}$`, num_shear`$^{\text{in}}$`)` wandelt eine Matrix des aufrufenden Programms in die interne Repräsentation um.

Die folgenden Funktionen sind nützlich, um Matrixkomponenten in einem Vektor zu speichern oder daraus zu extrahieren:

- `pure function Mat_To_Vec9(mat`$^{\text{in}}$`)` um die Komponenten einer Matrix aus der aktuellen Repräsentation in einen Vektor mit neun Komponenten zu transformieren und
- `pure function Vec9_To_Mat(vec9`$^{\text{in}}$`)` um aus einem Vektor mit neun Matrixkomponenten eine Matrix in der aktuellen Repräsentation zu erzeugen.

Für die Abschätzung der partiellen Ableitungen nach einer Zustandsgröße wie bei der numerischen Berechnung der Jacobi-Matrix werden die folgenden Funktionen genutzt:

- `pure subroutine Ref_Index(ref_idx`$^{\text{in}}$`, idx`$_{\text{out}}$`, jdx`$_{\text{out}}$`)` erwartet einen Laufindex zwischen eins und sechs für die reduzierte Repräsentation `MAT66` oder sonst zwischen eins und neun. Es werden Spalten- und Zeilenindex für eine 3×3-Matrix in der tensoriellen Darstellung zurückgegeben,
- `pure subroutine Perturbate(mat`$^{\text{in}}_{\text{out}}$`, idx`$^{\text{in}}$`, jdx`$^{\text{in}}$`, theta`$^{\text{in}}$`)` erwartet eine Matrix in der aktuellen Repräsentation und addiert den Wert `theta` zu dem Eintrag, der mit den beiden Indizes für eine tensorielle Darstellung beschrieben wird,
- `pure function Get_Matrix_From_Tensorpart(tens`$^{\text{in}}$`, idx`$^{\text{in}}$`, jdx`$^{\text{in}}$`)` extrahiert eine Matrix $M_{kl} = T_{ijkl}\delta_{ij}$ aus dem übergebenen Tensor $\mathcal{T}$ und den Indizes der tensoriellen Darstellung,
- `pure subroutine Set_Matrix_In_Tensorpart(tens`$^{\text{in}}_{\text{out}}$`, idx`$^{\text{in}}$`, jdx`$^{\text{in}}$`, mat`$^{\text{in}}$`)` ist die Umkehroperation und schreibt eine Matrix in den übergebenen Tensor,
- `pure function Get_Elements_From_Matrixlist(matlist`$^{\text{in}}$`, nel`$^{\text{in}}$`, idx`$^{\text{in}}$`, jdx`$^{\text{in}}$`)` extrahiert Einträge aus einer Liste an Matrizen. Aus jeder der `nel` Matrizen wird der Wert zurückgegeben, der durch die Indizes der tensoriellen Darstellung definiert ist und
- `pure subroutine Set_Elements_In_Matrixlist(matlist`$^{\text{in}}_{\text{out}}$`, nel`$^{\text{in}}$`, idx`$^{\text{in}}$`, jdx`$^{\text{in}}$`, elemlist`$^{\text{in}}$`)` ist die Umkehroperation und schreibt in alle Matrizen der Liste einen Wert an die Position, die durch die Indizes der tensoriellen Darstellung definiert ist.

Für die direkte Manipulation einzelner Tensoreinträge ist die folgende Funktion verfügbar:

- `pure subroutine Set_Element_In_Tensor(tens`$^{\text{in}}_{\text{out}}$`, idx`$^{\text{in}}$`, jdx`$^{\text{in}}$`, kdx`$^{\text{in}}$`, ldx`$^{\text{in}}$`, val`$^{\text{in}}$`)` erwartet einen Tensor in der genutzten Repräsentation und schreibt den Wert `val` in die Position von `tens`, die durch die Indizes der tensoriellen Darstellung definiert ist. Wie bei `Perturbate()` können durch entsprechende Laufindizes und mehrere Aufrufe von `Set_Element_In_Tensor()` alle Einträge eines Tensors unabhängig von der gewählten Repräsentation beschrieben werden.

Interne Funktionen abhängig von der gewählten Repräsentationen

Einige Funktionen mit Abhängigkeit zur gewählten Repräsentation nutzen die folgenden Hilfsfunktionen, die nur dort definiert sind (siehe auch Abschnitt 5.2). Nur in den Repräsentationen `MAT66` und `MAT99` sind die folgenden Funktionen definiert:

- `pure function Vec_To_Mat33(vec`$^{\text{in}}$`)` um aus einem Vektor der aktuellen Repräsentation eine 3×3-Matrix zu erzeugen und
- `pure function Mat33_To_Vec(mat33`$^{\text{in}}$`)` um aus einer 3×3-Matrix einen Vektor der aktuellen Repräsentation zu erzeugen. Für Vektoren in der reduzierten Repräsentation wird die symmetrische Version der Matrix gespeichert.

Nur in der direkten Umsetzung werden die folgenden Funktionen verwendet:

- `pure function Mat99_To_Tens(mat`$^{\text{in}}$`)` um aus einer 9×9-Matrix einen S4-Tensor zu erzeugen und
- `pure function Tens_To_Mat99(tens`$^{\text{in}}$`)` um aus einen S4-Tensor eine 9×9-Matrix zu erzeugen.

Stoffmodelle (Modul `Constitutive_Models`)

Basisklasse

Für Stoffmodelle gibt es eine gemeinsame Basisklasse `Constitutive_Model_Baseclass`. Jedes Stoffmodell wird als Erweiterung dieser Basisklasse definiert und hat damit Zugriff auf die Variablen der Basisklasse. In der Basisklasse werden die folgenden Variablen definiert:

- `overall_dt` ist die Dauer des gesamten Zeitschritts, der mit dem Aufruf der Benutzerroutine betrachtet wird,
- `dot_strain` sind die Dehnungsraten für den Zeitschritt,
- `calculate_jacobian` ist ein Wahrheitswert der angibt, ob die Jacobi-Matrix berechnet werden soll,
- `provide_jacobian` gibt an, ob die Jacobi-Matrix vom ausgewählten Stoffmodell bereitgestellt werden kann,
- `direct_variables` ist ein Vektor ähnlich dem Vektor der Statusvariablen für Werte, die für den nächsten Zustand direkt zugewiesen werden sollen und
- `direct_variables_mask` ist ein Vektor mit Wahrheitswerten mit `.True.`, mit dem gesteuert wird, welche Werte aus `direct_variables` nicht in der Fehlerabschätzung bei der Integration berücksichtigt werden sollen.

Die beiden abstrakten Schnittstellen geben zusammen mit dem `deferred`-Attribut vor, dass und wie die Funktionen `Initialize()` und `Get_Dot_State()` in jedem Stoffmodell definiert werden müssen (siehe auch Bild 4.2):

- `subroutine Initialize(this`$^{\text{in}}_{\text{out}}$`, params`$^{\text{in}}$`, calculate_jacobian`$^{\text{in}}$`, firstcall`$^{\text{in}}$`)` initialisiert Materialparameter und andere Variablen der Stoffmodellklasse. Es können auch konstante Werte berechnet (`this%derived...`) oder Überprüfungen durchgeführt werden und
- `function Calculate_Dot_State(this`$^{\text{in}}_{\text{out}}$`, ref_dt`$^{\text{in}}$`, cur_time`$^{\text{in}}$`, cur_state`$^{\text{in}}$`)` berechnet die Spannungsrate und die Änderung aller Statusvariablen auf Basis der aktuellen Statusvariablen, der übergebenen Zeitinformationen und der zwischengespeicherten Variablen der Klasse (v. a. `this%dot_strain`).

In der Basisklasse werden die folgenden Funktionen bereitgestellt:

- `subroutine Base_Initialization(this`$^{\text{in}}_{\text{out}}$`, calculate_jacobian`$^{\text{in}}$`, provide_jacobian`$^{\text{in}}$`)` speichert in den Variablen der Klasse, ob die Jacobi-Matrix für die Schnittstelle berechnet werden soll und ob das gewählte Stoffmodell die Matrix bereitstellen kann,
- `function Get_Dot_State(this`$^{\text{in}}_{\text{out}}$`, ref_dt`$^{\text{in}}$`, cur_time`$^{\text{in}}$`, cur_state`$^{\text{in}}$`)` zum Berechnen der Antwort des Stoffmodells. Diese Funktion ruft intern `Calculate_Dot_State()` des jeweiligen Stoffmodells auf und berechnet zusätzlich die numerische Abschätzung der Jacobi-Matrix (sofern erforderlich), was zusätzliche Auswertungen von `Calculate_Dot_State()` erfordert,
- `subroutine Set_Variables(this`$^{\text{in}}_{\text{out}}$`, overall_dt`$^{\text{in}}$`, dot_strain`$^{\text{in}}$`)` weist die beiden übergebenen Werte den entsprechenden Variablen der Klasse zu,
- `Get_Direct_Variables(this`$^{\text{in}}$`, direct_variables`$_{\text{out}}$`)` gibt die zwischengespeicherten direkten Variablen aus,
- `Get_Direct_Variables_Mask(this`$^{\text{in}}$`)` gibt eine Maske mit Wahrheitswerten aus, die anzeigt, welche Statusvariablen durch direkt gespeicherte Variablen ersetzt werden sollen,
- `subroutine Approximate_Jacobian(this`$^{\text{in}}_{\text{out}}$`, ref_dt`$^{\text{in}}$`, cur_time`$^{\text{in}}$`, cur_state`$^{\text{in}}$`)` schätzt die Jacobi-Matrix für den aktuellen Spannungszustand ohne Dehnungsinkremente ab. Dazu werden zwei Auswertungen von `Calculate_Dot_State()` pro Matrixelement benötigt,
- `subroutine Consistent_Jacobian(this`$^{\text{in}}_{\text{out}}$`, ref_dt`$^{\text{in}}$`, cur_time`$^{\text{in}}$`, cur_state`$^{\text{in}}$`)` berechnet die Jacobi-Matrix für den aktuellen Spannungszustand. Es werden sowohl die Änderungen der Spannungen als auch der Zustandsgrößen berücksichtigt. Nach dem ersten Aufruf von `Calculate_Dot_State()` wird eine zusätzliche Auswertung pro Matrixelement benötigt,
- `pure subroutine Elasticity(this`$^{\text{in}}$`, youngs_modulus`$^{\text{in}}$`, nu`$^{\text{in}}$`, dot_stress`$_{\text{out}}$`, jacobian`$_{\text{out}}$`)` berechnet die Spannungsänderung über ein linear elastisches Stoffmodell, mit übergebenem E-Modul und Querdehnzahl. Die Funktion ist in der Basisklasse implementiert, damit es bspw. als Ersatzmodell von allen Stoffmodellen genutzt werden kann,

- `pure function Is_Valid_Stress_State(stress`$^{\text{in}}$`)` zur Überprüfung, ob nach einem gewählten Kriterium ein gültiger Spannungszustand vorliegt,
- `pure function Get_Factor_F(stress_dless_dev`$^{\text{in}}$`)` berechnet den Faktor F in den beiden Modellen `Hypo-VW96` und `ViHy-Ni03` basierend auf dem normierten Deviator der Spannung und
- `pure subroutine Intergranular_Strain(this`$^{\text{in}}$`, L_mat`$^{\text{in}}$`, fdN_mat`$^{\text{in}}$`, D_vis`$^{\text{in}}$`, hypoplastic`$^{\text{in}}$`, igran_strain`$^{\text{in}}$`, R_max`$^{\text{in}}$`, m_T`$^{\text{in}}$`, m_R`$^{\text{in}}$`, beta_R`$^{\text{in}}$`, chi`$^{\text{in}}$`, dt`$^{\text{in}}$`, dot_stress`$_{\text{out}}$`, dot_igran_strain`$_{\text{out}}$`, jacobian`$_{\text{out}}$`)` ist eine Funktion zur Berechnung der intergranularen Dehnung (bzw. der Rate davon), sowie der dadurch erzeugten Spannungsänderung. Als Übergabewerte werden mehrere Größen aus dem hypoplastischen Modell benötigt.

Hauptklasse

Um ein Stoffmodell zu initialisieren, muss `Select_Constitutive_Model` aus dem Modul `Constitutive_Model_Class` mit dem Namen des gewünschten Stoffmodells und den passenden Argumenten aufgerufen werden:

- `subroutine Select_Constitutive_Model(new_constitutive_model`$_{\text{out}}$`, identifier`$^{\text{in}}$`, params`$^{\text{in}}$`, calculate_jacobian`$^{\text{in}}$`, firstcall`$^{\text{in}}$`)` erzeugt eine Instanz des gewählten Stoffmodells und initialisiert es mit den übergebenen Materialparametern. Außerdem wird überprüft, ob eine Jacobi-Matrix berechnet werden soll und dies der erste Aufruf ist.

Integrationsverfahren (Modul `Solver`)

Basisklasse

Alle Integrationsverfahren werden von der Basisklasse `Solver_Baseclass` abgeleitet, die die folgenden Variablen enthält:

- `name` speichert den Namen des Integrationsverfahrens,
- `stepsize_fixed` enthält einen Wahrheitswert, ist standardmäßig auf `.False.` gesetzt und nur für Integrationsverfahren mit konstanter Schrittweite ohne Fehlerabschätzung auf `.True.`,
- `current_error_assigned` ist ein Wahrheitswert der anzeigt, ob in einem Integrationsschritt eine Fehlerabschätzung durchgeführt worden ist und die Werte in `current_error` durch `Set_Error_Estimate()` zugewiesen worden sind,
- `current_error` speichert den abgeschätzten Fehler aus dem letzten durchgeführten Schritt des jeweiligen Integrationsverfahrens,
- `next_dot_state_assigned` ist ein Wahrheitswert der angibt, ob bei Verfahren mit FSAL-Eigenschaft die letzte Auswertung der Differentialgleichung `next_dot_state`

zugewiesen worden ist und somit als Startwert in der nächsten Iteration genutzt werden kann,

- `next_dot_state` enthält die letzte Auswertung der letzten Iteration bei Verfahren mit FSAL-Eigenschaft,
- `mask_ignored_indices` ist eine lokale Kopie der Maske, welche Statusvariablen aus dem benutzten Stoffmodell als direkte Variablen und somit nicht für die Fehlerabschätzung betrachtet werden sollen,
- `last_integ_success` gibt an, ob der letzte Aufruf von `Integrate()` erfolgreich war,
- `last_num_steps_accepted` enthält die Anzahl der erfolgreichen Integrationsschritte beim letzten Aufruf von `Integrate()`,
- `last_num_steps_rejected` speichert die Anzahl der fehlgeschlagenen Integrationsschritte. Die Summe aus diesem Wert und `last_num_steps_accepted` ist die Anzahl der insgesamt durchgeführten Integrationsschritte beim letzten Aufruf von `Integrate()`,
- `exp_stepgrow` ist der Exponent, mit dem das Verhältnis von neuem zu aktuellem Zeitschritt bei adaptiven Integrationsverfahren bei einer Vergrößerung der Schrittweite potenziert wird. Standardmäßig entspricht der Wert $1/(p+1)$ wie in Gleichung (2.18),
- `max_stepgrow` gibt eine maximale Vergrößerung der Schrittweite vor,
- `exp_stepshrink` ist der Exponent, mit dem das Verhältnis von neuem zu aktuellem Zeitschritts bei adaptiven Integrationsverfahren bei einer Verkleinerung der Schrittweite potenziert wird (entspricht typischerweise $1/p$) und
- `min_stepshrink` gibt eine maximale Verkleinerung der Schrittweite vor.

Über ein `deferred`-Attribut in der Basisklasse wird vorgegeben, dass jedes Integrationsverfahren die Funktion `Integration_Algorithm()` implementieren muss:

- `function Integration_Algorithm(this`$^{\text{in}}_{\text{out}}$`, integ_obj`$^{\text{in}}_{\text{out}}$`, cur_state`$^{\text{in}}$`, dot_state_k1`$^{\text{in}}$`, cur_time`$^{\text{in}}$`, dt`$^{\text{in}}$`)` führt die numerische Integration des übergebenen Stoffmodells (Integrationsobjekt) durch. Es werden die aktuellen Zustandsgrößen und die berechneten Änderungsraten für den Ausgangspunkt benötigt, sowie der aktuelle Zeitpunkt und das Zeitinkrement der Integration.

Die Basisklasse stellt außerdem die folgenden Funktionen bereit:

- `subroutine Initialize(this`$^{\text{in}}_{\text{out}}$`, name`$^{\text{in}}$`, exp_stepgrow`$^{\text{in}}$`, max_stepgrow`$^{\text{in}}$`, exp_stepshrink`$^{\text{in}}$`, min_stepshrink`$^{\text{in}}$`, stepsize_fixed`$^{\text{in}}$`)` führt die Initialisierung eines Integrationsverfahrens durch. Insbesondere werden die übergebenen Werte den gleichnamigen Variablen der Klasse zugewiesen.
- `subroutine Set_State_Mask(this`$^{\text{in}}_{\text{out}}$`, statemask`$^{\text{in}}$`)` erwartet einen Vektor an Wahrheitswerten und speichert ihn im passenden Teil von `mask_ignored_indices`. Alle mit `.True.` markierten Einträge markieren die Felder mit direkten Variablen, die für eine Fehlerabschätzung ignoriert werden. Die übergebene Maske sollte die Ausgabe von `Get_Direct_Variables_Mask()` des Stoffmodells sein,

- `subroutine Set_Error_Estimate(this`$^{\text{in}}_{\text{out}}$`, estimation`$^{\text{in}}$) speichert die aktuelle Fehlerabschätzung eines Integrationsverfahrens in der Variable `current_error` abhängig von den optional durch `mask_ignored_indices` gesetzten Feldern direkter Variablen. Diese Funktion sollte am Ende jeder Integrationsroutine mit Fehlerschätzung aufgerufen werden,
- `function Get_Statistics(this`$^{\text{in}}$) ist eine Hilfsfunktion, um Details über die Anzahl erfolgreicher und fehlgeschlagener Integrationsschritte während des letzten Aufrufs von `Integrate()` zu erhalten und
- `subroutine Integrate(this`$^{\text{in}}_{\text{out}}$`, integ_obj`$^{\text{in}}_{\text{out}}$`, cur_state`$^{\text{in}}$`, cur_time`$^{\text{in}}$`, dt`$^{\text{in}}$`, new_state`$_{\text{out}}$`, jacobian`$_{\text{out}}$`, mod_dt`$^{\text{in}}_{\text{out}}$`, successful_integration`$_{\text{out}}$) ist die Steuerungsfunktion für die numerische Integration (*„stepper“* und *„driver“*).

Hauptklasse

Die Initialisierung eines Integrationsverfahrens erfolgt über `Select_Solver` aus dem Modul `Solver_Class`:

- `subroutine Select_Solver(new_solver`$_{\text{out}}$`, name`$^{\text{in}}$) erzeugt eine Instanz des gewählten Integrationsverfahrens und ruft intern die `Initialize()`-Funktion der Basisklasse mit vorgegebenen Parametern zur Initialisierung auf.

Verarbeitungshilfen (Modul `Custom_Utilities`)

In diesem Modul sind Funktionen zur Vorverarbeitung von Eingangsgrößen definiert, die von mehreren Schnittstellen genutzt werden. Im Gegensatz zu den Funktionen in `General_Settings` ist das Verhalten der Funktionen in diesem Modul abhängig von der jeweiligen Schnittstelle. Es sind die folgenden Funktionen definiert:

- `function Preprocess_State_Matrices(identifier`$^{\text{in}}$`, internname`$^{\text{in}}$`, nstates`$^{\text{in}}$`, states`$^{\text{in}}$`, rotation`$^{\text{in}}$) erwartet den Namen eines Stoffmodells und einer Schnittstelle, um die übergebenen Statusvariablen zu modifizieren. Es wird auch eine Rotationsmatrix erwartet, falls das Bezugskoordinatensystem angepasst werden soll,
- `pure function Postprocess_State_Matrices(identifier`$^{\text{in}}$`, internname`$^{\text{in}}$`, nstates`$^{\text{in}}$`, states`$^{\text{in}}$) erwartet ebenfalls den Namen eines Stoffmodells und einer Schnittstelle, um Anpassungen an den Statusvariablen vor Rückgabe an das aufrufende Programm durchzuführen und
- `pure function Is_First_Call(step`$^{\text{in}}$`, iteration`$^{\text{in}}$) gibt auf Basis des übergebenen Schritts und Iteration einen Wahrheitswert zurück, ob es sich um den ersten Aufruf handelt.[2]

[2] Die aktuelle Implementierung dieser Funktion ist noch ausbaufähig, da je nach Schnittstelle weitere Übergabewerte sinnvoll sind.

Nutzung externer Bibliotheken

Grundsätzlich sind die mathematischen Hilfsfunktionen in der xMat so aufgebaut, dass die tatsächlichen Berechnungen mit wenigen Änderungen auch von externen Bibliotheken wie LAPACK durchgeführt werden können.

LAPACK stellt mehrere Varianten einer Funktion bereit, die abhängig von den Eigenschaften der verwendeten Matrizen eine effiziente Berechnung ermöglichen (siehe Anderson u. a., 1999). Eine schematische Darstellung von LAPACK-Bezeichnungen ist in Bild D.1 zu sehen.

So wird in der xMat mit `LU_Decomposition()` eine LU-Zerlegung und mit `LU_Solve()` ein Algorithmus zur Lösung von linearen Gleichungssystemen realisiert. LAPACK stellt eine LU-Zerlegung mit partieller Pivotisierung für beliebige reellwertige Matrizen bereit, u. a. über die Funktion `DGETRF()`. Aus der zerlegten Matrix kann mit `DGETRS()` ein Gleichungssystem gelöst oder über `DGETRI()` die Inverse berechnet werden.[3]

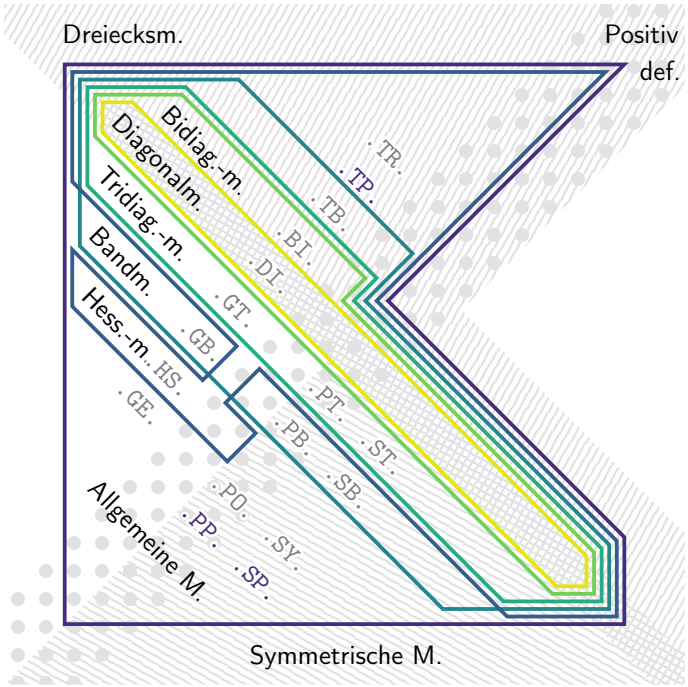


Bild D.1: Schematische Darstellung von LAPACK-Bezeichnungen, die als zweite und dritte Stelle des Funktionsnamens reellwertiger Matrixoperationen genutzt werden. Die erste Stelle ist für reelle Zahlen `S` (einfache Genauigkeit) oder `D` (doppelte Genauigkeit). Damit ergibt sich bspw. der Funktionsname `DSYEV()` für die Bestimmung der Eigenwerte (`...EV`) einer symmetrischen Matrix mit reellen Komponenten in doppelter Genauigkeit. Manche Varianten, wie bspw. `.PP.`, `.SP.` und `.TP.`, sind für angepasste Speicherstrukturen (*„packed storage“*) optimiert. Eine vollständige Übersicht findet sich in Anderson u. a. (1999) oder unter »http://www.netlib.org/lapack/lug/node24.html

[3] Für die Nutzung der LAPACK-Funktionen sind die folgenden Anpassungen erforderlich: Es müssen temporäre Variablen für die Anpassung der Übergabewerte bereitgestellt, der Aufruf der LAPACK-Funktion durchgeführt und der Erfolgsindikator mit den anderen Rückgabewerten verarbeitet werden. Beim Kompilieren der xMat müssen die entsprechenden Bibliotheken mit `-llapack` bereitgestellt und

Eigenwerte und Eigenvektoren von symmetrischen Matrizen können in der xMat mit `Eigendecomposition()` bestimmt werden. In LAPACK kann die Berechnung von Eigenwerten/Eigenvektoren über eine QR-/QL-Zerlegung für reelle und symmetrische Matrizen mit `DSYEV()` erfolgen. Die Funktion `DSTEV()` kann genutzt werden, falls es sich zusätzlich um eine Tridiagonalmatrix handelt.

Es sei angemerkt, dass LAPACK im Allgemeinen für große Matrizen mit mehreren Tausend oder Millionen von Elementen optimiert ist. Da die Matrizen in der xMat vergleichsweise klein sind, ist durch die Nutzung einer externen Bibliothek kein Leistungszuwachs zu erwarten.

Mögliche Probleme und Lösungsvorschläge

Die angegebenen Lösungsvorschläge basieren auf der Nutzung in einem Linux-System. Unter Windows können Anpassungen notwendig sein (z. B. werden die Optionen des Compilers mit einem Schrägstrich statt einem Minuszeichen eingeleitet).

Der Kompiliervorgang schlägt fehl mit der Ausgabe:

```
Illegal character in statement label field
```

- Der Compiler erwartet eine Routine im `fixed`-Format, in der die ersten sechs Zeichen eine besondere Bedeutung haben. Die Übergabe von `-ffree-form` (`gfortran`) bzw. `-free` (`ifort`) ist beim Kompilieren erforderlich. Für Abaqus kann das in der Umgebungsdatei `abaqus_v6.env` eingestellt werden.

Der Kompiliervorgang schlägt fehl mit der Ausgabe:

```
Bad # preprocessor line
```

- Der Compiler trifft auf eine Präprozessor-Anweisung, ohne dass der Präprozessor aktiv ist. Die Übergabe von `-fpp` (`gfortran`) oder `-cpp` (`ifort`) ist notwendig, um den Präprozessor zu nutzen.

Das Kompilieren schlägt beim Verlinken fehl mit einer Ausgabe wie

```
Undefined reference to `...'
```

- Wenn es sich um eine Funktion aus der xMat selbst handelt, dann ist entweder das passende Präprozessor-Makro um die Funktion nicht definiert (`-D<Makro>`) oder die Funktion hat eine aktive `!DEC$`-Direktive und wird direkt aufgerufen (fehlendes `-DNOBIB`).
- Bei externen Funktionen fehlt ggf. der Pfad zu einer Bibliothek (`-L<pfad>`) oder der passende Schalter mit `-l<name>`.

verlinkt werden. Gegebenenfalls wird auch der explizite Pfad zu den LAPACK-Bibliotheken mit `-L<pfad>` benötigt.

Im Rahmen einer Abaqus-Simulation schlägt das Kompilieren der Benutzerroutine mit der folgenden Ausgabe fehl:

```
Abqaqus Error: Problem during linking
```

- Wahrscheinlich sind Makros für zusätzliche Schnittstellen definiert, deren Hilfsfunktionen beim Verlinken nicht gefunden werden. Für eine Abaqus-Simulation sollte entweder `ABQ_STD_CALLING` oder `ABQ_EXP_CALLING` definiert sein.

Eine Abaqus-Simulation bricht ab mit dem folgenden Fehler ab (alternativ auch mit `UMAT` statt `VUMAT`):

```
ERROR: NO VUMAT SUBROUTINE WAS SUPPLIED FOR THE USER MATERIAL NAMED ...
```

- Die xMat nutzt die übergebenen Materialnamen, um eine Zuweisung zum gewünschten Stoffmodell durchzuführen. Die Berechnung bricht ab, falls der Materialname zu keinem implementierten und in `General_Settings` als `setting_id_...` definierten Stoffmodell passt.
- Möglicherweise wird die falsche Schnittstelle genutzt. Deshalb sollte überprüft werden, ob `ABAQUS_CALLING` vom Compiler definiert wird, d. h. in der Umgebungsdatei `abaqus_v6.env` eingetragen ist, und in der xMat das korrekte Präprozessor-Makro für die Simulation verwendet wird (`ABQ_STD_CALLING` oder `ABQ_EXP_CALLING`).

Eine Abaqus-Simulation bricht beim Start mit einem der folgenden unspezifischen Fehler ab und es werden keine `.dat` oder `.sta`-Dateien erstellt. Kompilieren und Linken der xMat funktioniert problemlos:

```
Abaqus Error: Abaqus/Standard Analysis exited with an error
```

bzw.

```
Abaqus Error: Abaqus/Explicit Packager exited with an error
```

- Eine Möglichkeit ist, dass weder `ABQ_STD_CALLING` noch `ABQ_EXP_CALLING` definiert ist und somit die Benutzerroutine nicht richtig in den Simulationsprozess eingebunden werden kann.

Bei der Verwendung anderer Programme oder neuen Programmversionen tritt ein Laufzeitfehler auf oder die berechneten Ergebnisse sind keine lesbaren Zahlenwerte:

- Prüfen, ob die Schnittstelle des aufrufenden Programms auch der implementierten Schnittstelle entspricht. Prinzipiell können sich die Schnittstellendefinitionen ändern und es ist nicht sichergestellt, dass unterschiedliche Versionen miteinander kompatibel sind (siehe z. B. Abschnitt 4.3.1 zu den Abaqus-Schnittstellen).

E Materialparameter und Statusvariablen der Benutzerroutinen

xMat

Alle intern genutzten Parameter der xMat werden in Modul `General_Settings` definiert. Eine Auflistung und Details zu den einzelnen Parametern finden sich in Abschnitt 4.2.1.

Elas-0000

Das elastische Modell der xMat nutzt keine Statusvariablen und erwartet die folgenden Materialparameter:

- 1: Elastizitätsmodul E in kPa und
- 2: Querdehnzahl ν.

Hypo-Wu92

Für die Hypoplastizität nach Wu (1992) wird in der xMat die Porenzahl e als erste und einzige Statusvariable verwendet. Die Materialparameter sind:

- 1: Parameter C_1,
- 2: Parameter C_2,
- 3: Parameter C_3 und
- 4: Parameter C_4.

Hypo-VW96

Das hypoplastische Stoffmodell nach von Wolffersdorff (1996) mit dem Konzept der intergranularen Dehnung nach Niemunis und Herle (1997) nutzt in der xMat die folgenden Statusvariablen:

- 1: Porenzahl e,
- 2 bis 10: intergranulare Dehnung h_{11}, h_{22}, h_{33}, h_{12}, h_{21}, h_{23}, h_{32}, h_{13} und h_{31} sowie
- 11: Ersatzsteifigkeit $E_{\mathrm{rep}} = (J_{1111} + J_{2222} + J_{3333})/3$ in kPa.

Die Materialparameter von `Hypo-VW96` sind:

- 1: Reibungswinkel φ_c in rad,
- 2: Querdehnzahl ν,
- 3: Granulathärte h_s in kPa,
- 4: Exponent n_H,
- 5: Porenzahl e_{d0},
- 6: Porenzahl e_{c0},
- 7: Porenzahl e_{i0},
- 8: Exponent α_H,
- 9: Exponent β_H,
- 10: Faktor m_T,
- 11: Faktor m_R,
- 12: Faktor $R_{\max}$,
- 13: Exponent β_R,
- 14: Exponent χ,
- 15: — und
- 16: Anfangsporenzahl e_{ini}.

Baro-Ko15

Die Barodesie nach Kolymbas (2015) nutzt in der xMat die Porenzahl e als erste und einzige Statusvariable. Die Materialparameter sind:

- 1: Reibungswinkel φ_c in rad,
- 2: Parameter c_2,
- 3: Parameter c_3,
- 4: Parameter c_4,
- 5: Parameter c_5,
- 6: Porenzahl e_{c0} und
- 7: Porenzahl $e_{\min}$.

Baro-Sc18

Mit der Barodesie nach Schranz (2018) wird in der xMat wiederum nur die Porenzahl als erste und einzige Statusvariable verwendet. Die Materialparameter sind:

- 1: Reibungswinkel φ_c in rad,
- 2: Referenzsteifigkeit K_r in kPa,
- 3: Exponent ξ,
- 4: Steifigkeitsverhältnis κ,
- 5: Porenzahl e_{c0},
- 6: Porenzahlverhältnis c_e,
- 7: Parameter c_5,
- 8: Parameter c_6 und
- 9: Parameter c_7.

Baro-Ko21

Die Barodesie nach Kolymbas (2021) hat in der xMat die folgenden Statusvariablen:

- 1: Porenzahl e,
- 2: Anteil Porenzahl $e_c(1)$ und
- 3: Anteil Porenzahl $e_c(2)$ mit $e_c = e_c(1) + e_c(2)$, siehe auch Gleichung (2.12), sowie
- 4 bis 12: Richtung der Dehnungsrate $\vec{\mathbf{D}}$ mit den Komponenten $\vec{D}_{11}$, $\vec{D}_{22}$, $\vec{D}_{33}$, $\vec{D}_{12}$, $\vec{D}_{21}$, $\vec{D}_{23}$, $\vec{D}_{32}$, $\vec{D}_{13}$ und $\vec{D}_{31}$.

Die Materialparameter sind:

- 1: Reibungswinkel φ_c in rad,
- 2: Parameter c_2,
- 3: Parameter c_3,
- 4: Parameter c_4,
- 5: Parameter c_5,
- 6: Parameter κ_1,
- 7: Parameter κ_2,
- 8: Parameter k_1,
- 9: Parameter k_2 und
- 10: Porenzahl e_{c0}.

ViHy-Ni03

Die Viskohypoplastizität nach Niemunis (2003) hat in der xMat die folgenden Statusvariablen:

- 1: Porenzahl e,
- 2 bis 10: intergranulare Dehnung h_{11}, h_{22}, h_{33}, h_{12}, h_{21}, h_{23}, h_{32}, h_{13} und h_{31} sowie
- 11: Ersatzsteifigkeit $E_{\text{rep}} = (J_{1111} + J_{2222} + J_{3333})/3$ in kPa.

Die Materialparameter sind:

- 1: Porenzahl e_{100},
- 2: Querdehnzahl ν,
- 3: Steigung CSL λ,
- 4: Steigung κ,
- 5: Faktor β_b,
- 6: Viskositätsindex I_v,
- 7: Referenzkriechrate D_r,
- 8: Reibungswinkel φ_c in rad,
- 9: —,
- 10: Faktor m_T,
- 11: Faktor m_R,
- 12: Faktor $R_{\max}$,
- 13: Exponent β_R,
- 14: Exponent χ und
- 15: Parameter OCR (aktuell nicht verwendet).

Andere Benutzerroutinen

ABQS02

Intern werden die folgenden Parameter gesetzt:

- `theta` definiert die Störungsgröße bei der numerischen Abschätzung der Jacobi-Matrix $\mathcal{J}$ in `ddsdde` (Vorgabewert: 10^{-7}),
- `nsub` gibt die Anzahl an Schritten für eine Integration mit einem der beiden Euler-Verfahren ohne Richardson-Extrapolation an (Vorgabewert: 100) und
- `hmin` definiert das minimale Subinkrement h_{sub} eines Zeitschritts bei einer Integration mit Richardson-Extrapolation (Vorgabewert: 10^{-10}).

Die folgenden Statusvariablen werden erwartet bzw. übergeben:

- 1: Integrationsverfahren mit den Optionen
  - 1: explizites Euler-Verfahren,
  - 2: implizites Euler-Verfahren,
  - 3: explizites Euler-Verfahren mit Richardson-Extrapolation sowie
  - 4: implizites Euler-Verfahren mit Richardson-Extrapolation,
- 2: Länge h_{sub} des ersten Subinkrements für das Integrationsverfahren,
- 3: mobilisierter Reibungswinkel φ_{mob},
- 4: Anzahl skalarer Statusvariablen n_s,
- 5: Anzahl tensorieller Statusvariablen n_t,
- 6: Index der ersten Toleranzkomponente i_t,
- 7 bis $i_t - 1$: Übergabe der n_s skalaren Statusvariablen (für $n_s > 0$) gefolgt von den n_t tensoriellen Statusvariablen (für $n_t > 0$). Für jede tensorielle Statusvariable werden sechs Komponenten in der Reihenfolge q_{11}, q_{22}, q_{33}, q_{12}, q_{13} und q_{23} erwartet,
- i_t: absolute Toleranz für Spannungskomponenten $t_{A,T}$,
- $i_t + 1$: relative Toleranz für Spannungskomponenten $t_{R,T}$,
- $i_t + 2$: absolute Toleranz für Komponenten der Jacobi-Matrix $t_{A,BG}$,
- $i_t + 3$: relative Toleranz für Komponenten der Jacobi-Matrix $t_{R,BG}$,
- $i_t + 4$: relative Toleranz aller definierten Statusvariablen $t_{R,Q}$,
- $i_t + 5$ bis $i_t + n_s + 4$: die absolute Toleranz der jeweiligen skalaren Statusvariablen $t_{A,Q,t,i}$ $(0 \leq n_s \leq 5)$ und
- $i_t + n_s + 5$ bis $i_t + n_s + n_t + 4$: die absolute Toleranz der jeweiligen tensoriellen Statusvariablen $t_{A,Q,s,i}$ $(0 \leq n_t \leq 3)$.

Die Materialparameter von ABQS02 sind:

- 1: Reibungswinkel φ_c in Grad,
- 2: Granulathärte h_s in kPa,
- 3: Exponent n,
- 4: Porenzahl e_{d0},
- 5: Porenzahl e_{c0},
- 6: Porenzahl e_{i0},
- 7: Exponent α_H,
- 8: Exponent β_H,
- 9: isotroper Druck p_c,
- 10: Faktor $R_{\max}$,
- 11: Faktor m_R,
- 12: Faktor m_T,
- 13: Exponent β_R und
- 14: Exponent χ.

ABQS04

Intern werden die folgenden Parameter verwendet:

- sub definiert einen Steuerwert t_s, der bei der numerischen Integration für das Subinkrement $h_{\text{sub}} = \min(h,\, t_s/||\mathbf{D}||)$ genutzt wird (Vorgabewert: 10^{-5}),
- mu definiert einen Faktor μ_t, der die Zeitschrittgröße bei der Integration der intergranularen Dehnung mit $\mu_t h_{\text{sub}}$ anpasst (Vorgabewert: 0,1, der dem einprogrammierten Wert in Benutzerroutine ABQS09 entspricht),
- ela als Spannung σ_{el}, die zum Umschalten auf das elastische Ersatzmodell verwendet wird (Vorgabewert: $-0{,}01\,\text{kPa}$),
- imp_intg bestimmt, ob die intergranularen Dehnungen implizit, explizit oder gemischt integriert werden sollen (Vorgabewert: 0),
- tan_stiff steuert die Berechnung der Jacobi-Matrix ddsdde (Vorgabewert: 1) über
  - 1: Tangentensteifigkeit oder
  - 2: Sekantensteifigkeit und
- ddsdde_min gibt eine minimale Steifigkeit $E_{\min}$ für das elastische Ersatzmodell vor (Vorgabewert: $10\,\text{kPa}$).

Die folgenden Statusvariablen werden von `ABQS04` erwartet bzw. übergeben:

- 1: aktuelle Porenzahl e,
- 2: volumetrische Dehnung $\varepsilon_V = \varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}$,
- 3: mittlerer Druck $p = (T_{11} + T_{22} + T_{33})/3$ in kPa,
- 4 bis 6: Deviatorspannung q in kPa, mobilisierter Reibungswinkel φ_{mob} in Grad und Winkel atan(T_{22}/T_{21}) in rad, jeweils unter der Annahme eines ebenen Dehnungszustands,
- 7 bis 12: intergranulare Dehnung $\mathbf{h}$ mit bis zu sechs Komponenten in der Reihenfolge h_{11}, h_{22}, h_{33}, h_{12}, h_{13} und h_{23},
- 13: Kapillardruck p_c in kPa,
- 14: Norm des Spannungstensors vor dem Zeitschritt $||\mathbf{T}_n||$ in kPa,
- 15: Indikator für Boden (1) oder Luft (2),
- 16: —,
- 17: —,
- 18: Ersatzsteifigkeit $E_{\text{rep}} = (J_{1111} + J_{2222} + J_{3333})/3$ in kPa und
- 19: Skalierte Norm der intergranularen Dehnung $||\mathbf{h}|| \cdot 10^6$.

Die Materialparameter sind:

- 1: Reibungswinkel φ_c in rad,
- 2: —,
- 3: Granulathärte h_s in kPa,
- 4: Exponent n,
- 5: Porenzahl e_{d0},
- 6: Porenzahl e_{c0},
- 7: Porenzahl e_{i0},
- 8: Exponent α_H,
- 9: Exponent β_H,
- 10: Faktor m_T,
- 11: Faktor m_R,
- 12: Faktor R_{max},
- 13: Exponent β_R,
- 14: Exponent χ,
- 15: — und
- 16: Anfangsporenzahl e_{ini}.

PLX09

Intern werden die folgenden Parameter gesetzt:

- `nasvdim` definiert die maximale Anzahl der Statusvariablen neben den Spannungen (Vorgabewert: 15),
- `tolintT` legt die (absolute) Fehlertoleranz t_A für das adaptive Integrationsverfahren fest (Vorgabewert: 10^{-4}),
- `maxnint` limitiert die maximale Anzahl an Unterschritten für das adaptive Integrationsverfahren (Vorgabewert: 10000),
- `DTmin` definiert das minimale Subinkrement h_{sub} des Zeitschritts der Integration (Vorgabewert: 10^{-17}),
- `perturb` ist der Wert der Störung zur numerischen Berechnung der Jacobi-Matrix `ddsdde` (Vorgabewert: 10^{-5}, wird aber nicht genutzt),
- `nfasv` bezeichnet die Anzahl zusätzlicher Statusvariablen in `statev` (Vorgabewert: 1) und
- `prsw` steuert die Ausgabe von zusätzlichen Informationen (Vorgabewert: `.True.`).

Die folgenden Statusvariablen werden erwartet bzw. übergeben:

- 1 bis 6: intergranulare Dehnung $\mathbf{h}$ in der Reihenfolge h_{11}, h_{22}, h_{33}, h_{12}, h_{13} und h_{23},
- 7: aktuelle Porenzahl e,
- 8: Porenwasserüberdruck u in kPa,
- 9: mittlerer Druck p in kPa,
- 10: Anzahl Aufrufe/Auswertungen der Stoffmodellgleichungen,
- 11: mobilisierter Reibungswinkel φ_{mob} in Grad,
- 12: normalisierte intergranulare Dehnung ρ (siehe Gleichung (2.9)) und
- 13: initiales bzw. zuletzt verwendetes Subinkrement eines Integrationsschritts h_{sub}.

Die Materialparameter sind:

- 1: Reibungswinkel φ_c in Grad,
- 2: isotroper Druck p_c,
- 3: Granulathärte h_s in kPa,
- 4: Exponent n,
- 5: Porenzahl e_{d0},
- 6: Porenzahl e_{c0},
- 7: Porenzahl e_{i0},
- 8: Exponent α_H,
- 9: Exponent β_H,
- 10: Faktor m_R,
- 11: Faktor m_T,
- 12: Faktor $R_{\max}$,
- 13: Exponent β_R,
- 14: Exponent χ und
- 15: Dichte ρ_w in t/m^3.

ABQS09

Intern werden die folgenden Parameter gesetzt:

- `imp_intg` bestimmt, ob die intergranularen Dehnungen implizit, explizit oder gemischt integriert werden sollen (Vorgabewert: 0,5),
- `subincr_length` definiert einen Steuerwert t_s, mit dem das Subinkrement $h_{\text{sub}} = \min(h,\, t_s/||\mathbf{D}||)$ bei der numerischen Integration begrenzt werden kann (Vorgabewert: 10^{-4}),
- `easy_inhomogeneous` legt fest, ob die Jacobi-Matrix `ddsdde` ohne intergranulare Dehnung nach Gleichung (3.4) abgeschätzt wird, oder als $2\mathcal{L}$ (Vorgabewert: `.True.` und somit die erste Option) und
- `W2min` definiert, ob die Arbeit zweiter Ordnung $W_{2,\text{min}}$ berechnet werden soll oder nicht (Vorgabewert: `.False.` und somit nicht berechnen).

Die folgenden Statusvariablen werden erwartet bzw. übergeben:

- 1: aktuelle Porenzahl e,
- 2: Kapillardruck p_c in kPa,
- 3: mobilisierter Reibungswinkel $\sin(\varphi_{\text{mob}})$ unter der Annahme eines ebenen Dehnungszustands,
- 4: Arbeit zweiter Ordnung $W_{2,\text{min}}$,
- 5: —,
- 6: — und
- 7 bis 6+n: intergranulare Dehnung $\mathbf{h}$ mit bis zu sechs Komponenten in der Reihenfolge h_{11}, h_{22}, h_{33}, h_{12}, h_{13} und h_{23}.

Die Materialparameter sind:

- 1: Reibungswinkel φ_c in rad,
- 2: Querdehnzahl ν,
- 3: Granulathärte h_s in kPa,
- 4: Exponent n,
- 5: Porenzahl e_{d0},
- 6: Porenzahl e_{c0},
- 7: Porenzahl e_{i0},
- 8: Exponent α_H,
- 9: Exponent β_H,
- 10: Faktor m_T,
- 11: Faktor m_R,
- 12: Faktor R_{max},
- 13: Exponent β_R und
- 14: Exponent χ.

ABQE10

Diese Routine basiert auf `ABQS09` und hat die gleichen internen Parameter. Die folgenden Vorgabewerte unterschieden sich:

- Die intergranulare Dehnung wird in der Standardeinstellung (`imp_intg` = 0) explizit integriert und
- der Steuerwert t_s für das Subinkrement ist auf 10^{-5} reduziert.

Die folgenden Statusvariablen werden erwartet bzw. übergeben:

- 1: aktuelle Porenzahl e,
- 2: Kapillardruck p_c in kPa,
- 3: mobilisierter Reibungswinkel $\sin(\varphi_{\mathrm{mob}})$ unter der Annahme eines ebenen Dehnungszustands,
- 4: —,
- 5: —,
- 6: —,
- 7 bis 6+n: intergranulare Dehnung $\mathbf{h}$ mit bis zu sechs Komponenten in der Reihenfolge h_{11}, h_{22}, h_{33}, h_{12}, h_{13} und h_{23},
- 13: Porendruck p_p,
- 14 bis 16: D_{11}, D_{22} sowie D_{33} und
- 17: Ersatzsteifigkeit $K = |\sum_{i=1}^{3}(T_{ii,n+1} - T_{ii,n})/(3D_{ii})|$ in kPa.

Die Materialparameter sind:

- 1: Reibungswinkel φ_c in rad,
- 2: Querdehnzahl ν,
- 3: Granulathärte h_s in kPa,
- 4: Exponent n,
- 5: Porenzahl e_{d0},
- 6: Porenzahl e_{c0},
- 7: Porenzahl e_{i0},
- 8: Exponent α_H,
- 9: Exponent β_H,
- 10: Faktor m_T,
- 11: Faktor m_R,
- 12: Faktor R_{max},
- 13: Exponent β_R,
- 14: Exponent χ,
- 15: Anfangsporenzahl e_{ini} und
- 16: Dichte ρ_w in t/m^3.

F Beispiele zur Nutzung der xMat

In diesem Abschnitt werden verschiedene Testprogramme und Skripte aufgelistet, mit denen die xMat aus verschiedenen Programmen und Programmiersprachen heraus genutzt werden kann.

Fortran-Programm

Der Quellcode in Code F.1 zeigt den Aufruf von `xmat_console()` und `Dot_Values()` aus einem Fortran-Programm. Für die anderen Schnittstellen ist die Vorgehensweise ähnlich. Bei der `UMAT`-Schnittstelle muss zusätzlich auf die andere Reihenfolge der Komponenten und die doppelt so großen Scherdehnungen geachtet werden (siehe Abschnitt 4.3.1). Hinweise zum Kompilieren finden sich in Code F.2.

Code F.1: Fortran-Testprogramm für den Aufruf der xMat.

```
program xmat_tester
   implicit none

   integer, parameter :: dp = selected_real_kind(15)
   character(len=80) :: materialname
   real(dp), dimension(16) :: materialparameters
   real(dp), dimension(20) :: oldstate, newstate, dotstate
   real(dp), dimension(6) :: initial_stress, stress, strain, dotstress, newstress
   real(dp), dimension(6, 6) :: jacobian
   real(dp) :: timeincrement, totaltime, voidratio

   initial_stress = [-100.0_dp, -250.0_dp, -250.0_dp, 50.0_dp, 10.0_dp, 0.0_dp]
   strain = [-0.00005_dp, 0.000015_dp, 0.000015_dp, 0.0_dp, 0.0_dp, 0.0_dp]
   voidratio = 0.8_dp
   timeincrement = 0.001_dp
   totaltime = 0.0_dp

   materialname = 'HYPO-VW96_Test'
   materialparameters = [0.5777_dp, 0.0_dp, 4000000.0_dp, 0.27_dp, &
                         0.677_dp, 1.054_dp, 1.212_dp, 0.14_dp, 2.5_dp, &
                         1.1_dp, 2.2_dp, 0.0001_dp, 0.1_dp, 5.5_dp, 0.0_dp, voidratio]
   oldstate = 0.0_dp
   oldstate(1) = voidratio

   call Dot_Values(materialname, size(materialparameters), materialparameters, &
      size(oldstate), oldstate, size(strain), initial_stress, strain, timeincrement, totaltime, &
      dotstress, dotstate, jacobian)

   call printvector(vector=dotstress, name='dotstress')
   call printvector(vector=dotstate, name='dotstate')
   call printmatrix(matrix=jacobian, name='jacobian')

```

```
   call xmat_console(materialname, size(materialparameters), materialparameters, &
      size(oldstate), oldstate, size(strain), initial_stress, strain, timeincrement, totaltime, &
      newstress, newstate, jacobian)

   call printvector(vector=newstress, name='newstress')
   call printvector(vector=newstate, name='newstate')
   call printmatrix(matrix=jacobian, name='jacobian')


   contains


   ! ----------------------------------------------------------------------------------------------
   subroutine printvector(vector, name)
   ! ----------------------------------------------------------------------------------------------
      implicit none

      real(dp), dimension(:), intent(in) :: vector
      character(len=*), intent(in) :: name
      character(len=16) :: formatstring

      formatstring = '(a,a,  f16.7)'
      write(formatstring(6:7), fmt='(i2)') size(vector)
      write(*, formatstring) name, ' = ', vector
   end subroutine printvector


   ! ----------------------------------------------------------------------------------------------
   subroutine printmatrix(matrix, name)
   ! ----------------------------------------------------------------------------------------------
      implicit none

      real(dp), dimension(:, :), intent(in) :: matrix
      character(len=*), intent(in) :: name
      character(len=16) :: formatstring
      integer :: idx

      write(*, '(a,a)') name, ' = '
      formatstring = '(a,i2,a,  f16.7)'
      write(formatstring(9:10), fmt='(i2)') size(matrix, 2)

      do idx = 1, size(matrix, 1)
         write(*, formatstring) '(', idx, ', :) ', matrix(idx, :)
      end do
      write(*, '(a)') ''
   end subroutine printmatrix
end program xmat_tester
```

Code F.2: Makefile zum Kompilieren der xMat mit dem Testprogramm `tester.f` unter Linux.

```
tester_gfortran: xmat.f tester.f
	gfortran -ffree-form -cpp $^ -o $@;
	-rm *.mod;

tester_intel: xmat.f tester.f
	ifort -free -fpp $^ -o $@;
	-rm *.mod;
```

C-Programm

Der Quellcode in Code F.3 zeigt die Anbindung der Funktion `Dot_Values_CInter()` in einem beispielhaften C-Testprogramm. Für die Nutzung von `xmat_console_CInter()` sind nur minimale Anpassungen erforderlich. Unter Linux kann bspw. ein `Makefile` im gleichen Verzeichnis wie `xmat.f` und `dot_values_cinter.c` erstellt werden mit einem ähnlichen Inhalt wie in Code F.4. Unter Windows kann bspw. eine Batchdatei genutzt werden, deren Inhalt sich an Code F.5 orientiert.

Code F.3: C-Testprogramm `dot_values_cinter_tester.c` für den Aufruf der xMat.

```c
#include <stdio.h>
#include <stdlib.h>

extern void Dot_Values_CInter(const char *materialname, const int *nparams,
   double materialparameters[], const int *nstatevar, double statevariables[],
   const int *ncomponents, double oldstress[], double oldstrain[], double timeincrement[],
   double totaltime[], double dotstress[], double dotstate[], double jacobian[]);

extern void xmat_console_CInter(const char *materialname, const int *nparams,
   double materialparameters[], const int *nstatevar, double statevariables[],
   const int *ncomponents, double oldstress[], double oldstrain[], double timeincrement[],
   double totaltime[], double newstress[], double newstate[], double jacobian[]);


typedef enum {
   num_components = 6,
   num_materialparameters = 16,
   num_statevariables = 20
} fixedLengths;


double *MemoryAllocationAndZeroArray(const int num_elements) {
   double *mem_newarray = calloc(num_elements, sizeof(double));
   return mem_newarray;
}

double *MemoryAllocationAndArrayAssignment(const int num_elements, double *originaldata) {
   double *mem_newarray = malloc(num_elements*sizeof(double));
   for (int idx = 0; idx < num_elements; idx++) {
      mem_newarray[idx] = originaldata[idx];
   }
   return mem_newarray;
}


int main() {
   char materialname[80] = "HYPO-VW96_Test      ";

   double p_val_materialparameters[num_materialparameters] = {0.5777, 0.0, 4000000.0, 0.27, 0.677,
      1.054, 1.212, 0.14, 2.5, 1.1, 2.2, 0.0001, 0.1, 5.5, 0.0, 0.8};
   double p_val_statevariables[num_statevariables] = {0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
   double p_val_oldstress[num_components] = {-100.0, -250.0, -250.0, 50.0, 10.0, 0.0};
   double p_val_oldstrain[num_components] = {-0.00005, 0.000015, 0.000015, 0.0, 0.0, 0.0};

   double p_timeincrement = 0.001;
   double p_totaltime = 0.0;

```

```
49    // Save array length as variables (so they can be referenced by an address)
50    int r_num_materialparameters = num_materialparameters;
51    int r_num_statevariables = num_statevariables;
52    int r_num_components = num_components;
53
54    // Allocate contiguous memory for input and output arrays
55    double *p_materialparameters = MemoryAllocationAndArrayAssignment(num_materialparameters,
56       p_val_materialparameters);
57    double *p_statevariables = MemoryAllocationAndArrayAssignment(num_statevariables,
58       p_val_statevariables);
59    double *p_oldstress = MemoryAllocationAndArrayAssignment(num_components, p_val_oldstress);
60    double *p_oldstrain = MemoryAllocationAndArrayAssignment(num_components, p_val_oldstrain);
61
62    double *p_dotstress = MemoryAllocationAndZeroArray(num_components);
63    double *p_dotstate = MemoryAllocationAndZeroArray(num_statevariables);
64    double *p_jacobian = MemoryAllocationAndZeroArray(num_components*num_components);
65
66    // Call the interface function and do the calculation
67    Dot_Values_CInter(materialname, &r_num_materialparameters, p_materialparameters,
68       &r_num_statevariables, p_statevariables, &r_num_components, p_oldstress, p_oldstrain,
69       &p_timeincrement, &p_totaltime, p_dotstress, p_dotstate, p_jacobian);
70
71    printf("dotstress = ");
72    for (int idx = 0; idx < num_components; idx++) {
73       printf("%16.7f", p_dotstress[idx]);
74    }
75    printf("\n");
76
77    printf("dotstate = ");
78    for (int idx = 0; idx < num_statevariables; idx++) {
79       printf("%16.7f", p_dotstate[idx]);
80    }
81    printf("\n");
82
83    printf("jacobian = ");
84    for (int idx = 0; idx < num_components*num_components; idx++) {
85       if ((idx % num_components) == 0) {
86          printf("\n( %i, :) ", (int)(1+idx/num_components));
87       }
88       printf("%16.7f", p_jacobian[idx]);
89    }
90    printf("\n\n");
91
92    double *p_newstress = p_dotstress;
93    double *p_newstate = p_dotstate;
94
95    xmat_console_CInter(materialname, &r_num_materialparameters, p_materialparameters,
96       &r_num_statevariables, p_statevariables, &r_num_components, p_oldstress, p_oldstrain,
97       &p_timeincrement, &p_totaltime, p_newstress, p_newstate, p_jacobian);
98
99    printf("newstress = ");
100   for (int idx = 0; idx < num_components; idx++) {
101      printf("%16.7f", p_newstress[idx]);
102   }
103   printf("\n");
104
105   printf("newstate = ");
106   for (int idx = 0; idx < num_statevariables; idx++) {
107      printf("%16.7f", p_newstate[idx]);
108   }
109   printf("\n");
```

```
110
111    printf("jacobian = ");
112    for (int idx = 0; idx < num_components*num_components; idx++) {
113       if ((idx % num_components) == 0) {
114          printf("\n( %i, :) ", (int)(1+idx/num_components));
115       }
116       printf("%16.7f", p_jacobian[idx]);
117    }
118    printf("\n\n");
119
120    // Free allocated memory
121    free(p_materialparameters);
122    free(p_statevariables);
123    free(p_oldstress);
124    free(p_oldstrain);
125    free(p_dotstress);
126    free(p_dotstate);
127    free(p_jacobian);
128
129    return 0;
130 }
```

Code F.4: Makefile zur Erzeugung des C-Programms namens `tester_gfortran` bzw. `tester_ifort` aus `tester.c` und `xmat.f` mit dem GNU-Compiler oder dem Intel-Compiler unter Linux.

```
1  tester_gfortran: xmat.f tester.c
2         gfortran -ffree-form -cpp -DCINTER -DNOBIB $< -c;
3         gcc $(^:.f=.o) -o $@ -lgfortran -lm;
4         -rm *.mod xmat.o;
5
6  tester_ifort: xmat.f tester.c
7         ifort -free -fpp -DCINTER -DNOBIB $< -c;
8         icc $(^:.f=.o) -o $@ -lifcore;
9         -rm *.mod xmat.o;
```

Code F.5: Batch-Datei zur Erzeugung von `dot_values_cinter_tester.exe` aus `dot_values_cinter_tester.c` und `xmat.f` mit `ifort` unter Windows mit Visual Studio 2019 und dem Intel oneAPI-Toolkit. Der Pfad zu `vars.bat` in Zeile 2 muss im Allgemeinen noch angepasst werden.

```
1  @echo off
2  @call "C:\Program Files (x86)\Intel\oneAPI\compiler\2021.4.0\env\vars.bat" intel64 vs2019
3  ifort /free /fpp /DCINTER xmat.f /c
4  cl xmat.obj tester.c /link /out:tester.exe
5  del *.mod *.obj *.exp *.lib
6  pause
```

MATLAB-Skript

Um die xMat aus MATLAB heraus zu nutzen, muss sie zuerst als MEX-Datei kompiliert werden (siehe Code F.7). Ein einfaches Skript, um die `mexFunction`-Schnittstelle zu nutzen, ist in Code F.6 zu sehen.

Code F.6: MATLAB-Skript zum Aufruf der als MEX-Datei kompilierten xMat über die `mexFunction`-Schnittstelle.

```
materialname = 'HYPO-VW96_Test';
materialparameters = [0.5777, 0.0, 4000000.0, 0.27, 0.677, 1.054, 1.212, 0.14, 2.5, ...
                      1.1, 2.2, 0.0001, 0.1, 5.5, 0.0, 0.8];
statevariables = [0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];
oldstress = [-100.0, -250.0, -250.0, 50.0, 10.0, 0.0];
oldstrain = [-0.00005, 0.000015, 0.000015, 0.0, 0.0, 0.0];
timeincrement = 0.001;
totaltime = 0.0;

start=tic();
[newstress, newstate, jacobian] = xmat(materialname, materialparameters, statevariables, ...
                                       oldstress, oldstrain, timeincrement, totaltime);
toc(start)

fid = fopen('ausgabe', 'w');
fprintf(fid, 'newstress = ');
fprintf(fid, '%16.7f', newstress);
fprintf(fid, '\n');

fprintf(fid, 'newstate = ');
fprintf(fid, '%16.7f', newstate);
fprintf(fid, '\n');

fprintf(fid, 'jacobian = \n');
for idx = 1:6
   fprintf(fid, '( %i, :) ', idx);
   fprintf(fid, '%16.7f', jacobian(((idx-1)*6+1):6*idx));
   fprintf(fid, '\n');
end
fprintf(fid, '\n');
fclose(fid);
```

Code F.7: MATLAB-Befehl zum Kompilieren der `xmat.f` als MEX-Datei (unter Windows), auf das aus MATLAB zugegriffen werden kann (Intel-Compiler unter Windows). Unter Linux sollte `-free -fpp` verwendet werden. Mit dem GCC sind die Flags stattdessen `-ffree-form -cpp`.

```
mex 'COMPFLAGS="$COMPFLAGS /free /fpp"' -largeArrayDims -DMATLAB_CALLING xmat.f
delete *.mod;
```

GNU Octave-Skript

Die Nutzung aus GNU Octave erfordert ebenfalls ein Vorkompilieren. Im Gegensatz zu MATLAB wird intern nicht die `mexFunction`-Schnittstelle, sondern die C-Schnittstelle genutzt. Zum Erzeugen einer OCT-Datei wird eine C++-Hilfsdatei wie in Code F.8 benötigt. Die Befehle zum Kompilieren der Dateien sind in Code F.9 dargestellt. Für die Nutzung kann das MATLAB-Skript aus Code F.6 genutzt werden, in dem nur der Aufruf `xmat()` durch `xmat_oct()` ersetzt werden muss.

Code F.8: C++-Hilfsdatei `xmat_oct.cpp` zum Kompilieren der xMat als OCT-Datei. Aktuell wird nur die Funktion `Dot_Values_CInter()` der xMat genutzt, was bei Bedarf mit `xmat_console_CInter()` ersetzt werden kann. Das Makro `DEFUN_DLD` definiert die Funktion `xmat_oct`, die aus GNU Octave genutzt werden kann.

```
#include <octave/oct.h>


extern "C" {
   void xmat_console_CInter(const char *materialname, const int *nparams,
      double materialparameters[], const int *nstatevar, double statevariables[],
      const int *ncomponents, double oldstress[], double oldstrain[], double timeincrement[],
      double totaltime[], double newstress[], double newstate[], double jacobian[]);
}


DEFUN_DLD(xmat_oct, args, , "Calling xMat from Octave") {
   octave_value_list retval;
   if (args.length () != 7) {
      print_usage();
      return retval;
   }

   std::string s_materialname = args(0).string_value();
   const char* materialname = s_materialname.c_str();

   NDArray array_materialparameters = args(1).array_value();
   octave_idx_type o_num_materialparameters = array_materialparameters.numel();
   int r_num_materialparameters = o_num_materialparameters;
   double *p_materialparameters = array_materialparameters.fortran_vec();

   NDArray array_statevariables = args(2).array_value();
   octave_idx_type o_num_statevariables = array_statevariables.numel();
   int num_statevariables = o_num_statevariables;
   double *p_statevariables = array_statevariables.fortran_vec();

   NDArray array_oldstress = args(3).array_value();
   octave_idx_type o_num_components = array_oldstress.numel();
   int num_components = o_num_components;
   double *p_oldstress = array_oldstress.fortran_vec();

   NDArray array_oldstrain = args(4).array_value();
   double *p_oldstrain = array_oldstrain.fortran_vec();

   octave_value val_timeincrement = args(5);
   double p_timeincrement = val_timeincrement.double_value();

   octave_value val_totaltime = args(6);
   double p_totaltime = val_totaltime.double_value();
```

```
45
46    OCTAVE_LOCAL_BUFFER(double, p_newstress, num_components);
47    OCTAVE_LOCAL_BUFFER(double, p_newstate, num_statevariables);
48    OCTAVE_LOCAL_BUFFER(double, p_jacobian, num_components*num_components);
49
50    // Call the interface function and do the calculation
51    xmat_console_CInter(materialname, &r_num_materialparameters, p_materialparameters,
52       &num_statevariables, p_statevariables, &num_components, p_oldstress, p_oldstrain,
53       &p_timeincrement, &p_totaltime, p_newstress, p_newstate, p_jacobian);
54
55    dim_vector dim_stress(num_components, 1);
56    NDArray newstress(dim_stress);
57    for (int idx = 0; idx < num_components; idx++) {
58       newstress(idx) = p_newstress[idx];
59    }
60    retval.append(newstress);
61
62    dim_vector dim_states(num_statevariables, 1);
63    NDArray newstate(dim_states);
64    for (int idx = 0; idx < num_statevariables; idx++) {
65       newstate(idx) = p_newstate[idx];
66    }
67    retval.append(newstate);
68
69    dim_vector dim_jacobi(num_components, num_components);
70    NDArray jacobian(dim_jacobi);
71    for (int idx = 0; idx < num_components; idx++) {
72       for (int jdx = 0; jdx < num_components; jdx++) {
73          // Transpose matrix while assigning components back
74          jacobian(jdx, idx) = p_jacobian[idx*num_components+jdx];
75       }
76    }
77    retval.append(jacobian);
78
79    return retval;
80 }
```

Code F.9: Kompilieren der `xmat.f` und der Hilfsdatei `xmat_oct.cpp` als OCT-Datei (unter Linux). Der Pfad zu den Octave-Bibliotheken `-L...` ist abhängig von der Octave-Installation.

```
1 mkoctfile -ffree-form -cpp -DOCTAVE_CALLING xmat.f -c;
2 delete *.mod;
3 mkoctfile -L/usr/lib64/octave/6.4.0 -o xmat_oct xmat.o xmat_oct.cpp;
```

Anbindung an Simulink

Für die Nutzung der xMat in Simulink muss zuerst eine passende S-Funktion erzeugt und kompiliert werden. Code F.10 zeigt eine solche C MEX S-Funktion, um auf die Funktion `Dot_Values_CInter()` zuzugreifen. Das Skript in Code F.11 kann benutzt werden, um die S-Funktion aus MATLAB heraus zu erzeugen. Anschließend kann die xMat S-Funktion als Block in einem Modell genutzt werden. Code F.12 erzeugt ein Blockdiagramm ähnlich wie in Bild 4.5. Ein Skript zur Nutzung des Modells findet sich in Code F.13.

Code F.10: S-Funktion `xmat_sl.c` (geschrieben in C) für Simulink, um die Funktionalität der xMat als Simulink-Block verfügbar zu machen.

```c
#define S_FUNCTION_NAME  xmat_sl
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

#define MDL_SET_INPUT_PORT_DIMENSION_INFO
#define MDL_SET_OUTPUT_PORT_DIMENSION_INFO
#define MDL_SET_DEFAULT_PORT_DIMENSION_INFO


extern void Dot_Values_CInter(const char *materialname, const int *nparams,
   double materialparameters[], const int *nstatevar, double statevariables[],
   const int *ncomponents, double oldstress[], double oldstrain[], double timeincrement[],
   double totaltime[], double dotstress[], double dotstate[], double jacobian[]);


typedef enum {
   idx_dotstress = 0,
   idx_dotstate,
   idx_jacobian,
   num_outputparams
} outputParam;


typedef enum {
   idx_materialname = 0,
   idx_materialparameters,
   idx_statevariables,
   idx_oldstress,
   idx_oldstrain,
   idx_timeincrement,
   idx_totaltime,
   num_inputparams
} inputParam;


// Currently fixed port widths are used
typedef enum {
   num_components = 6,
   num_materialparameters = 16,
   num_statevariables = 20
} fixedPortWidths;


static void mdlInitializeSizes(SimStruct *S) {
   // Set the number of input ports
   if (!ssSetNumInputPorts(S, num_inputparams)) {
      return;
   }
   // Set the number of output ports
   if (!ssSetNumOutputPorts(S, num_outputparams)) {
      return;
   }
   // Number of expected parameters
   ssSetNumSFcnParams(S, 0);

   int in_dataportwidth[num_inputparams] = {DYNAMICALLY_SIZED, num_materialparameters,
      num_statevariables, num_components, num_components, 1, 1};
   int out_dataportwidth[num_inputparams] = {num_components, num_statevariables,
```

```
60          num_components*num_components};
61
62      // Set the number of work vectors (contiguous allocated memory provided for Fortran output values)
63      ssSetNumDWork(S, num_outputparams);
64      for (int idx = 0; idx < num_outputparams; idx++) {
65         ssSetDWorkWidth(S, idx, out_dataportwidth[idx]);
66         ssSetDWorkDataType(S, idx, SS_DOUBLE);
67      }
68
69      // Prepare all input ports
70      for (int idx = 0; idx < num_inputparams; idx++) {
71         if (idx == idx_materialname) {
72            ssSetInputPortDataType(S, idx, SS_UINT8);
73         }
74         else {
75            ssSetInputPortDataType(S, idx, SS_DOUBLE);
76         }
77         ssSetInputPortWidth(S, idx, in_dataportwidth[idx]);
78         ssSetInputPortComplexSignal(S, idx, COMPLEX_NO);
79         ssSetInputPortDirectFeedThrough(S, idx, 1);
80         ssSetInputPortAcceptExprInRTW(S, idx, 0);
81         ssSetInputPortOverWritable(S, idx, 0);
82         ssSetInputPortOptimOpts(S, idx, SS_REUSABLE_AND_LOCAL);
83         ssSetInputPortRequiredContiguous(S, idx, 1);
84      }
85
86      // Prepare all output ports
87      for (int idx = 0; idx < num_outputparams; idx++) {
88         ssSetOutputPortDataType(S, idx, SS_DOUBLE);
89         ssSetOutputPortWidth(S, idx, out_dataportwidth[idx]);
90         ssSetOutputPortComplexSignal(S, idx, COMPLEX_NO);
91         ssSetOutputPortOptimOpts(S, idx, SS_REUSABLE_AND_LOCAL);
92         ssSetOutputPortOutputExprInRTW(S, idx, 0);
93      }
94
95      // Register reserved identifiers to avoid name conflict
96      if (ssRTWGenIsCodeGen(S) || ssGetSimMode(S)==SS_SIMMODE_EXTERNAL) {
97         ssRegMdlInfo(S, "dot_values", MDL_INFO_ID_RESERVED, 0, 0, ssGetPath(S));
98      }
99
100     // This S-function can be used in referenced model simulating in normal mode
101     ssSetModelReferenceNormalModeSupport(S, MDL_START_AND_MDL_PROCESS_PARAMS_OK);
102
103     // Set the number of sample time
104     ssSetNumSampleTimes(S, 1);
105
106     // Set the compliance for the operating point save/restore.
107     ssSetOperatingPointCompliance(S, USE_DEFAULT_OPERATING_POINT);
108
109     ssSetArrayLayoutForCodeGen(S, SS_ALL);
110
111     // Set the Simulink version this S-Function has been generated in
112     ssSetSimulinkVersionGeneratedIn(S, "10.0");
113
114     // All options are documented in matlabroot/simulink/include/simstruc.h
115     ssSetOptions(S,
116        SS_OPTION_USE_TLC_WITH_ACCELERATOR |
117        SS_OPTION_CAN_BE_CALLED_CONDITIONALLY |
118        SS_OPTION_EXCEPTION_FREE_CODE |
119        SS_OPTION_WORKS_WITH_CODE_REUSE |
120        SS_OPTION_SFUNCTION_INLINED_FOR_RTW |
```

```
121         SS_OPTION_DISALLOW_CONSTANT_SAMPLE_TIME
122     );
123 }
124 
125 
126 static void mdlInitializeSampleTimes(SimStruct *S) {
127     ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
128     ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);
129 
130     #if defined(ssSetModelReferenceSampleTimeDefaultInheritance)
131     ssSetModelReferenceSampleTimeDefaultInheritance(S);
132     #endif
133 }
134 
135 
136 static void mdlSetInputPortDimensionInfo(SimStruct *S, int_T portIndex, const DimsInfo_T *dimsInfo) {
137     // Set input port dimension
138     if (!ssSetInputPortDimensionInfo(S, portIndex, dimsInfo)) {
139         return;
140     }
141 }
142 
143 
144 static void mdlSetOutputPortDimensionInfo(SimStruct *S, int_T portIndex, const DimsInfo_T *dimsInfo) {
145     // Set output port dimension
146     if(!ssSetOutputPortDimensionInfo(S, portIndex, dimsInfo)) {
147         return;
148     }
149 }
150 
151 
152 static void mdlSetDefaultPortDimensionInfo(SimStruct *S) {
153     // Default dimension for all input ports
154     for (int idx = 1; idx <= num_inputparams; idx++) {
155         if (ssGetInputPortWidth(S, idx) == DYNAMICALLY_SIZED) {
156             ssSetInputPortWidth(S, idx, 1);
157         }
158     }
159 }
160 
161 
162 static void mdlStart(SimStruct *S) {
163     // Allocate the memory availabe for DWork data and initialize it with zeros
164     double* mem_dotstress = (double*) ssGetDWork(S, 0);
165     for (int idx = 0; idx < num_components; idx++) {
166         mem_dotstress[idx] = 0.0;
167     }
168     double* mem_dotstate = (double*) ssGetDWork(S, 1);
169     for (int idx = 0; idx < num_statevariables; idx++) {
170         mem_dotstate[idx] = 0.0;
171     }
172     double* mem_jacobian = (double*) ssGetDWork(S, 2);
173     for (int idx = 0; idx < num_components*num_components; idx++) {
174         mem_jacobian[idx] = 0.0;
175     }
176 }
177 
178 
179 static void mdlOutputs(SimStruct *S, int_T tid) {
180     // Get access to input, output and DWork data
181     char* f_materialname = (char*) ssGetInputPortSignal(S, idx_materialname);
```

```
    double* p_materialparameters = (double*) ssGetInputPortSignal(S, idx_materialparameters);
    double* p_statevariables = (double*) ssGetInputPortSignal(S, idx_statevariables);
    double* p_oldstress = (double*) ssGetInputPortSignal(S, idx_oldstress);
    double* p_oldstrain = (double*) ssGetInputPortSignal(S, idx_oldstrain);
    double* p_timeincrement = (double*) ssGetInputPortSignal(S, idx_timeincrement);
    double* p_totaltime = (double*) ssGetInputPortSignal(S, idx_totaltime);

    double* p_dotstress = (double*) ssGetOutputPortSignal(S, idx_dotstress);
    double* p_dotstate = (double*) ssGetOutputPortSignal(S, idx_dotstate);
    double* p_jacobian = (double*) ssGetOutputPortSignal(S, idx_jacobian);

    double* mem_dotstress = (double*) ssGetDWork(S, 0);
    double* mem_dotstate = (double*) ssGetDWork(S, 1);
    double* mem_jacobian = (double*) ssGetDWork(S, 2);

    // Save array length as variables (so they have an address as a reference)
    int r_num_materialparameters = num_materialparameters;
    int r_num_statevariables = num_statevariables;
    int r_num_components = num_components;

    // Call the xMat interface function
    Dot_Values_CInter(f_materialname, &r_num_materialparameters, p_materialparameters,
       &r_num_statevariables, p_statevariables, &r_num_components, p_oldstress, p_oldstrain,
       p_timeincrement, p_totaltime, mem_dotstress, mem_dotstate, mem_jacobian);

    // Reassign output values
    for (int idx = 0; idx < num_components; idx++) {
       p_dotstress[idx] = mem_dotstress[idx];
    }
    for (int idx = 0; idx < num_statevariables; idx++) {
       p_dotstate[idx] = mem_dotstate[idx];
    }
    for (int idx = 0; idx < num_components*num_components; idx++) {
       p_jacobian[idx] = mem_jacobian[idx];
    }
}


static void mdlTerminate(SimStruct *S) {
    // This method must exist, but nothing has to be done for this block
}


#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Code F.11: Kleines MATLAB-Skript, um `xmat.f` und `xmat_sl.c` zu kompilieren und als S-Funktion bereitzustellen. Dazu muss MATLAB auf einen C- und einen Fortran-Compiler zugreifen können. Das Skript nimmt eine Windows-Umgebung und Intel-Compiler an.

```
%% Compile xmat.f and wrapper xmat_sl.c to S-Function MEX file
mex 'COMPFLAGS="$COMPFLAGS /free /fpp"' -largeArrayDims -DMATLAB_CALLING -DCINTER xmat.f -c
mex -largeArrayDims xmat.obj xmat_sl.c
delete *.mod xmat.obj;
```

Code F.12: Skript zur Erstellung eines einfachen Simulink-Modells, das die xMat S-Funktion benutzt. Einige Befehle und die genaue Angabe der Positionen sind nur für eine Darstellung als Blockdiagramm und nicht für die Funktionalität erforderlich.

```
%% Create a new Simulink system xmat_test
% If any system named xmat_test is open, close it without saving any changes.
% Also remove xmat_test model file if it exists
if bdIsLoaded('xmat_test')
   bdclose('xmat_test')
end
if exist('xmat_test.slx', 'file')
   delete('xmat_test.slx')
end

materialname = '"HYPO-VW96_Test"';
materialparameters = [0.5777, 0.0, 4000000.0, 0.27, 0.677, 1.054, 1.212, 0.14, 2.5, ...
                      1.1, 2.2, 0.0001, 0.1, 5.5, 0.0, 0.8];
statevariables = [0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0];
stress = [-100.0, -250.0, -250.0, 50.0, 10.0, 0.0];
strain = [-0.00005, 0.000015, 0.000015, 0.0, 0.0, 0.0];
dt = 0.001;
totaltime =  0.0;

new_system('xmat_test');

% Input blocks
add_block('built-in/StringConstant', 'xmat_test/materialname', 'String', materialname, ...
   'Position', [205, 120, 320, 150]);
add_block('built-in/StringToASCII', 'xmat_test/String to ASCII', 'Position', [445, 120, 540, 150]);
add_block('built-in/Constant', 'xmat_test/materialparameters', 'Value', mat2str(materialparameters), ...
   'Position', [225, 180, 320, 210]);
add_block('built-in/Constant', 'xmat_test/statevariables', 'Value', mat2str(statevariables), ...
   'Position', [225, 240, 320, 270]);
add_block('built-in/Constant', 'xmat_test/stress', 'Value', mat2str(stress), ...
   'Position', [225, 300, 320, 330]);
add_block('built-in/Constant', 'xmat_test/strain', 'Value', mat2str(strain), ...
   'Position', [225, 360, 320, 390]);
add_block('built-in/Constant', 'xmat_test/dt', 'Value', num2str(dt), 'Position', [290, 420, 320, 450]);
add_block('built-in/Constant', 'xmat_test/totaltime', 'Value', num2str(totaltime), ...
   'Position', [290, 480, 320, 510]);

% xMat S-Function block
add_block('built-in/S-Function', 'xmat_test/S-Function of xMat', 'FunctionName', 'xmat_sl', ...
   'Position', [445, 220, 575, 410], 'BackgroundColor', '[0.569, 0.843, 0.263]');

% Simple one-step explicit Euler integration
add_block('hdlsllib/HDL Operations/Multiply-Add', 'xmat_test/Mult-Add-stress', ...
   'Position', [515, 0, 575, 50]);
add_block('hdlsllib/HDL Operations/Multiply-Add', 'xmat_test/Mult-Add-statev', ...
   'Position', [515, 580, 575, 630]);

% Output to workspace blocks
add_block('built-in/ToWorkspace', 'xmat_test/out_dotstress', 'VariableName', 'dotstress', ...
   'Position', [700, 140, 790, 170]);
add_block('built-in/ToWorkspace', 'xmat_test/out_newstress', 'VariableName', 'newstress', ...
   'Position', [700, 220, 790, 250]);
add_block('built-in/ToWorkspace', 'xmat_test/out_dotstate', 'VariableName', 'dotstate', ...
   'Position', [700, 300, 790, 330]);
add_block('built-in/ToWorkspace', 'xmat_test/out_newstate', 'VariableName', 'newstate', ...
   'Position', [700, 380, 790, 410]);
```

```
add_block('built-in/ToWorkspace', 'xmat_test/out_jacobian', 'VariableName', 'jacobian', ...
   'Position', [700, 460, 790, 490]);

% Display blocks
add_block('built-in/Display', 'xmat_test/new_stress', 'Position', [910, -100, 1000, 30]);
add_block('built-in/Display', 'xmat_test/dot_stress', 'Position', [910, 60, 1000, 190]);
add_block('built-in/Display', 'xmat_test/dot_state', 'Position', [910, 220, 1000, 350]);
add_block('built-in/Display', 'xmat_test/jacobian', 'Position', [910, 380, 1000, 510]);
add_block('built-in/Display', 'xmat_test/new_state', 'Position', [910, 540, 1000, 670]);

% Connections into xMat block
add_line('xmat_test', ['materialname', '/1'], ['String to ASCII', '/1']);
lh = add_line('xmat_test', ['String to ASCII', '/1'], ['S-Function of xMat', '/1']);
set(lh, 'Points', [540, 135; 560, 135; 560, 185; 430, 185; 430, 240; 445, 240]);
lh = add_line('xmat_test', ['materialparameters', '/1'], ['S-Function of xMat', '/2']);
set(lh, 'Points', [325, 195; 385, 195; 385, 265; 445, 265]);
lh = add_line('xmat_test', ['statevariables', '/1'], ['S-Function of xMat', '/3']);
set(lh, 'Points', [325, 255; 370, 255; 370, 290; 445, 290]);
add_line('xmat_test', ['stress', '/1'], ['S-Function of xMat', '/4']);
lh = add_line('xmat_test', ['strain', '/1'], ['S-Function of xMat', '/5']);
set(lh, 'Points', [325, 375; 385, 375; 385, 340; 445, 340]);
lh = add_line('xmat_test', ['dt', '/1'], ['S-Function of xMat', '/6']);
set(lh, 'Points', [325, 435; 400, 435; 400, 365; 445, 365]);
lh = add_line('xmat_test', ['totaltime', '/1'], ['S-Function of xMat', '/7']);
set(lh, 'Points', [325, 495; 430, 495; 430, 390; 445, 390]);

% Connections out of xMat block
lh = add_line('xmat_test', ['S-Function of xMat', '/1'], ['out_dotstress', '/1']);
set(lh, 'Points', [580, 250; 620, 250; 620, 155; 695, 155]);
lh = add_line('xmat_test', ['S-Function of xMat', '/1'], ['dot_stress', '/1']);
set(lh, 'Points', [580, 250; 620, 250; 620, 155; 680, 155; 680, 125; 905, 125]);
add_line('xmat_test', ['S-Function of xMat', '/2'], ['out_dotstate', '/1']);
lh = add_line('xmat_test', ['S-Function of xMat', '/2'], ['dot_state', '/1']);
set(lh, 'Points', [580, 315; 680, 315; 680, 285; 905, 285]);
lh = add_line('xmat_test', ['S-Function of xMat', '/3'], ['out_jacobian', '/1']);
set(lh, 'Points', [580, 380; 620, 380; 620, 475; 695, 475]);
lh = add_line('xmat_test', ['S-Function of xMat', '/3'], ['jacobian', '/1']);
set(lh, 'Points', [580, 380; 620, 380; 620, 475; 680, 475; 680, 445; 905, 445]);

% Connections to Multiply-Adds
lh = add_line('xmat_test', ['dt', '/1'], ['Mult-Add-stress', '/1']);
set(lh, 'Points', [325, 435; 400, 435; 400, 365; 400, 10; 510, 10]);
lh = add_line('xmat_test', ['S-Function of xMat', '/1'], ['Mult-Add-stress', '/2']);
set(lh, 'Points', [580, 250; 620, 250; 620, 155; 620, 75; 490, 75; 490, 25; 510, 25]);
lh = add_line('xmat_test', ['stress', '/1'], ['Mult-Add-stress', '/3']);
set(lh, 'Points', [325, 315; 415, 315; 415, 40; 510, 40]);
lh = add_line('xmat_test', ['Mult-Add-stress', '/1'], ['out_newstress', '/1']);
set(lh, 'Points', [575, 25; 640, 25; 640, 235; 695, 235]);
lh = add_line('xmat_test', ['Mult-Add-stress', '/1'], ['new_stress', '/1']);
set(lh, 'Points', [575, 25; 640, 25; 640, -35; 905, -35]);

lh = add_line('xmat_test', ['dt', '/1'], ['Mult-Add-statev', '/1']);
set(lh, 'Points', [325, 435; 400, 435; 400, 590; 510, 590]);
lh = add_line('xmat_test', ['S-Function of xMat', '/2'], ['Mult-Add-statev', '/2']);
set(lh, 'Points', [580, 315; 600, 315; 600, 555; 490, 555; 490, 605; 510, 605]);
lh = add_line('xmat_test', ['statevariables', '/1'], ['Mult-Add-statev', '/3']);
set(lh, 'Points', [325, 255; 370, 255; 370, 290; 370, 620; 510, 620]);
lh = add_line('xmat_test', ['Mult-Add-statev', '/1'], ['out_newstate', '/1']);
set(lh, 'Points', [575, 605; 640, 605; 640, 395; 695, 395]);
add_line('xmat_test', ['Mult-Add-statev', '/1'], ['new_state', '/1']);

```

```
% Annotations
note = Simulink.Annotation('xmat_test', 'Input_values');
set(note, 'FixedHeight', 'on', 'FixedWidth', 'on', 'Position', [175, 85, 355, 535], ...
   'ForegroundColor', '[0.45, 0.45, 0.45]', 'BackgroundColor', '[0.95, 0.95, 0.95]', ...
   'FontWeight', 'bold');
note = Simulink.Annotation('xmat_test', 'Output_values');
set(note, 'FixedHeight', 'on', 'FixedWidth', 'on', 'Position', [665, 85, 835, 535], ...
   'ForegroundColor', '[0.45, 0.45, 0.45]', 'BackgroundColor', '[0.95, 0.95, 0.95]', ...
   'FontWeight', 'bold');

save_system('xmat_test', 'xmat_test.slx')
```

Code F.13: MATLAB-Skript zum Starten des Simulink-Modells.

```
% Run this script in MATLAB after compiling xmat.f and xmat_sl.c as a C MEX S-function
% and creating the model xmat_test.slx

% Start test system in Simulink and run it
open_system('xmat_test.slx');
sim_res = sim('xmat_test', 'SimulationMode', 'normal', 'CaptureErrors', 'on')
disp(sim_res.ErrorMessage);

disp('-- Done --');

dotstate = sim_res.dotstate(end, :);
dotstress = sim_res.dotstress(end, :);
jacobian = sim_res.jacobian(end, :);
newstate = sim_res.newstate(end, :);
newstress = sim_res.newstress(end, :);

fid = fopen('ausgabe', 'w');
fprintf(fid, 'dotstress = ');
fprintf(fid, '%16.7f', dotstress);
fprintf(fid, '\n');

fprintf(fid, 'dotstate = ');
fprintf(fid, '%16.7f', dotstate);
fprintf(fid, '\n');

fprintf(fid, 'jacobian = \n');
for idx = 1:6
   fprintf(fid, '( %i, :) ', idx);
   fprintf(fid, '%16.7f', jacobian(((idx-1)*6+1):6*idx));
   fprintf(fid, '\n');
end
fprintf(fid, '\n');

fprintf(fid, 'newstress = ');
fprintf(fid, '%16.7f', newstress);
fprintf(fid, '\n');

fprintf(fid, 'newstate = ');
fprintf(fid, '%16.7f', newstate);
fprintf(fid, '\n');
fclose(fid);
```

Python-Skript

In Python können Pakete wie `ctypes` genutzt werden, um auf C-Bibliotheken zugreifen zu können. Code F.14 zeigt den Code einer Python-Datei `xmat_py.py`, in der Funktionalitäten in einer Python-Klasse `xMat` verfügbar gemacht werden. Diese Klasse kann anschließend in einem Skript geladen und normal aus Python heraus genutzt werden (siehe Code F.15). Beispielhafte Anweisungen zum Kompilieren der C-Bibliothek finden sich in Code F.16 für Linux und Code F.17 für Windows.

Code F.14: Python-Datei `xmat_py.py` mit einer Hilfsklasse `xMat`, die von anderen Python-Skripten genutzt werden kann.

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-


class xMat(object):
    """Wrapper class for Python to access the C interface of the xMat library.
    """
    # ----------------------------------------------------------------------------------------------
    def __init__(self, lib_path):
        import os

        self.lib = None

        loaded = False
        if (os.name == 'posix'):
            from ctypes import cdll

            try:
                self.lib = cdll.LoadLibrary(lib_path)
                loaded = True
            except:
                print('Could not load <xmat>.so library (check path/name)')
        elif (os.name == 'nt'):
            from ctypes import WinDLL

            try:
                self.lib = WinDLL(lib_path)
                loaded = True
            except:
                print('Could not load <xmat>.dll library (check path/name)')
        else:
            print('OS not supported')

        if (loaded):
            from ctypes import c_char_p, c_int, c_double, POINTER
            self.lib.Dot_Values_CInter.restype = None
            self.lib.Dot_Values_CInter.argtypes = [  # --- Expected C Interface
                c_char_p,                            # const char *materialname,
                POINTER(c_int),                      # const int *nparams,
                POINTER(c_double),                   # double materialparameters[],
                POINTER(c_int),                      # const int *nstatevar,
                POINTER(c_double),                   # double statevariables[],
                POINTER(c_int),                      # const int *ncomponents,
                POINTER(c_double),                   # double oldstress[],
                POINTER(c_double),                   # double oldstrain[],
                POINTER(c_double),                   # double timeincrement[],
                POINTER(c_double),                   # double totaltime[],
```

```
            POINTER(c_double),                  # double dotstress[],
            POINTER(c_double),                  # double dotstate[],
            POINTER(c_double)                   # double jacobian[]
        ]
        self.lib.xmat_console_CInter.restype = None
        self.lib.xmat_console_CInter.argtypes = [# --- Expected C Interface
            c_char_p,                           # const char *materialname,
            POINTER(c_int),                     # const int *nparams,
            POINTER(c_double),                  # double materialparameters[],
            POINTER(c_int),                     # const int *nstatevar,
            POINTER(c_double),                  # double statevariables[],
            POINTER(c_int),                     # const int *ncomponents,
            POINTER(c_double),                  # double oldstress[],
            POINTER(c_double),                  # double oldstrain[],
            POINTER(c_double),                  # double timeincrement[],
            POINTER(c_double),                  # double totaltime[],
            POINTER(c_double),                  # double newstress[],
            POINTER(c_double),                  # double newstate[],
            POINTER(c_double)                   # double jacobian[]
        ]

    # ------------------------------------------------------------------------------------------
    def _PrepareInput(self, materialname, materialparameters, statevariables, oldstress,
        oldstrain, timeincrement, totaltime):
        import numpy
        from ctypes import c_double, c_char_p, c_int

        c_materialname = c_char_p(materialname.encode())

        nparams = len(materialparameters)
        c_nparams = c_int(nparams)
        cont_materialparameters = numpy.ascontiguousarray(materialparameters, dtype=numpy.double)
        c_materialparameters = (c_double * nparams)(*cont_materialparameters)

        nstatevar = len(statevariables)
        c_nstatevar = c_int(nstatevar)
        cont_statevariables = numpy.ascontiguousarray(statevariables, dtype=numpy.double)
        c_statevariables = (c_double * nstatevar)(*cont_statevariables)

        ncomponents = len(oldstress)
        c_ncomponents = c_int(ncomponents)
        cont_oldstress = numpy.ascontiguousarray(oldstress, dtype=numpy.double)
        c_oldstress = (c_double * ncomponents)(*cont_oldstress)
        cont_oldstrain = numpy.ascontiguousarray(oldstrain, dtype=numpy.double)
        c_oldstrain = (c_double * ncomponents)(*cont_oldstrain)

        c_timeincrement = c_double(timeincrement)
        c_totaltime = c_double(totaltime)

        # Preparation of return values
        outstress = [0.0 for x in range(ncomponents)]
        outstress = numpy.ascontiguousarray(outstress, dtype=numpy.double)
        double_ncomponents = c_double * ncomponents
        c_outstress = double_ncomponents(*outstress)

        outstate = [0.0 for x in range(nstatevar)]
        outstate = numpy.ascontiguousarray(outstate, dtype=numpy.double)
        double_nstatevar = c_double * nstatevar
        c_outstate = double_nstatevar(*outstate)

        jacobian = [0.0 for x in range(ncomponents*ncomponents)]
```

```
109         jacobian = numpy.ascontiguousarray(jacobian, dtype=numpy.double)
110         double_n2components = c_double * (ncomponents*ncomponents)
111         c_jacobian = double_n2components(*jacobian)
112 
113         return [c_materialname, c_nparams, c_materialparameters,
114            c_nstatevar, c_statevariables, c_ncomponents, c_oldstress, c_oldstrain,
115            c_timeincrement, c_totaltime, c_outstress, c_outstate, c_jacobian]
116 
117     # ----------------------------------------------------------------------------------------
118     def Dot_Values_CInter(self, materialname, materialparameters, statevariables, oldstress,
119        oldstrain, timeincrement, totaltime):
120         from ctypes import byref, c_double, POINTER, cast
121 
122         if (self.lib is None):
123            print('No library loaded')
124            return [None, None, None]
125 
126         c_materialname, c_nparams, c_materialparameters, c_nstatevar, c_statevariables, \
127         c_ncomponents, c_oldstress, c_oldstrain, c_timeincrement, c_totaltime, c_dotstress, \
128         c_dotstate, c_jacobian = self._PrepareInput(materialname=materialname,
129            materialparameters=materialparameters, statevariables=statevariables, oldstress=oldstress,
130            oldstrain=oldstrain, timeincrement=timeincrement, totaltime=totaltime)
131 
132         # Call the routine
133         self.lib.Dot_Values_CInter(c_materialname, byref(c_nparams), c_materialparameters,
134            byref(c_nstatevar), c_statevariables, byref(c_ncomponents), c_oldstress, c_oldstrain,
135            byref(c_timeincrement), byref(c_totaltime), c_dotstress, c_dotstate, c_jacobian)
136 
137         nparams = len(materialparameters)
138         nstatevar = len(statevariables)
139         ncomponents = len(oldstress)
140 
141         # Assign return values to python lists
142         temp_dotstress = cast(c_dotstress, POINTER(c_double*ncomponents))
143         dotstress = [temp_dotstress.contents[idx] for idx in range(ncomponents)]
144 
145         temp_dotstate = cast(c_dotstate, POINTER(c_double*nstatevar))
146         dotstate = [temp_dotstate.contents[idx] for idx in range(nstatevar)]
147 
148         temp_jacobian = cast(c_jacobian, POINTER(c_double*(ncomponents*ncomponents)))
149         jacobian = [temp_jacobian.contents[idx] for idx in range(ncomponents*ncomponents)]
150 
151         return [dotstress, dotstate, jacobian]
152 
153     # ----------------------------------------------------------------------------------------
154     def xmat_console_CInter(self, materialname, materialparameters, statevariables, oldstress,
155        oldstrain, timeincrement, totaltime):
156         from ctypes import byref, c_double, POINTER, cast
157 
158         if (self.lib is None):
159            print('No library loaded')
160            return [None, None, None]
161 
162         c_materialname, c_nparams, c_materialparameters, c_nstatevar, c_statevariables, \
163         c_ncomponents, c_oldstress, c_oldstrain, c_timeincrement, c_totaltime, c_newstress, \
164         c_newstate, c_jacobian = self._PrepareInput(materialname=materialname,
165            materialparameters=materialparameters, statevariables=statevariables, oldstress=oldstress,
166            oldstrain=oldstrain, timeincrement=timeincrement, totaltime=totaltime)
167 
168         # Call the routine
169         self.lib.xmat_console_CInter(c_materialname, byref(c_nparams), c_materialparameters,
```

```
        byref(c_nstatevar), c_statevariables, byref(c_ncomponents), c_oldstress, c_oldstrain,
        byref(c_timeincrement), byref(c_totaltime), c_newstress, c_newstate, c_jacobian)

    nparams = len(materialparameters)
    nstatevar = len(statevariables)
    ncomponents = len(oldstress)

    # Assign return values to python lists
    temp_newstress = cast(c_newstress, POINTER(c_double*ncomponents))
    newstress = [temp_newstress.contents[idx] for idx in range(ncomponents)]

    temp_newstate = cast(c_newstate, POINTER(c_double*nstatevar))
    newstate = [temp_newstate.contents[idx] for idx in range(nstatevar)]

    temp_jacobian = cast(c_jacobian, POINTER(c_double*(ncomponents*ncomponents)))
    jacobian = [temp_jacobian.contents[idx] for idx in range(ncomponents*ncomponents)]

    return [newstress, newstate, jacobian]
```

Code F.15: Python-Skript zum Testen der xMat-Funktionen.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

from xmat_py import xMat


materialname = 'HYPO-VW96_Test'
materialparameters = [0.5777, 0.0, 4000000.0, 0.27, 0.677, 1.054, 1.212, 0.14, 2.5,
                      1.1, 2.2, 0.0001, 0.1, 5.5, 0.0, 0.8]
statevariables = [0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

oldstress = [-100.0, -250.0, -250.0, 50.0, 10.0, 0.0]

oldstrain = [-0.00005, 0.000015, 0.000015, 0.0, 0.0, 0.0]
timeincrement = 0.001
totaltime = 0.0

import os

if (os.name == 'posix'):
   xmat_obj = xMat(lib_path='./xmatc.so')
elif (os.name == 'nt'):
   xmat_obj = xMat(lib_path='./xmatc.dll')
else:
   print('OS not supported')
   xmat_obj = None


dotstress, dotstate, jacobian = xmat_obj.Dot_Values_CInter(materialname=materialname,
   materialparameters=materialparameters, statevariables=statevariables, oldstress=oldstress,
   oldstrain=oldstrain, timeincrement=timeincrement, totaltime=totaltime)

print('dotstress = ' + ''.join(['{:16.7f}'.format(x) for x in dotstress]))
print('dotstate = ' + ''.join(['{:16.7f}'.format(x) for x in dotstate]))

print('jacobian = ')
for idx in range(6):
   print('( ' + str(idx+1) + ', :) ' + ''.join(['{:16.7f}'.format(x) for x in jacobian[6*idx:6*(idx+1)]]))

```

```
print('')

newstress, newstate, jacobian = xmat_obj.xmat_console_CInter(materialname=materialname,
   materialparameters=materialparameters, statevariables=statevariables, oldstress=oldstress,
   oldstrain=oldstrain, timeincrement=timeincrement, totaltime=totaltime)

print('newstress = ' + ''.join(['{:16.7f}'.format(x) for x in newstress]))
print('newstate = ' + ''.join(['{:16.7f}'.format(x) for x in newstate]))

print('jacobian = ')
for idx in range(6):
   print('( ' + str(idx+1) + ', :) ' + ''.join(['{:16.7f}'.format(x) for x in jacobian[6*idx:6*(idx+1)]]))

print('')
```

Code F.16: Befehl zur Erzeugung einer C-Bibliothek unter Linux mit `gfortran` bzw. `ifort`.

```
# Variante mit gfortran
gfortran -ffree-form -cpp -shared -fPIC -DCINTER xmat.f -o xmatc.so;

# Variante mit ifort
ifort -free -fpp -shared -fPIC -DCINTER xmat.f -o xmatc.so;
```

Code F.17: Batch-Datei zur Erzeugung einer C-Bibliothek unter Windows mit `ifort`. Der Pfad zu `vars.bat` und die Optionen müssen im Allgemeinen angepasst werden.

```
@echo off
@call "C:\Program Files (x86)\Intel\oneAPI\compiler\2021.4.0\env\vars.bat" intel64 vs2019
ifort /free /fpp /dll /recursive /MT /DCINTER xmat.f /exe:xmatc.dll
del *.mod
del xmatc.exp xmatc.lib xmat.obj
pause
```

Plaxis SoilTest

Um Elementversuche mit der xMat im SoilTest-Modul von Plaxis nachzurechnen, muss die xMat als Bibliothek bereitgestellt werden (siehe Abschnitt 4.3.5). Ödometer- und Triaxialversuche können entweder direkt oder über einen allgemeinen Belastungszustand definiert werden. In Code F.18 und Code F.19 wird dargestellt, wie der allgemeine Belastungszustand definiert werden kann, um vergleichbare Ergebnisse zu den Berechnungen mit dem ElementSimulator in Abschnitt 7.3 zu erhalten.

Code F.18: Auszug der Setup-Datei `general_oedo.vls` für einen Ödometerversuch im SoilTest-Modul von Plaxis.

```
[header]
8       =file version number

kN      =unit of force
m       =unit of length
s       =unit of time


```

```
[testparameters]
5       =general
0       =behavior
0.01    =error1
1       =relax
50      =maxiter
0       =arclength off
0       =isotropic preconsolidation stress
-1      =mob. rel. shear strength
0       =inner radius
0       =test control type
-5      -10     -5      0       0       0       =initial stresses
3       =number of phases
500     =number of steps in phase 1
1       =phase duration for phase 1
0       0       0       0       0       0       =strain increments for phase 1
0       -90     0       0       0       0       =stress increments for phase 1
0       1       0       0       0       0       =boundary conditions
500     =number of steps in phase 2
1       =phase duration for phase 2
0       0       0       0       0       0       =strain increments for phase 2
0       50      0       0       0       0       =stress increments for phase 2
0       1       0       0       0       0       =boundary conditions
500     =number of steps in phase 3
1       =phase duration for phase 3
0       0       0       0       0       0       =strain increments for phase 3
0       -150    0       0       0       0       =stress increments for phase 3
0       1       0       0       0       0       =boundary conditions
```

Code F.19: Auszug der Setup-Datei `general_triax.vls` für einen Triaxialversuch im SoilTest-Modul von Plaxis.

```
[header]
8       =file version number

kN      =unit of force
m       =unit of length
s       =unit of time


[testparameters]
5       =general
0       =behavior
0.01    =error1
1       =relax
50      =maxiter
0       =arclength off
0       =isotropic preconsolidation stress
-1      =mob. rel. shear strength
0       =inner radius
0       =test control type
-100    -100    -100    0       0       0       =initial stresses
1       =number of phases
2000    =number of steps in phase 1
0.1     =phase duration for phase 1
0       -0.2    0       0       0       0       =strain increments for phase 1
0       0       0       0       0       0       =stress increments for phase 1
1       0       1       1       0       0       =boundary conditions
```

Erddruckversuche mit Abaqus/Explizit

Die numerischen Modelle der Simulationen in Kapitel 8 sind mit dem Python-Skript in Code F.20 erstellt worden.

Code F.20: Python-Skript zur Erstellung des numerischen Modells der Erddruckversuche mit der gekoppelten Euler-Lagrange-Methode in Abaqus/Explizit.

```
# -*- coding: utf-8 -*-
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *

startZeit = time.time()

abapys_dir = '.'             # Hier muss der Pfad zum lokalen abapys-Ordner angegeben werden

sys.path.insert(0, abapys_dir)

from abapys import abapys_tol, InitialisiereAbapys, BedingteAuswahl, ElementAuswahl, Log
from abapys import Rechteck, Linienzug
from abapys import Bodenparameter, BodenmaterialUndSectionErstellen

InitialisiereAbapys(session=session, version=version, pfad=abapys_dir)

materialwahl = 'sand'     # sand, glas
userroutine = 'xmat.f'


# ---------------------------------------- Bodenparameter ---------------------------------------- #
#
basisdicke = 0.030 # [m]
bodenbereich = [0.573 + 2.0*basisdicke, 0.340 + basisdicke]

# mat_bezeichnung ist der Name des verwendeten Bodenmaterials aus der Materialdatenbank von abapys
# Es werden modifizierte Parameter für beide Materialien verwendet!
if (materialwahl == 'sand'):
    mat_bezeichnung = 'mod. Hamburger Sand 2023'
    bodenhoehe = 0.084 # [m]
    verd_grad = 0.395
elif (materialwahl == 'glas'):
    mat_bezeichnung = 'mod. SiLibeads 3-5mm'
    bodenhoehe = 0.154 # [m]
    verd_grad = 0.91
else:
   log('ERROR: Material unknown')
#
xpos_schieber = bodenbereich[0] - basisdicke


```

```
# ------------------------------------ Materialparameter ------------------------------------ #
#
g = 9.81 # [m/s^2]

# Stahlbauteile
stahl_dichte = 7.87; # [kN/m^3]
#                   E-Modul   Querdehnz.
#                   E [kPa]   nu [-]
#                 |---------|-----------|
stahl_elastisch = [ 210e6,    0.3         ]

bodenmaterial = 'Hypo-VW96'

parameter_boden = [
#     Abaqus-Bez.      Datenbankname          Saettigung   Verd-grad     Stoffgesetz
#     >''              >''                    [0-1]        [0-1]         >''
#   |---------------|---------------------|------------|------------|----------------|
    [ bodenmaterial,   mat_bezeichnung,       0.0,         verd_grad,    'Hypoplastisch' ],
#   |---------------|---------------------|------------|------------|----------------|
]
# Verdichtungsgrad V_D ergibt sich aus
#   a) Anfangsporenzahl e_0:    V_D = (e_max - e_0)/(e_max - e_min) * (e_min + 1)/(e_0 + 1)
#   b) Anfangsdichte rho_0:     V_D = (rho_0 - rho_min)/(rho_max - rho_min)

geostatische_nullspannung = 0.1; # [kN/m^2]


# ------------------------------------ Werkzeugparameter ------------------------------------ #
#
schieber_dimensions = [0.030, 0.3 + basisdicke] # [m]
modelthickness = 0.01 # [m]
meshsize = modelthickness # [m]

refweg = 0.198 # [m]
reftime = 8.0 # [s]


# ------------------------------------ Simulationsparameter ------------------------------------ #
#
zeit_anfangszustand = 0.01*reftime # [s]

timescaling = 1.0
numoutputs_initial_step = 5
numoutputs_per_step = 100

# Kontakt
kontakt_mit_reibung = True
reibungskoeffizient = 0.22

numDepVar = 20



# ========================================================================================== #
#                        A u t o m a t i s c h e    B e a r b e i t u n g                    #
# ========================================================================================== #
#
modelname = 'Schiebermodell'
if session.mdbData.has_key(modelname):
    log('WARNING: Model name already exists - this run has no effect')

```

```
mdb.Model(name=modelname, modelType=STANDARD_EXPLICIT)
mymodel = mdb.models[modelname]


# ------------------------------------ Materialdefinitionen ------------------------------------ #
#
mymodel.Material(name='Stahl')
mymodel.materials['Stahl'].Density(table=((stahl_dichte, ), ))
mymodel.materials['Stahl'].Elastic(table=(stahl_elastisch, ))
mymodel.HomogeneousSolidSection(material='Stahl', name='secStahl', thickness=None)

benoetigtUserroutine, verwendeteBodenwerte = BodenmaterialUndSectionErstellen(modell=mymodel,
    verwendeteMaterialien=[bodenmaterial], verfuegbareMaterialien=parameter_boden,
    userroutine=userroutine, numDepVar=numDepVar)

tempBodenparameter = Bodenparameter(name=mat_bezeichnung, stoffgesetz='Hypoplastisch', bezeichnung='')
korndichte, mindichte, maxdichte, kritReibwinkel = tempBodenparameter[0:4]
trockendichte = mindichte + verd_grad*(maxdichte-mindichte)
aktuelle_porenzahl = korndichte/trockendichte - 1.0
max_porenzahl = korndichte/mindichte - 1.0

# ------------------------------------ Kontaktbedingungen ------------------------------------ #
#
mymodel.ContactProperty('Kontakt')
mymodel.interactionProperties['Kontakt'].NormalBehavior(allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT, pressureOverclosure=HARD)
if (kontakt_mit_reibung):
    mymodel.interactionProperties['Kontakt'].TangentialBehavior(
        dependencies=0, directionality=ISOTROPIC, elasticSlipStiffness=None,
        formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
        pressureDependency=OFF, shearStressLimit=None, slipRateDependency=OFF,
        table=((reibungskoeffizient, ), ), temperatureDependency=OFF)
else:
    mymodel.interactionProperties['Kontakt'].TangentialBehavior(formulation=FRICTIONLESS)


# ------------------------------------ Part: Boden ------------------------------------ #
#
zeichnung = mymodel.ConstrainedSketch(name='__profile__', sheetSize=2.0*max(bodenbereich))
Rechteck(zeichnung=zeichnung, punkt1=(0.0, 0.0), punkt2=(bodenbereich[0], bodenbereich[1]))
partBoden = mymodel.Part(dimensionality=THREE_D, name='Boden', type=EULERIAN)
partBoden.BaseSolidExtrude(depth=modelthickness, sketch=zeichnung)
del zeichnung

ptoben = partBoden.DatumPointByCoordinate(coords=(xpos_schieber, bodenhoehe+basisdicke, 0.0))
partBoden.features.changeKey(fromName='Datum pt-1', toName='datum_Oberkante')
ptlrand = partBoden.DatumPointByCoordinate(coords=(basisdicke, basisdicke, 0.0))
partBoden.features.changeKey(fromName='Datum pt-1', toName='datum_Lrand')
xachse = partBoden.DatumAxisByPrincipalAxis(principalAxis=XAXIS)
partBoden.features.changeKey(fromName='Datum axis-1', toName='xAchse')
yachse = partBoden.DatumAxisByPrincipalAxis(principalAxis=YAXIS)
partBoden.features.changeKey(fromName='Datum axis-1', toName='yAchse')

partBoden.PartitionCellByPlanePointNormal(cells=partBoden.cells,
    normal=partBoden.datums[yachse.id], point=partBoden.datums[ptoben.id])
partBoden.PartitionCellByPlanePointNormal(cells=partBoden.cells,
    normal=partBoden.datums[xachse.id], point=partBoden.datums[ptoben.id])
partBoden.PartitionCellByPlanePointNormal(cells=partBoden.cells,
    normal=partBoden.datums[yachse.id], point=partBoden.datums[ptlrand.id])
partBoden.PartitionCellByPlanePointNormal(cells=partBoden.cells,
    normal=partBoden.datums[xachse.id], point=partBoden.datums[ptlrand.id])
```

```

flaechen_Unterseite = BedingteAuswahl(elemente=partBoden.faces, bedingung='elem.pointOn[0][1] <= 0.0')
partBoden.Set(name='setUnterseite', faces=flaechen_Unterseite)
flaechen_X = BedingteAuswahl(elemente=partBoden.faces,
    bedingung='(abs(elem.pointOn[0][0]) < var[1]) or (abs(elem.pointOn[0][0]) > var[0]-var[1])',
    var=[bodenbereich[0], abapys_tol])
partBoden.Set(name='setWandX', faces=flaechen_X)
flaechen_Y = BedingteAuswahl(elemente=partBoden.faces,
    bedingung='abs(elem.pointOn[0][2] - var[0]) > var[0]-var[1]', var=[modelthickness/2.0, abapys_tol])
partBoden.Set(name='setWandY', faces=flaechen_Y)
Vol_Sand = BedingteAuswahl(elemente=partBoden.cells,
    bedingung='(elem.pointOn[0][0] < var[0]+var[3]) and (elem.pointOn[0][0] > var[1]-var[3]) '\
      'and (elem.pointOn[0][1] < var[2]+var[3]) and (elem.pointOn[0][1] > var[1]-var[3])',
    var=[xpos_schieber, basisdicke, bodenhoehe+basisdicke, abapys_tol])
partBoden.Set(name='setSand', cells=Vol_Sand)
partBoden.Set(cells=partBoden.cells, name='setAll')

partBoden.setElementType(regions=(partBoden.cells, ), elemTypes=(
    ElemType(elemCode=EC3D8R, elemLibrary=EXPLICIT, secondOrderAccuracy=OFF, hourglassControl=DEFAULT),
    ElemType(elemCode=UNKNOWN_WEDGE, elemLibrary=EXPLICIT),
    ElemType(elemCode=UNKNOWN_TET, elemLibrary=EXPLICIT)))

partBoden.seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=meshsize)
partBoden.generateMesh()

lockere_Zelle = ElementAuswahl(elemente=partBoden.elements, punktliste=partBoden.nodes,
    bedingung='(punkt.coordinates[0] < var[0]+var[3]) and (punkt.coordinates[0] > var[0]-var[2]-var[3]) '\
       'and (punkt.coordinates[1] < var[1]+var[2]+var[3]) and (punkt.coordinates[1] > var[1]-var[3])',
    var=[xpos_schieber, basisdicke, 1.1*meshsize, abapys_tol])
partBoden.Set(name='setLockereZelle', elements=lockere_Zelle)

partBoden.SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE,
    region=partBoden.sets['setAll'], sectionName='secEuler', thicknessAssignment=FROM_SECTION)


# ---------------------------------------- Part: Schieber ---------------------------------------- #
#
zeichnung = mymodel.ConstrainedSketch(name='__profile__', sheetSize=2.0*max(schieber_dimensions))
Rechteck(zeichnung=zeichnung, punkt1=(-schieber_dimensions[0]/2, 0.0),
    punkt2=(schieber_dimensions[0]/2, schieber_dimensions[1]))
partSchieber = mymodel.Part(dimensionality=THREE_D, name='Schieber', type=DEFORMABLE_BODY)
partSchieber.BaseSolidExtrude(depth=modelthickness, sketch=zeichnung)
del zeichnung

partSchieber.Set(cells=partSchieber.cells, name='setAll')

partSchieber.ReferencePoint(point=(0.0, schieber_dimensions[1], modelthickness/2.0))
partSchieber.features.changeKey(fromName='RP', toName='RP_Schieber')
partSchieber.Set(name='setRP',
    referencePoints=(partSchieber.referencePoints[partSchieber.features['RP_Schieber'].id], ))

partSchieber.setElementType(elemTypes=(ElemType(
    elemCode=C3D8R, elemLibrary=EXPLICIT, secondOrderAccuracy=OFF,
    distortionControl=DEFAULT), ElemType(elemCode=C3D6, elemLibrary=EXPLICIT),
    ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)), regions=(partSchieber.cells, ))

partSchieber.seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=meshsize)
partSchieber.generateMesh()

partSchieber.SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE,
    region=partSchieber.sets['setAll'], sectionName='secStahl', thicknessAssignment=FROM_SECTION)
```

```
236
237
238 # ---------------------------------------- Part: Lrand ---------------------------------------- #
239 #
240 zeichnung = mymodel.ConstrainedSketch(name='__profile__', sheetSize=2.0*max(bodenbereich))
241 Linienzug(zeichnung=zeichnung, punkte=[
242     (0.0, 0.0),
243     (0.0, bodenbereich[1]),
244     (basisdicke, bodenbereich[1]),
245     (basisdicke, basisdicke),
246     (bodenbereich[0], basisdicke),
247     (bodenbereich[0], 0.0)], geschlossen=True)
248 partLrand = mymodel.Part(dimensionality=THREE_D, name='Lrand', type=DEFORMABLE_BODY)
249 partLrand.BaseSolidExtrude(depth=modelthickness, sketch=zeichnung)
250 del zeichnung
251
252 xachse = partLrand.DatumAxisByPrincipalAxis(principalAxis=XAXIS)
253 partLrand.features.changeKey(fromName='Datum axis-1', toName='xAchse')
254
255 partLrand.PartitionCellByPlanePointNormal(cells=partLrand.cells,
256     normal=partLrand.datums[xachse.id], point=(basisdicke, basisdicke, 0.0))
257
258 partLrand.Set(cells=partLrand.cells, name='setAll')
259
260 partLrand.ReferencePoint(point=(0.0, bodenbereich[1], modelthickness/2.0))
261 partLrand.features.changeKey(fromName='RP', toName='RP_Lrand')
262 partLrand.Set(name='setRP',
263     referencePoints=(partLrand.referencePoints[partLrand.features['RP_Lrand'].id], ))
264
265 partLrand.setElementType(elemTypes=(ElemType(
266     elemCode=C3D8R, elemLibrary=EXPLICIT, secondOrderAccuracy=OFF,
267     distortionControl=DEFAULT), ElemType(elemCode=C3D6, elemLibrary=EXPLICIT),
268     ElemType(elemCode=C3D4, elemLibrary=EXPLICIT)), regions=(partLrand.cells, ))
269
270 partLrand.seedPart(deviationFactor=0.1, minSizeFactor=0.1, size=meshsize)
271 partLrand.generateMesh()
272
273 partLrand.SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE,
274     region=partLrand.sets['setAll'], sectionName='secStahl', thicknessAssignment=FROM_SECTION)
275
276
277 # ------------------------ Assembly: Positionierung und starre Elemente ------------------------ #
278 #
279 mymodel.rootAssembly.DatumCsysByDefault(CARTESIAN)
280 instBoden = mymodel.rootAssembly.Instance(dependent=ON, name='instBoden', part=partBoden)
281 instSchieber = mymodel.rootAssembly.Instance(dependent=ON, name='instSchieber', part=partSchieber)
282 instLrand = mymodel.rootAssembly.Instance(dependent=ON, name='instLrand', part=partLrand)
283
284 mymodel.rootAssembly.rotate(angle=90.0, axisDirection=(1.0, 0.0, 0.0),
285     axisPoint=(0.0, 0.0, 0.0), instanceList=('instSchieber', 'instLrand', 'instBoden'))
286 mymodel.rootAssembly.translate(instanceList=('instSchieber', ),
287     vector=(xpos_schieber+schieber_dimensions[0]/2.0, 0.0, 0.0))
288
289 mymodel.rootAssembly.Surface(name='surfaceSchieber', side1Faces=instSchieber.faces)
290 mymodel.rootAssembly.Surface(name='surfaceLrand', side1Faces=instLrand.faces)
291
292 mymodel.RigidBody(bodyRegion=instSchieber.sets['setAll'],
293     name='RigidSchieber', refPointRegion=instSchieber.sets['setRP'])
294 mymodel.RigidBody(bodyRegion=instLrand.sets['setAll'],
295     name='RigidLrand', refPointRegion=instLrand.sets['setRP'])
296
```

```
297
298 # ---------------------- Step: Initiale Bedingungen und Anfangsrandbedingungen -------------------- #
299 #
300 sand_wichte = verwendeteBodenwerte[0][0] * g
301 mymodel.MaterialAssignment(name='Materialzuweisung', useFields=False,
302     assignmentList=((instBoden.sets['setSand'], (1, )), ), instanceList=(instBoden, ), )
303 mymodel.GeostaticStress(lateralCoeff1=verwendeteBodenwerte[0][1], lateralCoeff2=None,
304     name='Bodenspannung', region=instBoden.sets['setAll'], stressMag1=-geostatische_nullspannung,
305     stressMag2=-geostatische_nullspannung-sand_wichte*bodenbereich[1],
306     vCoord1=bodenbereich[1], vCoord2=0.0)
307
308 mymodel.ContactExp(createStepName='Initial', name='Kontaktinteraktion')
309 interaktion = mymodel.interactions['Kontaktinteraktion']
310 interaktion.includedPairs.setValuesInStep(stepName='Initial', useAllstar=ON)
311 interaktion.contactPropertyAssignments.appendInStep(
312     assignments=((GLOBAL, SELF, 'Kontakt'), ), stepName='Initial')
313 interaktion.excludedPairs.setValuesInStep(stepName='Initial',
314     addPairs=((mymodel.rootAssembly.surfaces['surfaceSchieber'],
315     mymodel.rootAssembly.surfaces['surfaceLrand']), ))
316
317 mymodel.VelocityBC(amplitude=UNSET, createStepName='Initial', distributionType=UNIFORM,
318     fieldName='', localCsys=None, name='bcBodenWandX', region=instBoden.sets['setWandX'],
319     v1=0.0, v2=UNSET, v3=UNSET, vr1=UNSET, vr2=UNSET, vr3=UNSET)
320 mymodel.VelocityBC(amplitude=UNSET, createStepName='Initial', distributionType=UNIFORM,
321     fieldName='', localCsys=None, name='bcBodenWandY', region=instBoden.sets['setWandY'],
322     v1=UNSET, v2=0.0, v3=UNSET, vr1=UNSET, vr2=UNSET, vr3=UNSET)
323 mymodel.VelocityBC(amplitude=UNSET, createStepName='Initial', distributionType=UNIFORM,
324     fieldName='', localCsys=None, name='bcBodenUnterseite', region=instBoden.sets['setUnterseite'],
325     v1=UNSET, v2=UNSET, v3=0.0, vr1=UNSET, vr2=UNSET, vr3=UNSET)
326 mymodel.VelocityBC(amplitude=UNSET, createStepName='Initial', distributionType=UNIFORM,
327     fieldName='', localCsys=None, name='bcSchieber', region=instSchieber.sets['setRP'],
328     v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0)
329 mymodel.VelocityBC(amplitude=UNSET, createStepName='Initial', distributionType=UNIFORM,
330     fieldName='', localCsys=None, name='bcLrand', region=instLrand.sets['setRP'],
331     v1=0.0, v2=0.0, v3=0.0, vr1=0.0, vr2=0.0, vr3=0.0)
332
333
334 # ------------------------------------ Step: Anfangszustand ------------------------------------ #
335 #
336 mymodel.ExplicitDynamicsStep(name='Anfangszustand', previous='Initial', scaleFactor=timescaling,
337     timePeriod=zeit_anfangszustand, linearBulkViscosity=0.48, quadBulkViscosity=1.2)
338 mymodel.Gravity(comp3=-g, createStepName='Anfangszustand', distributionType=UNIFORM,
339     field='', name='Schwerkraft')
340
341 mymodel.fieldOutputRequests.changeKey(fromName='F-Output-1', toName='FieldOutAll')
342 mymodel.fieldOutputRequests['FieldOutAll'].setValues(variables=('U', ),
343     numIntervals=numoutputs_initial_step)
344 mymodel.FieldOutputRequest(createStepName='Anfangszustand', name='FOut-Kontakt', rebar=EXCLUDE,
345     numIntervals=numoutputs_initial_step, sectionPoints=DEFAULT, variables=('CSTRESS', 'CFORCE'))
346 mymodel.FieldOutputRequest(createStepName='Anfangszustand', name='FOut-BodenAll',
347     numIntervals=numoutputs_initial_step, region=instBoden.sets['setAll'], rebar=EXCLUDE,
348     sectionPoints=DEFAULT, variables=('A', 'SVAVG', 'ER', 'SDV', 'EVF'))
349 mymodel.FieldOutputRequest(createStepName='Anfangszustand', name='FOut-SchieberRP',
350     numIntervals=numoutputs_initial_step, region=instSchieber.sets['setRP'], rebar=EXCLUDE,
351     sectionPoints=DEFAULT, variables=('RF', 'RM'))
352
353 mymodel.historyOutputRequests.changeKey(fromName='H-Output-1', toName='HistoryOutAll')
354 mymodel.historyOutputRequests['HistoryOutAll'].setValues(numIntervals=numoutputs_initial_step)
355
356
357
```

```
# ------------------------------------ Steps: Schiebung1-3 ---------------------------------- #
#
mymodel.ExplicitDynamicsStep(name='Schiebung1', previous='Anfangszustand',
    linearBulkViscosity=0.48, quadBulkViscosity=1.2, scaleFactor=timescaling, timePeriod=reftime)
mymodel.boundaryConditions['bcSchieber'].setValuesInStep(stepName='Schiebung1', v1=-refweg/reftime)

for outputrequestname in ['FieldOutAll', 'FOut-Kontakt', 'FOut-BodenAll', 'FOut-SchieberRP']:
    mymodel.fieldOutputRequests[outputrequestname].setValuesInStep(stepName='Schiebung1',
        numIntervals=numoutputs_per_step)

mymodel.ExplicitDynamicsStep(name='Schiebung2', previous='Schiebung1',
    linearBulkViscosity=0.48, quadBulkViscosity=1.2, scaleFactor=timescaling, timePeriod=reftime)
mymodel.boundaryConditions['bcSchieber'].setValuesInStep(stepName='Schiebung2', v1=refweg/reftime)

mymodel.ExplicitDynamicsStep(name='Schiebung3', previous='Schiebung2',
    linearBulkViscosity=0.48, quadBulkViscosity=1.2, scaleFactor=timescaling, timePeriod=reftime)
mymodel.boundaryConditions['bcSchieber'].setValuesInStep(stepName='Schiebung3', v1=-refweg/reftime)


# ------------------------------------------- Job ------------------------------------------- #
#
mdb.Job(name=modelname, model=modelname, description='',
      type=ANALYSIS, atTime=None, waitMinutes=0, waitHours=0, queue=None,
      memory=90, memoryUnits=PERCENTAGE, explicitPrecision=DOUBLE_PLUS_PACK,
      nodalOutputPrecision=FULL, echoPrint=OFF, modelPrint=OFF, contactPrint=OFF,
      historyPrint=OFF, userSubroutine=userroutine, scratch='',
      resultsFormat=ODB, parallelizationMethodExplicit=DOMAIN, numDomains=2,
      activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=2)
myjob = mdb.jobs[modelname]


# --------------------------- Nachbearbeitung im Schluesselworteditor --------------------------- #
mymodel.keywordBlock.synchVersions(storeNodesAndElements=False)
changed = False
for idx, text in enumerate(mymodel.keywordBlock.sieBlocks):
    if (text == '*Contact, op=NEW'):
        mymodel.keywordBlock.replace(idx, '*Contact') # Kontakt ohne op=NEW
    if ((text[0:19] == '*Initial Conditions') and (not changed)):
        # Fuer jeden der angeforderten SDV muss eine Loesung angegeben werden
        # In der inp-Datei duerfen nicht mehr als 8 Werte pro Zeile stehen!
        neuerBlock = '**\n*Initial Conditions, type=SOLUTION\ninstBoden.setSand, ' \
            + str(aktuelle_porenzahl) + ', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0\n' \
            + ' 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0\n' \
            + ' 0.0, 0.0, 0.0, 0.0, 0.0\n'
        neuerBlock += '**\n*Initial Conditions, type=SOLUTION\ninstBoden.setLockereZelle, ' \
            + str(max_porenzahl) + ', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0\n' \
            + ' 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0\n' \
            + ' 0.0, 0.0, 0.0, 0.0, 0.0\n'
        mymodel.keywordBlock.insert(idx, neuerBlock)
        changed = True

myjob.writeInput(consistencyChecking=OFF)


# ------------------------------------------------------------------------------------------------ #
myviewport = session.viewports[session.currentViewportName]
myviewport.setValues(displayedObject=mymodel.rootAssembly)
myviewport.view.rotate(xAngle=-90, yAngle=0, zAngle=15, mode=MODEL)

Log('Modellerstellung: %s sec' %(time.time()-startZeit))
```

G Zur Bestimmung der Unsicherheit von Laborversuchen

Unsicherheit der Korndichte

Für die Bestimmung der Korndichte wird das Gewicht von Proben und Behältern gemessen sowie die Wassertemperatur bestimmt (siehe Bild G.1). Aus der Wassertemperatur wird die Dichte von destillierten und entlüfteten Wasser nach DIN EN ISO 17892-3:2016, S. 14, mit $\rho = h(T)$ abgeschätzt. Die Anzahl der Nachkommastellen der dortigen Tabellenwerte legen eine Genauigkeit von $10\,\mathrm{g/m^3}$ nahe.[1]

Nach kurzer Umformung ergibt sich für die Korndichte (basierend auf den Eingangswerten aus Bild G.1):

$$\rho_S = f(m_{t,B}, m_B, T_W, m_{t,P}, m_P, T_{W,P}) = \frac{m_{t,B} - m_P}{(m_P - m_{t,P}) \cdot h(T_W) - (m_B - m_{t,B}) \cdot h(T_{W,P})}$$

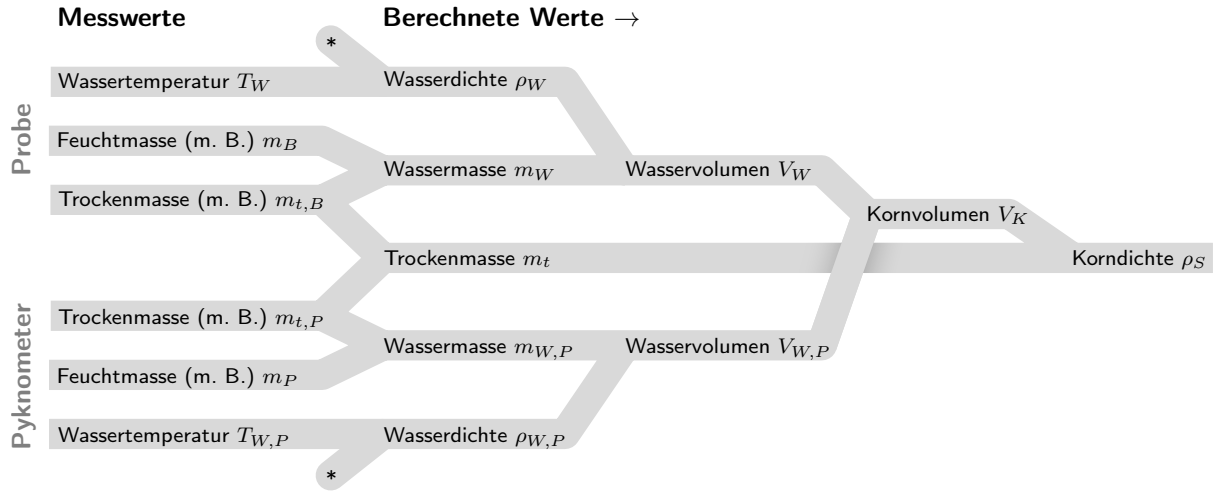


Bild G.1: Gemessene und berechnete Werte bei der Bestimmung der Korndichte (nach DIN EN ISO 17892-3:2016). Die Darstellung zeigt einen möglichen Berechnungsverlauf von links nach rechts. Wie in der Norm empfohlen, werden die Wasserdichten mit einer Hilfsformel $h(T)$ über die Temperatur abgeschätzt (mit $*$ markiert).

[1] Die Formel zur Abschätzung der Dichte von destillierten Wasser wird in DIN EN ISO 17892-3:2016, S. 14, als „korrigiert für den Auftrieb in Luft“ bezeichnet. Grundsätzlich könnten die physikalischen Eigenschaften von Wasser auch mit dem IAPWS-95-Modells aus Wagner und Pruß (2002) abgeschätzt werden.

mit der Abschätzung (basierend auf der Wassertemperatur T in Grad Celsius).

$$h(T) = \frac{1}{1 + \left[(2.31T - 2)^2 - 182\right] \cdot 10^{-6}} \quad .$$

Die partiellen Ableitungen für spätere Abschätzungen ergeben sich zu

$$\begin{bmatrix} \frac{\partial f}{\partial m_{t,B}} \\ \frac{\partial f}{\partial m_P} \\ \frac{\partial f}{\partial m_{t,P}} \\ \frac{\partial f}{\partial m_B} \\ \frac{\partial f}{\partial T_W} \\ \frac{\partial f}{\partial T_{W,P}} \end{bmatrix} = \begin{bmatrix} \frac{(m_P - m_{t,P}) \cdot h(T_W) + (m_P - m_B) \cdot h(T_{W,P})}{\left[\left(m_P - m_{t,P}\right) \cdot h(T_W) - \left(m_B - m_{t,B}\right) \cdot h(T_{W,P})\right]^2} \\ \frac{(m_{t,P} - m_{t,B}) \cdot h(T_W) + (m_B - m_{t,B}) \cdot h(T_{W,P})}{\left[\left(m_P - m_{t,P}\right) \cdot h(T_W) - \left(m_B - m_{t,B}\right) \cdot h(T_{W,P})\right]^2} \\ \frac{(m_{t,B} - m_P)}{\left[\left(m_P - m_{t,P}\right) \cdot h(T_W) - \left(m_B - m_{t,B}\right) \cdot h(T_{W,P})\right]^2} \cdot h(T_W) \\ \frac{(m_{t,B} - m_P)}{\left[\left(m_P - m_{t,P}\right) \cdot h(T_W) - \left(m_B - m_{t,B}\right) \cdot h(T_{W,P})\right]^2} \cdot h(T_{W,P}) \\ \frac{-(m_{t,B} - m_P)(m_P - m_{t,P})}{\left[\left(m_P - m_{t,P}\right) \cdot h(T_W) - \left(m_B - m_{t,B}\right) \cdot h(T_{W,P})\right]^2} \cdot \frac{\partial h(T_W)}{\partial T_W} \\ \frac{-(m_{t,B} - m_P)(m_{t,B} - m_B)}{\left[\left(m_P - m_{t,P}\right) \cdot h(T_W) - \left(m_B - m_{t,B}\right) \cdot h(T_{W,P})\right]^2} \cdot \frac{\partial h(T_{W,P})}{\partial T_{W,P}} \end{bmatrix} \quad \text{mit}$$

$$\frac{\partial h(T)}{\partial T} = \frac{10{,}6722T - 9{,}24}{(5{,}3361T^2 - 9{,}24T + 999822)^2} \quad .$$

Unsicherheit der lockersten und dichtesten Lagerung

Analog zum Vorgehen bei der Korndichte gilt für die Bestimmung der lockersten und dichtesten Lagerung der Zusammenhang

$$\rho_d = f(m_d, V) = \frac{m_d}{V}$$

und es ergeben sich die partiellen Ableitungen

$$\begin{bmatrix} \frac{\partial f}{\partial m_d} \\ \frac{\partial f}{\partial V} \end{bmatrix} = \begin{bmatrix} \frac{1}{V} \\ -\frac{m_d}{V^2} \end{bmatrix} \quad .$$

Für die Unsicherheit der Porenzahlen muss die zuvor bestimmte Unsicherheit der Korndichte berücksichtigt werden und es gilt

$$e = f(\rho_S, m_d, V) = \frac{\rho_S V}{m_d} - 1 \quad \text{und somit} \qquad \begin{bmatrix} \frac{\partial f}{\partial \rho_S} \\ \frac{\partial f}{\partial m_d} \\ \frac{\partial f}{\partial V} \end{bmatrix} = \begin{bmatrix} \frac{V}{m_d} \\ -\frac{\rho_S V}{m_d^2} \\ \frac{\rho_S}{m_d} \end{bmatrix} \quad .$$

Unsicherheit abgeleiteter Werte bei Ödometerversuchen

Bei Ödometerversuch wird die Druckspannung auf der Probe durch die angehängten Gewichten mit der vertikalen Stauchung bzw. der Porenzahl der Bodenprobe ins Verhältnis gesetzt. Die Unsicherheit der Druckspannung ergibt hauptsächlich sich aus den Gewichten und der Oberfläche der Probe

$$\sigma_1 = f(m, A) = \frac{mg}{A} \quad \text{und somit} \qquad \begin{bmatrix} \frac{\partial f}{\partial m} \\ \frac{\partial f}{\partial A} \end{bmatrix} = \begin{bmatrix} \frac{g}{A} \\ \frac{mg}{A^2} \end{bmatrix} \quad .$$

Die Unsicherheit der Anfangsporenzahl ergibt sich wie bei der lockersten und dichtesten Lagerung. Für Stauchung ε und die Porenzahl e gilt

$$\varepsilon = f(s, h_{\text{ini}}) = \frac{s}{h_{\text{ini}}} \quad \text{und somit} \qquad \begin{bmatrix} \frac{\partial f}{\partial s} \\ \frac{\partial f}{\partial h_{\text{ini}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{h_{\text{ini}}} \\ \frac{s}{h_{\text{ini}}^2} \end{bmatrix} \quad \text{sowie}$$

$$e = f(e_{\text{ini}}, s, h_{\text{ini}}) = e_{\text{ini}} - \frac{s}{h_{\text{ini}}}\,(1 + e_{\text{ini}}) \quad \text{und somit} \qquad \begin{bmatrix} \frac{\partial f}{\partial e_{\text{ini}}} \\ \frac{\partial f}{\partial s} \\ \frac{\partial f}{\partial h_{\text{ini}}} \end{bmatrix} = \begin{bmatrix} 1 - \frac{s}{h_{\text{ini}}} \\ -\frac{1+e_{\text{ini}}}{h_{\text{ini}}} \\ \frac{(e_{\text{ini}}+1)\cdot s}{h_{\text{ini}}^2} \end{bmatrix} \quad .$$

Unsicherheit abgeleiteter Werte bei Triaxialversuchen

Die Abschätzung der Unsicherheit der Stauchung ε ist analog zum Ödometerversuch. Für die Unsicherheit der Axialspannung σ_1 gilt

$$\sigma_1 = f(F_1, \sigma_3, s, \Delta V_A, h_{\text{ini}}, d_0, \Delta V_K) = \frac{4F_1\,(h_{\text{ini}} - s)}{\pi h_{\text{ini}} d_0^2 + 4\,(\Delta V_A - \Delta V_K)} + \sigma_3$$

und somit

$$\begin{bmatrix} \frac{\partial f}{\partial F_1} \\ \frac{\partial f}{\partial \sigma_3} \\ \frac{\partial f}{\partial s} \\ \frac{\partial f}{\partial \Delta V_A} \\ \frac{\partial f}{\partial h_{\text{ini}}} \\ \frac{\partial f}{\partial d_0} \\ \frac{\partial f}{\partial \Delta V_K} \end{bmatrix} = \begin{bmatrix} \frac{4(h_{\text{ini}}-s)}{\pi h_{\text{ini}} d_0^2 + 4(\Delta V_A - \Delta V_K)} \\ 1 \\ \frac{-4F_1}{\pi h_{\text{ini}} d_0^2 + 4(\Delta V_A - \Delta V_K)} \\ \frac{16F_1\cdot(s-h_{\text{ini}})}{\left[\pi h_{\text{ini}} d_0^2 + 4(\Delta V_A - \Delta V_K)\right]^2} \\ \frac{4F_1\cdot\left[\pi d_0^2 s + 4(\Delta V_A - \Delta V_K)\right]}{\left[\pi h_{\text{ini}} d_0^2 + 4(\Delta V_A - \Delta V_K)\right]^2} \\ \frac{8F_1\pi h_{\text{ini}} d_0\cdot(h_{\text{ini}}-s)}{\left[\pi h_{\text{ini}} d_0^2 + 4(\Delta V_A - \Delta V_K)\right]^2} \\ \frac{16F_1\cdot(h_{\text{ini}}-s)}{\left[\pi h_{\text{ini}} d_0^2 + 4(\Delta V_A - \Delta V_K)\right]^2} \end{bmatrix} \quad .$$

Für den Sinus des Reibungswinkels gilt

$$\sin(\varphi) = f(\sigma_1, \sigma_3, u) = \frac{\sigma_1 - \sigma_3}{\sigma_1 + \sigma_3 - 2u} \quad \text{und somit} \quad \begin{bmatrix} \frac{\partial f}{\partial \sigma_1} \\ \frac{\partial f}{\partial \sigma_3} \\ \frac{\partial f}{\partial u} \end{bmatrix} = \begin{bmatrix} \frac{2(\sigma_3 - u)}{(\sigma_1 + \sigma_3 - 2u)^2} \\ \frac{2(u - \sigma_1)}{(\sigma_1 + \sigma_3 - 2u)^2} \\ \frac{2(\sigma_1 - \sigma_3)}{(\sigma_1 + \sigma_3 - 2u)^2} \end{bmatrix} .$$

Für die Porenzahl wird mit der Unsicherheit der Korndichte das Folgende bestimmt

$$e = f(\rho_S, \Delta V_A, h_{\text{ini}}, d_0, \Delta V_K, m_d) = \frac{\rho_S \left[h_{\text{ini}} \pi d_0^2 + 16 \left(\Delta V_A - \Delta V_K\right)\right]}{4 m_d} - 1$$

und somit

$$\begin{bmatrix} \frac{\partial f}{\partial \rho_S} \\ \frac{\partial f}{\partial \Delta V_A} \\ \frac{\partial f}{\partial h_{\text{ini}}} \\ \frac{\partial f}{\partial d_0} \\ \frac{\partial f}{\partial \Delta V_K} \\ \frac{\partial f}{\partial m_d} \end{bmatrix} = \begin{bmatrix} \frac{h_{\text{ini}} \pi d_0^2 + 16(\Delta V_A - \Delta V_K)}{4 m_d} \\ \frac{4 \rho_S}{m_d} \\ \frac{\rho_S \pi d_0^2}{4 m_d} \\ \frac{\rho_S h_{\text{ini}} \pi d_0}{2 m_d} \\ \frac{-4 \rho_S}{m_d} \\ \frac{-\rho_S \left[h_{\text{ini}} \pi d_0^2 + 16(\Delta V_A - \Delta V_K)\right]}{4 m_d^2} \end{bmatrix} .$$

H Eigene Abschätzungen

Abschätzung prozentualer Randporenanteil

Eine einfache Abschätzung zum Randporenanteil einer Probe wird an einem Volumen mit gleich großen Kugeln durchgeführt. Dazu wird angenommen, dass Kugeln mit dem Radius r am Rand eines Probenbehälters mit Radius R nebeneinander angeordnet sind (siehe Bild H.1). Die Kugeln in den darüber/darunter liegenden Ebenen werden entweder in der gleichen Position angenommen (Abschätzung für einen locker gelagerten Randbereich mit größeren Randporen) oder um eine halbe Kugellänge versetzt (Abschätzung für einen dicht gelagerten Randbereich mit kleineren Randporen).

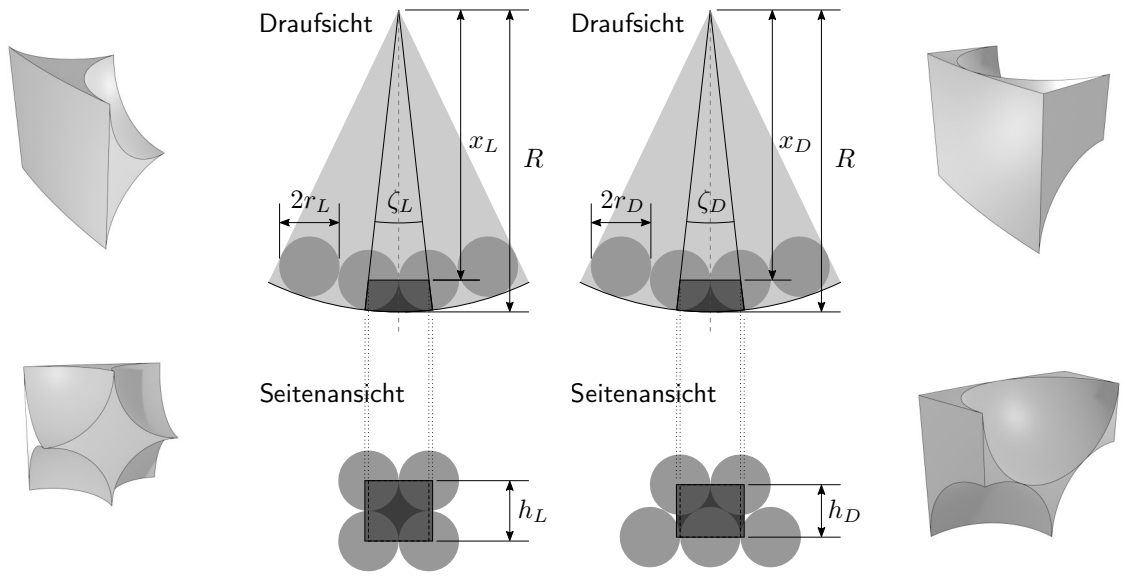


Bild H.1: Abschätzung des Membrandurchdringungseffekts am Beispiel von gleich großen Kugeln (Radius r) in einem Behälter mit Radius R. In der linken Bildhälfte ist eine lockere Lagerung an der Behälterwand angenommen, in der rechten Bildhälfte eine dichte Lagerung. Für die Berechnung des Randporenanteils ψ wird jeweils das dunkel markierte Volumen V_{RP} an der Behälterwand (ohne die Anteile der Kugeln) bestimmt und ins Verhältnis zum dazugehörigen Anteil $V_{\mathrm{S}} = hR^2\zeta/2$ gesetzt.

Über die Volumen im dargestellten Abschnitt eines Zylindersektors mit $h_L = 2r_L$ bzw. $h_D = \sqrt{3} \cdot r_D$ wird der Anteil n_{RP} der Randporen am Gesamtvolumen bestimmt. Mit $\kappa = r/R$ gilt

$$\zeta = 2 \arcsin\left(\frac{\kappa}{1-\kappa}\right) \quad , \qquad x = R\sqrt{1-2\kappa} \quad \text{und} \qquad V_{\mathrm{S}} = \frac{\zeta}{2} h R^2 \quad .$$

Für den locker gelagerten Randbereich ergeben sich

$$V_{\mathrm{ref},L} = h_L R^2 \left(\frac{\zeta_L}{2} - \kappa_L \sqrt{1-2\kappa_L}\right) \quad , \qquad V_{\mathrm{ref}}^{\infty} = 4 r_L^3 \quad ,$$

$$V_{\mathrm{RP},L} = V_{\mathrm{ref}} - \frac{2}{3}\left(\pi + \zeta_L\right) r_L^3 \quad , \qquad V_{\mathrm{RP},L}^{\infty} = 2\left(2 - \frac{\pi}{3}\right) r_L^3 \quad ,$$

$$n_{\mathrm{RP,max},L} = \frac{V_{\mathrm{RP},L}}{V_{\mathrm{ref},L}} \quad \text{und} \qquad n_{\mathrm{RP,max},L}^{\infty} = 1 - \frac{\pi}{6} \approx 0{,}476$$

mit dem Volumen der Randporen V_{RP} und dem Porenanteil $n_{\mathrm{RP}} = V_{\mathrm{RP}}/V_{\mathrm{ref}}$ im Referenzvolumen am Rand. Die Werte mit hochgestellten ∞ sind Grenzwerte für eine flache Wand ($R \to \infty$). Analog ergeben sich für den dicht gelagerten Randbereich

$$V_{\mathrm{ref},D} = h_D R^2 \left(\frac{\zeta_D}{2} - \kappa_D \sqrt{1-2\kappa_D}\right) \quad , \qquad V_{\mathrm{ref},D}^{\infty} = 2\sqrt{3} r_D^3 \quad ,$$

$$V_{\mathrm{RP},D} = V_{\mathrm{ref}} - \frac{\zeta + \pi}{3} r_D^3 + \frac{\pi}{6}\left(R - x_D - 3r_D\right)\left(R - x_D\right)^2 \quad ,$$

$$V_{\mathrm{RP},D}^{\infty} = 2\left(\sqrt{3} - \frac{\pi}{3}\right) r_D^3 \quad ,$$

$$n_{\mathrm{RP,min},D} = \frac{V_{\mathrm{RP},D}}{V_{\mathrm{ref},D}} \quad \text{und} \qquad n_{\mathrm{RP,min},D}^{\infty} = 1 - \frac{\pi}{3\sqrt{3}} \approx 0{,}395 \quad .$$

Mit $\psi = V_{\mathrm{RP}}/V_{\mathrm{S}}$ kann der Randporenanteil in Bezug auf das gesamte Behältervolumen abgeschätzt werden (siehe Bild H.2 und Tabelle H.1). Dabei ist zu sehen, dass für den betrachteten Triaxialversuch der Effekt der Randporen bei der Glaskugelmischung mit 5 bis 10 % auch fünf bis zehnmal höher ist als bei Hamburger Sand.

Tabelle H.1: Randporenanteile bezogen auf die Gesamtprobe für einen Triaxialversuch mit $r_{\mathrm{ref}} = 25\,\mathrm{mm}$. Für die Glaskugelmischung werden die Maximalwerte angesetzt, also $4{,}76 \leq \psi \leq 9{,}60$.

Material	Radius $d_{20}/2$ (mm)	κ (-)	n_{RP} (-)	ψ (%)
SiLibeads 5 mm	2,5	10,0	0,405 bis 0,488	7,97 bis 9,60
SiLibeads 3 mm	1,5	$16{,}\overline{6}$	0,401 bis 0,483	4,76 bis 5,74
Hamburger Sand	0,28	89,3	0,396 bis 0,478	0,89 bis 1,07
Karlsruher Feinsand	0,05	500	0,396 bis 0,477	0,16 bis 0,19

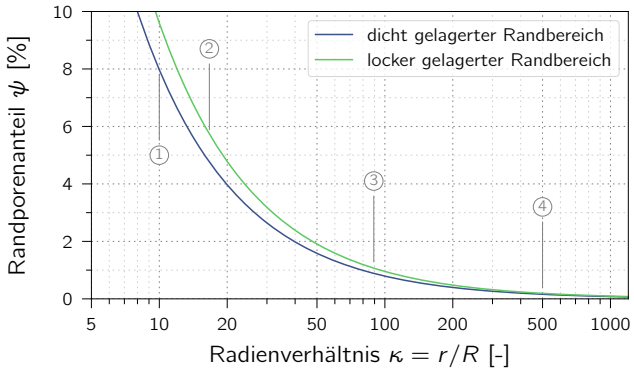


Bild H.2: Abgeschätzte Randporenanteile ψ bezogen auf das Volumen der Gesamtprobe in Abhängigkeit vom Radienverhältnis κ zwischen Körner und Versuchsbehälter für ideale Kugellagerungen. Die markierten Stellen entsprechen den untersuchten Materialien im Triaxialversuch mit $R = 25\,\text{mm}$. (1) Glaskugeln mit $r_l = 2{,}5\,\text{mm}$, (2) Glaskugeln mit $r_s = 1{,}5\,\text{mm}$, (3) Hamburger Sand mit $r_{20} \approx 0{,}28\,\text{mm}$ und (4) Karlsruher Feinsand mit $r_{20} \approx 0{,}05\,\text{mm}$.

Skizzen zu den idealisierten Verteilungsformen

Ergänzend zu Abschnitt 8.3 sind in Bild H.3 fast alle Situationen aufgelistet, die für die Abschätzung der idealisierten Verteilungsform in Bild 8.12 verwendet worden sind.

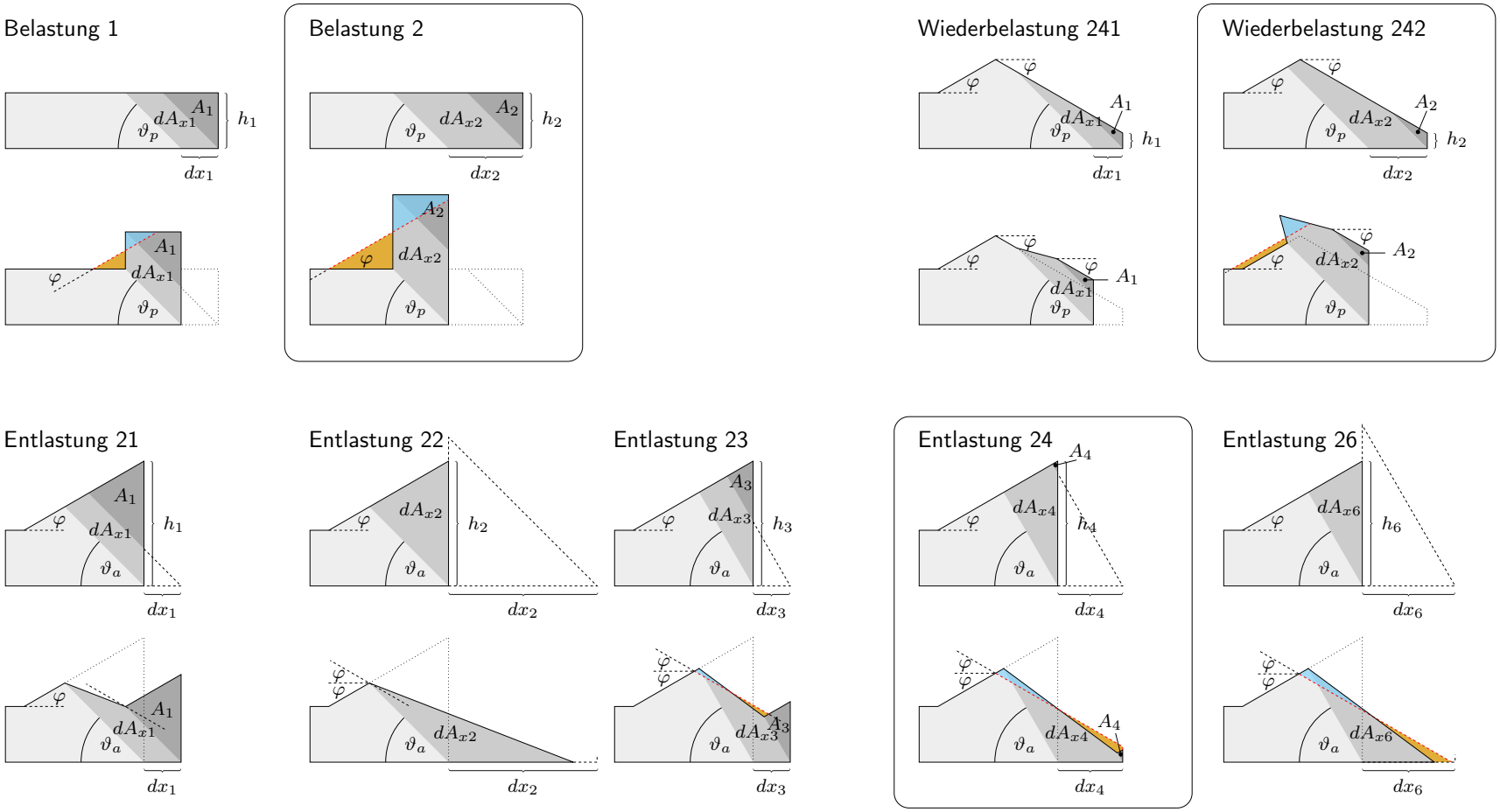


Bild H.3: Übersicht ausgewählter Be-, Ent- und Wiederbelastungssituationen für die Abschätzung der idealisierten Verteilungsform. Es wird eine horizontale Schrittlänge dx gewählt, die als geradlinige Scherfuge unter dem Winkel ϑ approximiert wird. Nach der Verformung wird ggf. die neue Fläche umgeformt (blau dargestellte Flächen zu orangenen Flächen), damit die freie Böschung maximal unter dem Winkel φ geneigt ist.

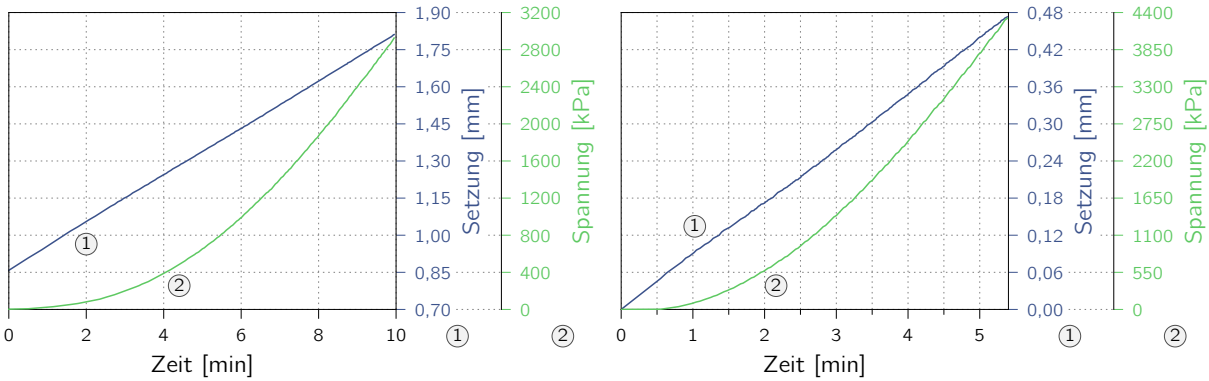
I Daten der Laborversuche

Hamburger Sand

Die in Abschnitt 6.2 vorgestellten Materialparameter für Hamburger Sand basieren u. a. auf den Daten von zwei Ödometerversuchen und sechs Triaxialversuchen. In Tabelle I.1 und Bild I.1 sind die aufbereiteten Messdaten der Ödometerversuche zu sehen. Die Triaxialversuche mit locker gelagerter Bodenprobe sind in Bild I.2 und Tabelle I.2 dargestellt. In Bild I.3 und Tabelle I.3 sind die Triaxialversuche mit dicht gelagerter Bodenprobe zu sehen. Bild I.4 zeigt die Nachrechnungen der Laborversuche mit dem Stoffmodell `Hypo-VW96` und den Materialparametern aus Tabelle 6.2 auf S. 119.

Tabelle I.1: Messdaten zu den Ödometerversuchen mit Hamburger Sand.

Versuch	h (mm)	d (mm)	m_d (g)	Δm (g)	$F_{\text{Startoffset}}$ (kN)
locker	22,5	70,0	130,36	464	0,010
dicht	19,4	70,0	130,36	464	0,0



(a) locker **(b)** dicht

Bild I.1: Messdaten zu den beiden Ödometerversuchen. Links mit locker gelagerten, rechts mit dicht gelagerten Hamburger Sand.

Tabelle I.2: Messdaten zu den drei drainierten Triaxialversuchen mit locker gelagerten Hamburger Sand: Probenhöhe h, -durchmesser d und Trockenmasse m_d der Bodenprobe beim Einbau sowie Backpressure-Volumen nach der Sättigung $V_{\mathrm{Back,S}}$, nach der Konsolidation $V_{\mathrm{Back,K}}$ und nach dem Abscheren $V_{\mathrm{Back,A}}$.

Versuch	h (mm)	d (mm)	m_d (g)	$V_{\mathrm{Back,S}}$ (mm^3)	$V_{\mathrm{Back,K}}$ (mm^3)	$V_{\mathrm{Back,A}}$ (mm^3)
1	48,5	50,0	148,72	3570	2494	2686
2	49,0	50,0	148,72	3687	1963	1678
3	48,8	50,0	148,72	5472	4436	2838

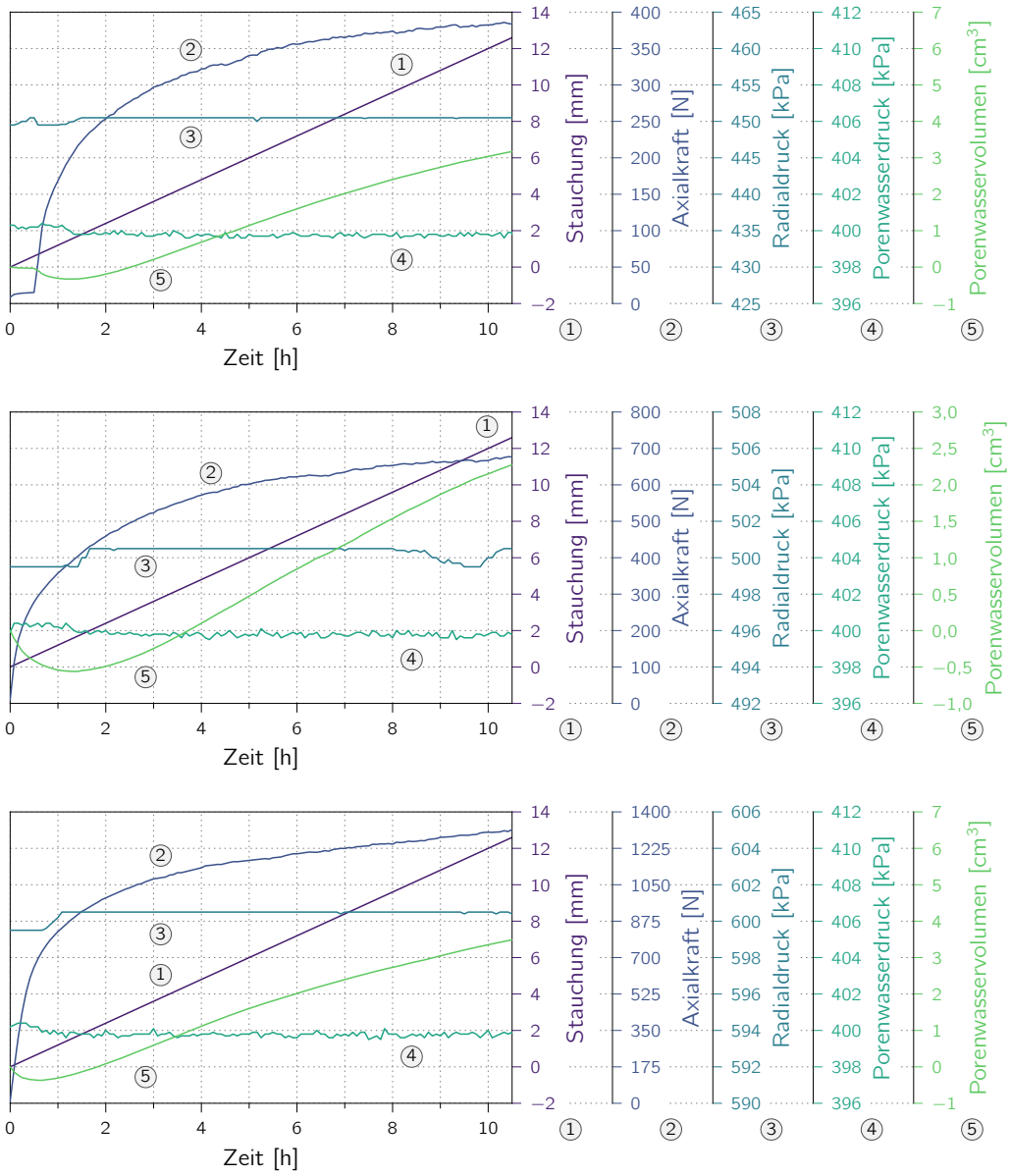


Bild I.2: Messdaten zu den drei drainierten Triaxialversuchen mit locker gelagerten Hamburger Sand.

Tabelle I.3: Messdaten zu den drei drainierten Triaxialversuchen mit dicht gelagerten Hamburger Sand: Probenhöhe h, -durchmesser d und Trockenmasse m_d der Bodenprobe beim Einbau sowie Backpressure-Volumen nach der Sättigung $V_{\mathrm{Back,S}}$, nach der Konsolidation $V_{\mathrm{Back,K}}$ und nach dem Abscheren $V_{\mathrm{Back,A}}$.

Versuch	h (mm)	d (mm)	m_d (g)	$V_{\mathrm{Back,S}}$ (mm^3)	$V_{\mathrm{Back,K}}$ (mm^3)	$V_{\mathrm{Back,A}}$ (mm^3)
1	50,0	50,0	169,29	5627	4956	8733
2	50,0	50,0	169,29	4522	3460	8983
3	50,0	50,0	169,29	5600	4019	9644

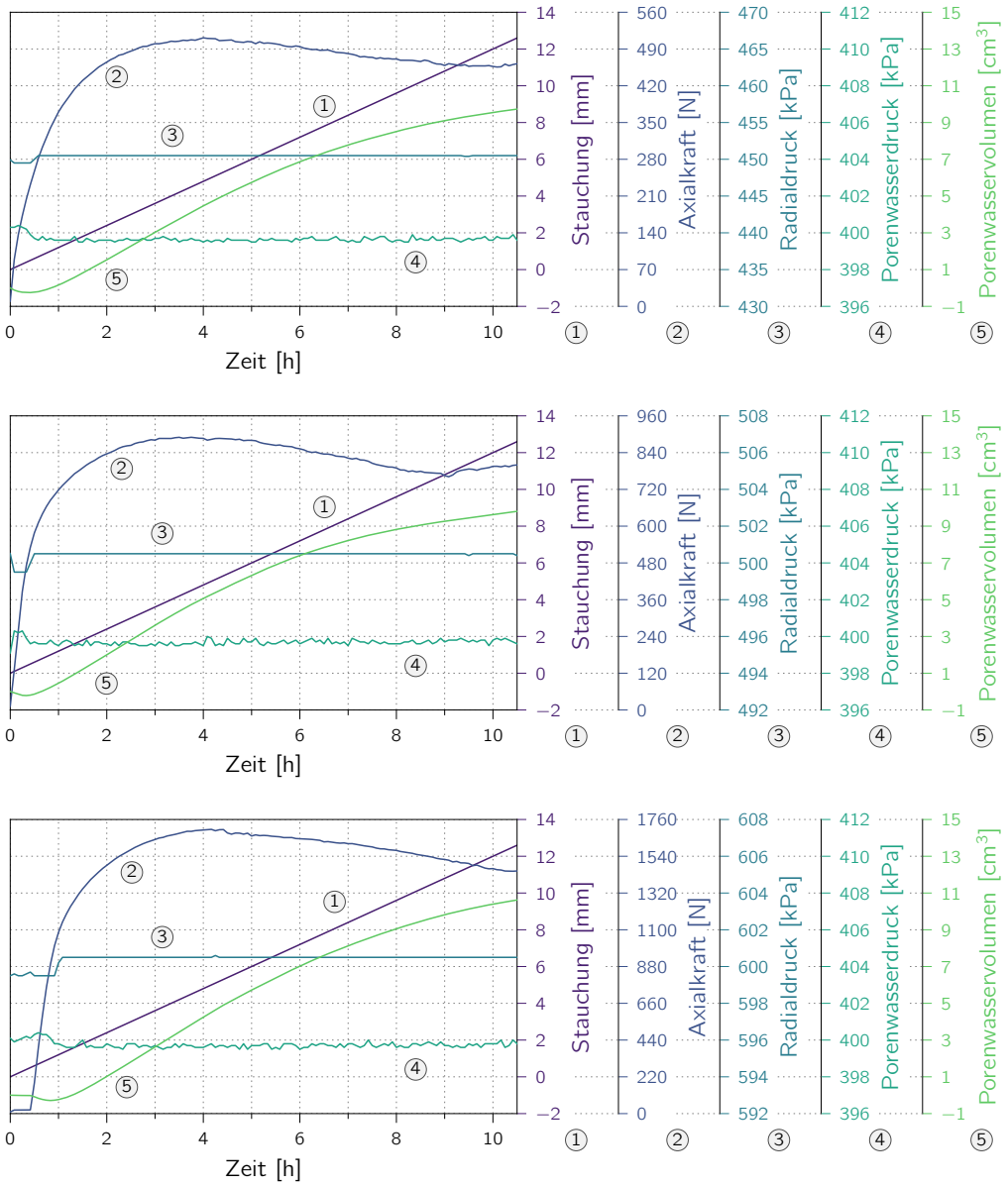


Bild I.3: Messdaten zu den drei drainierten Triaxialversuchen mit dicht gelagerten Hamburger Sand.

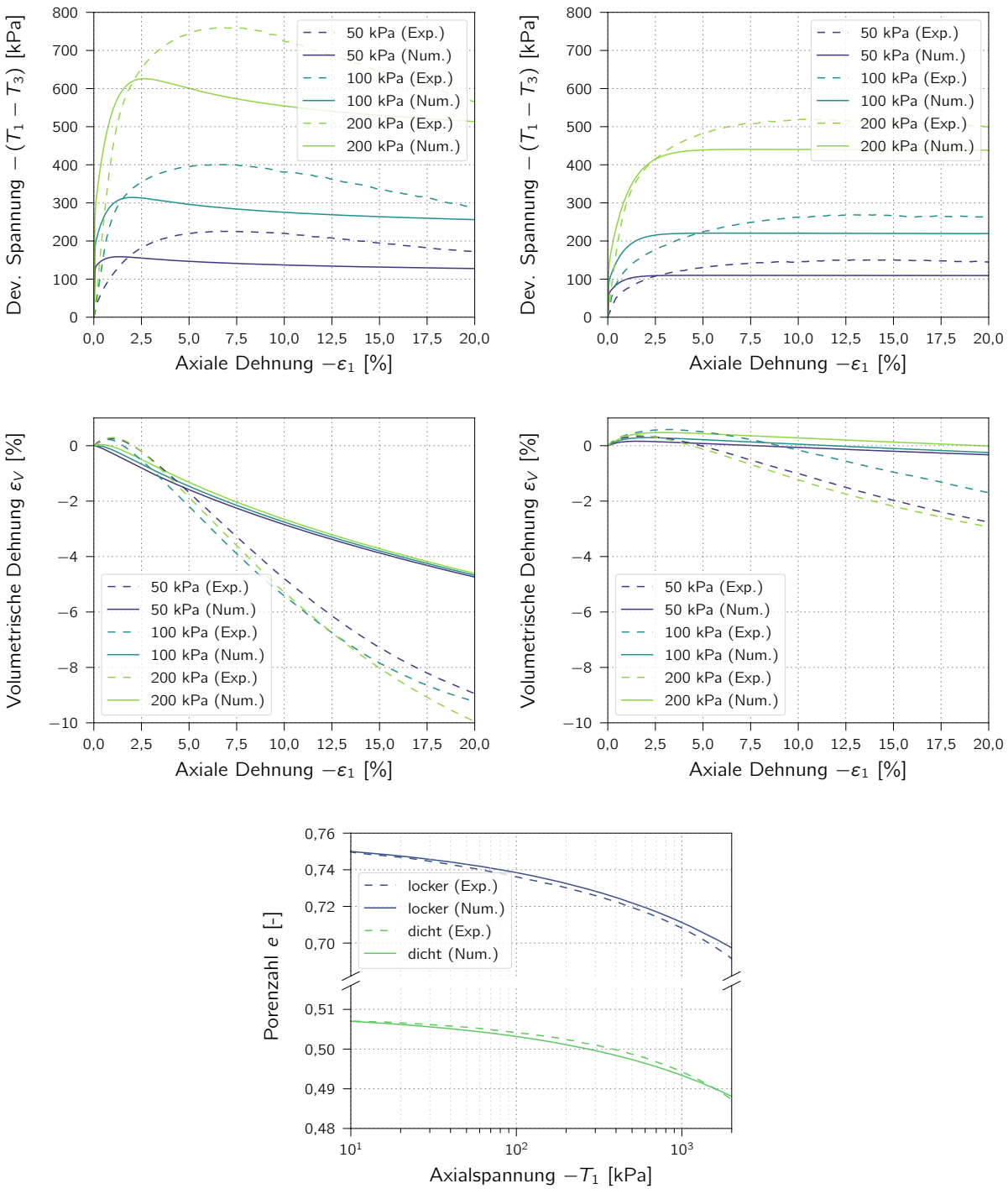


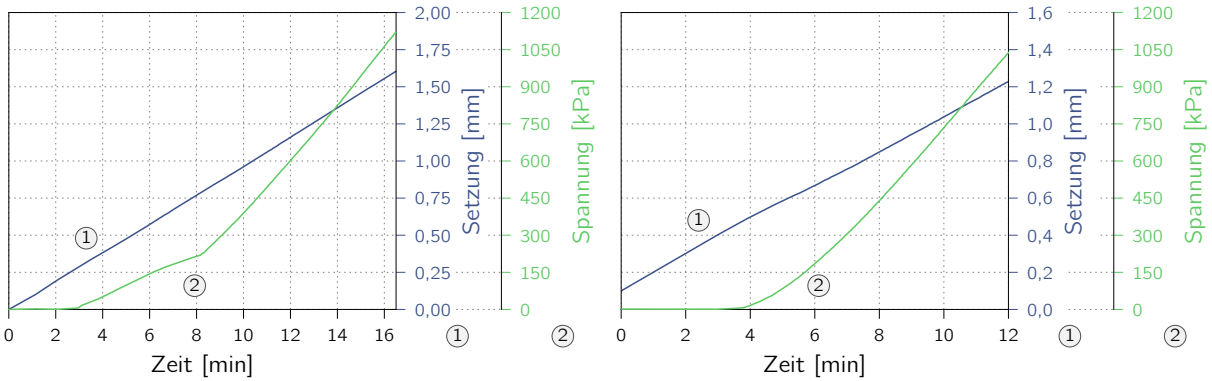
Bild I.4: Nachrechnung der im Labor durchgeführten Versuche mit Hamburger Sand mit dem Stoffmodell `Hypo-VW96`. Links mit einer dicht gelagerten Bodenprobe und den Anfangsporenzahlen 0,520 (50 kPa), 0,514 (100 kPa) und 0,506 (200 kPa), rechts mit einer locker gelagerten Bodenprobe und den Anfangsporenzahlen 0,750 (50 kPa), 0,711 (100 kPa) und 0,706 (200 kPa). Für die Ödometerversuche im unteren Bild werden die Anfangsporenzahlen 0,507 und 0,75 verwendet.

SiLibeads 3–5 mm

Analog sind auch zwei Ödometerversuche und sechs Triaxialversuche für die Bestimmung der Materialparameter der SiLibeads 3–5 mm notwendig. Die Daten der Ödometerversuche sind in Tabelle I.4 und Bild I.5 dargestellt. Für die Triaxialversuche mit locker gelagerter Bodenprobe sind die Daten in Bild I.6 und Tabelle I.5 zu sehen. Bild I.7 und Tabelle I.6 zeigen die aufbereiteten Messwerte der Triaxialversuche mit dicht gelagerter Bodenprobe. Bild I.8 zeigt die Nachrechnungen der Laborversuche mit dem Stoffmodell `Hypo-VW96` und den Materialparametern aus Tabelle 6.4 auf S. 121.

Tabelle I.4: Messdaten zu den Ödometerversuchen mit SiLibeads 3-5 mm.

Versuch	h (mm)	d (mm)	m_d (g)	Δm (g)	$F_{\text{Startoffset}}$ (kN)
locker	42,0	100,0	495,41	0,0	0,0
dicht	40,3	100,0	495,41	0,0	0,0



(a) locker **(b)** dicht

Bild I.5: Messdaten zu den beiden Ödometerversuchen. Links mit locker gelagerten, rechts mit dicht gelagerten SiLibeads 3–5 mm.

Tabelle I.5: Messdaten zu den drei drainierten Triaxialversuchen mit locker gelagerten SiLibeads 3–5 mm: Probenhöhe h, -durchmesser d und Trockenmasse m_d der Bodenprobe beim Einbau sowie Backpressure-Volumen nach der Sättigung $V_{\mathrm{Back,S}}$, nach der Konsolidation $V_{\mathrm{Back,K}}$ und nach dem Abscheren $V_{\mathrm{Back,A}}$.

Versuch	h (mm)	d (mm)	m_d (g)	$V_{\mathrm{Back,S}}$ (mm^3)	$V_{\mathrm{Back,K}}$ (mm^3)	$V_{\mathrm{Back,A}}$ (mm^3)
1	123,5	50,0	373,46	4753	−98	6340
2	123,3	50,0	373,44	3152	−3581	6377
3	123,4	50,0	373,47	2729	−6912	5332

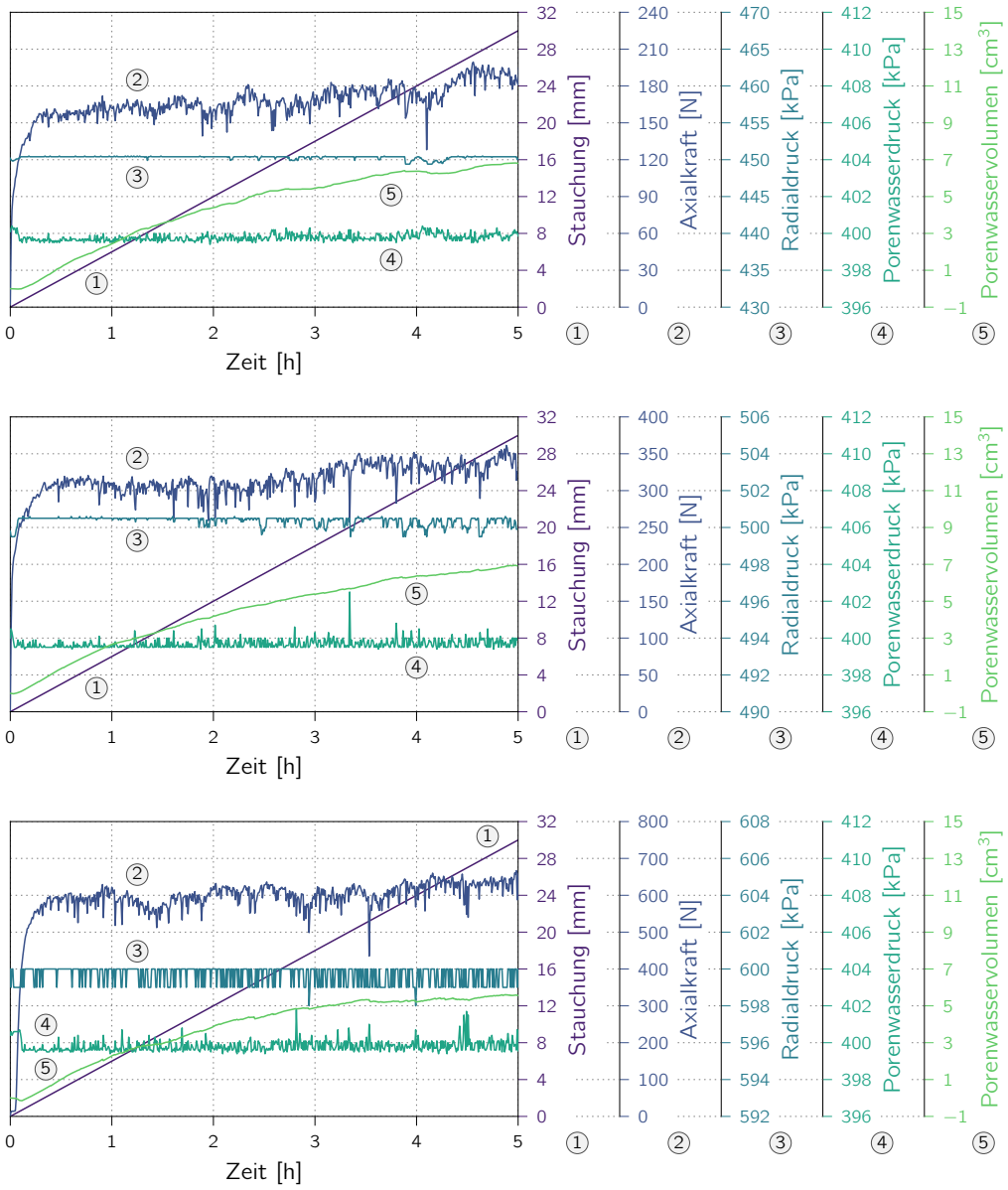


Bild I.6: Messdaten zu den drei drainierten Triaxialversuchen mit locker gelagerten SiLibeads 3–5 mm.

Tabelle I.6: Messdaten zu den drei drainierten Triaxialversuchen mit dicht gelagerten SiLibeads 3–5 mm: Probenhöhe h, -durchmesser d und Trockenmasse m_d der Bodenprobe beim Einbau sowie Backpressure-Volumen nach der Sättigung $V_{\mathrm{Back,S}}$, nach der Konsolidation $V_{\mathrm{Back,K}}$ und nach dem Abscheren $V_{\mathrm{Back,A}}$.

Versuch	h (mm)	d (mm)	m_d (g)	$V_{\mathrm{Back,S}}$ (mm^3)	$V_{\mathrm{Back,K}}$ (mm^3)	$V_{\mathrm{Back,A}}$ (mm^3)
1	123,6	50,0	387,29	5427	1671	15073
2	123,0	50,0	387,31	4678	−2228	17726
3	123,7	50,0	387,29	3688	−5094	19461

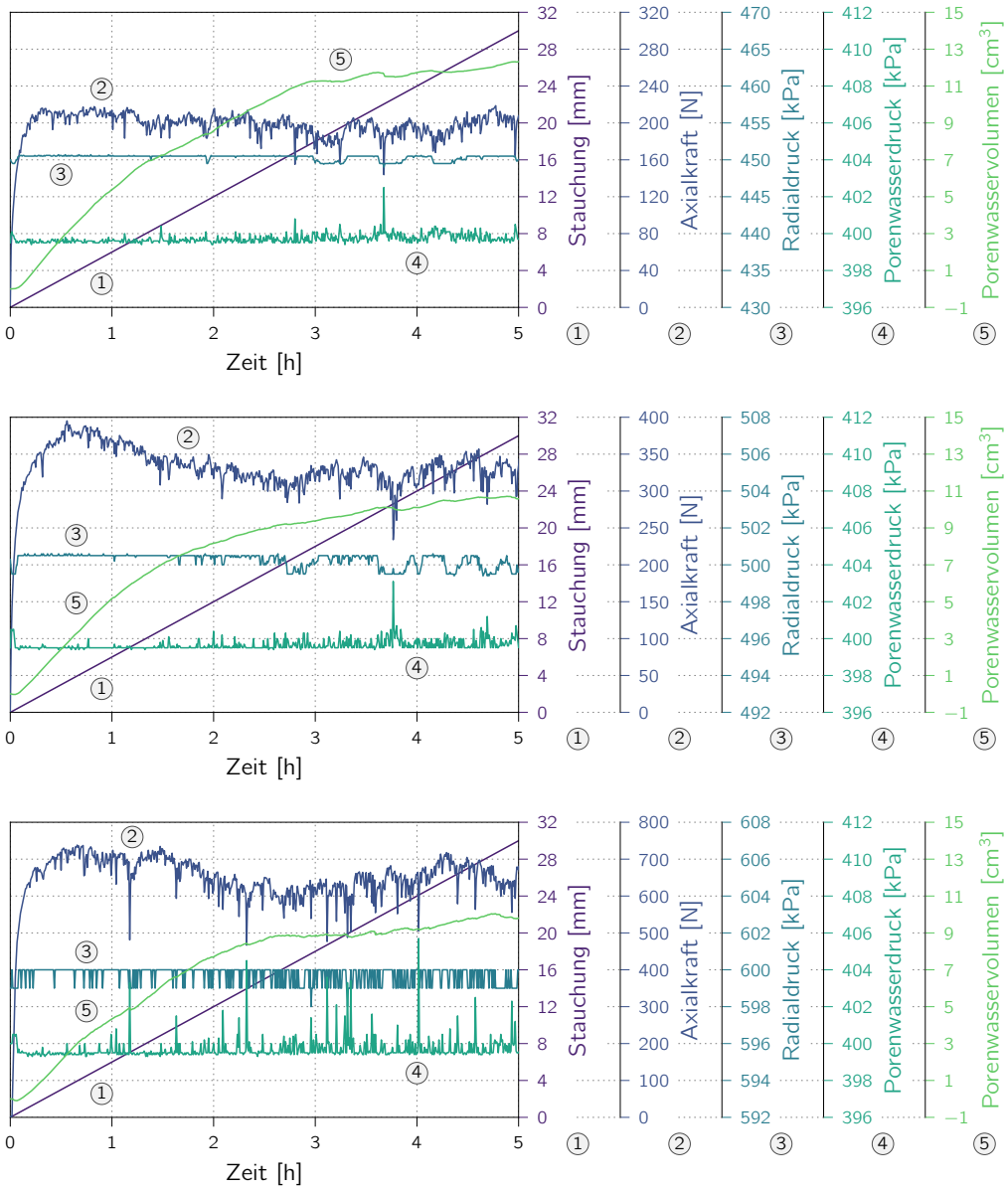


Bild I.7: Messdaten zu den drei drainierten Triaxialversuchen mit dicht gelagerten SiLibeads 3–5 mm.

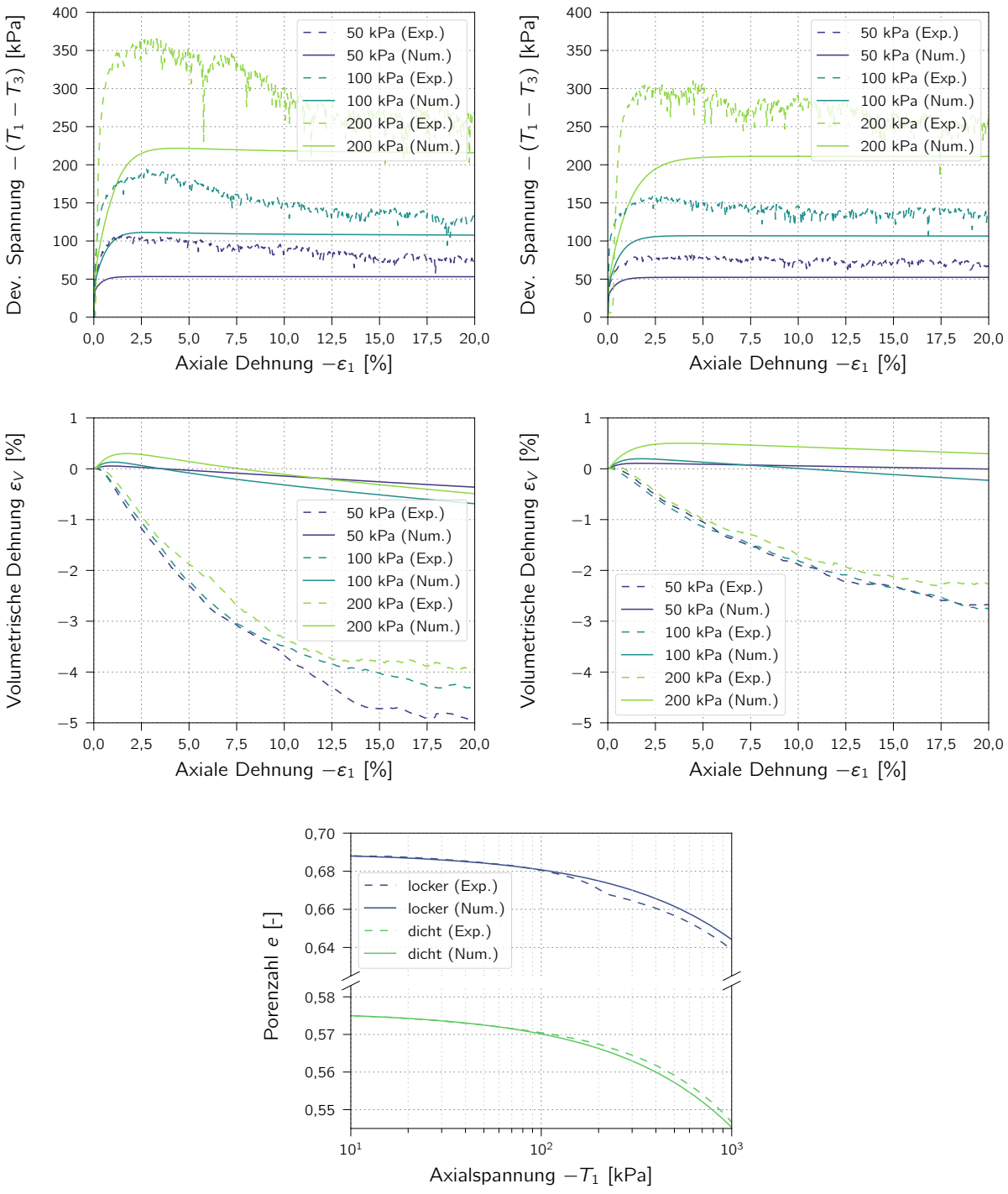


Bild I.8: Nachrechnung der im Labor durchgeführten Versuche mit SiLibeads 3–5 mm mit dem Stoffmodell `Hypo-VW96`. Links mit einer dicht gelagerten Bodenprobe und den Anfangsporenzahlen 0,575 (50 kPa), 0,546 (100 kPa) und 0,543 (200 kPa), rechts mit einer locker gelagerten Bodenprobe und den Anfangsporenzahlen 0,624 (50 kPa), 0,569 (100 kPa) und 0,588 (200 kPa). Für die Ödometerversuche im unteren Bild werden die Anfangsporenzahlen 0,575 und 0,688 verwendet.

J xMat Referenz

Die xMat ist unter »https://github.com/d-zo/xMat verfügbar. Im Folgenden ist als Referenz der im Rahmen dieser Arbeit verwendete Code der xMat in komprimierter Form beigefügt, der in eine Textdatei namens `out` kopiert und mit dem Befehl `base64 -d < out | tar xJ` entpackt werden kann.


```
/Td6WFoAAATm1rRGAgAhARYAAAB0L+Wj5Sf/sMJdADwbSEhqdww733s1Bzx7p/0ZyS4H/20B92q00nxf2WWAx50+qicjH3/PvI7zxlauEr2wihPHCkQpD0WAev+rQ83Xyke+a8VXTh
LbIDkY9by+Yql5fB4WxYdgz1KZDoThrYb9CVEb7RzEkK50S2dQcAHO3XbfpaZdXNVdszU6J7aOy0hdynPxb8Sd5jQg6WeBtgGlfOLNhi0Pz0Gjcl65UpYvRbbQfWW1iM0CIrdne7PM
vdM8KlM0cGPS7SmSKiu9LhfHGKGfa0NjZJVytDkms2TzgHfBj52S06Rsj9mRkXRkCjwUA5jt4DVi8rTAx9ed/Mw9K9WgqR0p4yQbaAzVz47cn4sVRzb2PmsaVHpz9FVpslZ9EtLTVA
43cH6q/J+NlZSLAo7Qdo8/U9m6/8ujPpqUfefUk5k4S+I7Y5Gxzl1wJBYjJ7s982mgMHudpDbsYLwrZkyAG/NQKTBXcByfSsGVFxQffVRoZohqUGEwjiPb/ERK+00DGxmsRW0Jhmwy
y7bjljKj0c70stFy9ACJj9eM5+/SwqQNeVibecAwJXXQuBP5ai/mZdk9KMfXHf85mYCn6OsYoi2AsNbaEpoiCfeWYbC5ZZvM+6pwRG2hZlfp1cOVC5B5zH+9mx7kOIK/U6EZQxZK19
P4+afSTBuTi1TsqzyUBAoFcpcqqS3C9bc/ZLGI//CYdf3hyVz8ZP5M3lXabZGTlyGM0tyqsL8V9X1YhfPyilBWc17NvYJ4TNgkCaRK7As6YZ6IuFj6CIJrb0lmMzJqfsoaqjGtGavK
3QUnpQ520RcXcOtRIBh9cTIGsSMtNiT1KTSErG48pVtAC8klhI0szcuF/yW5x0TEAAf/GlayA3D6fyHUcK4/12eW3ay0iNQOSpF3ALPD2yggudDXe81ZVeasJFWx1GEULQEt2Ths6V
FUvbqsPerf9UMEfhcWRDvRFfCVLBOXK/DntgpMdkii6EANjA5Aw9BDsob/E/U+0mXHArUiIDB+Iw5xhJj8m611e3TpqnFLt6WDr2W7zdFUINizMCW1rugOU+l+Zfz6F24E3qXPqCdK
rW4SXsBuiELzwf39+vTgjrz3HYxWiFN6XLDTB9Z6nB2p0BEz9U+U3mr5E2ZEOtUweRR8IXcKgcm+sX9S2jnXo6J2sXMB5OiiyCSJopQMZgzoh+GlCTyCJIyxoCJo1TIi1bUpGg2Vxe
fYdIzpWmKM68jHiF9txsm0Gi8BaqMJ8DVk7S6NkDtjliwW7/H+WiBndHYE2S29V9mQ37NRAUmC5rhvIDuQotTcGTHUDSRFS43O7B8CgWDITecPWmnog6yex4/+WpvgU4V+tpHaPAmt
t0Xu2Qnl8FBnedF7XTFfT6a+Z0l+peGFWrz5ypfJK1id4sbtVP+O1JkvT8+6HVcfhdweDJGe6hdD2/eiJn4a1RAinx46LgD4afL5ypmZQivH7ugdJnD/bVfyCg63m1cxoFldYLC4mw
uaNxLbTzH51MGrfSqj/WvgAQkNmdDpPpF3n4GXULF9LD7n9PZvZsdy9WmAsr4g/D4QoEGelVaeA+41cbNOPRQw3rdqG2SrNe9UI7p6zDBg9jKsVFMbodpdu15TG2s29vwgQKISJYVf
1FmUfAPBhKUwT7iWDI8qP1eGtB901LPp9nuUUyw8WyCFyV+fXqtRG4pZF75g6eglAwHFMRAkSMc7rVxpkONt1dfRn4Hs5QOSzwXyTO8OM94yfNg2yWMIxcOTEL30SpKdgAbEOa8dCq
9yrJSmE1ANGUXwAKLXV6IfAZ6wzS/u8qsHA02YjmhMTpOe65zpJsJf3II7in5bDy7OBiWOEW5uU9z6+1NwbJFZPu9zq8ev4yKWNDAh+jEqrGLzWoNhwzB7DZpZJolQEc2t2zqfh0cp
YMgmmR2OQzmq9Z0+s9dl7iQLth2UxrZjmEC95V61umw6va6fLtTP1oMVXbseaP9CZBY4PiFWQtVrrf75PG6AupLpm6kRmRafwXTjgyFWu6YfstOG9f3ONnpBdzv5qNjnlVSniFjl/B
DkljBIwGn8idJUPWem1prf/Q1wZbbsb3HfJrPMRsy9SLMetK8xcgCV8raS6dF8838IldXqH/vlnbW+SvzdkZXKfMBNCHbbjM2ur0wAFREsgIIuhn/tKnJ+liVxCckGN9FlDyPNhNy
E++ivhhEiCKLo7sO+6zjM4oaWQ5wZEbJ3Z0yT51Ka+AzyLis7gDNyTf7duj9YZX1iAMr/1L7mJ/dE8ScDdycEAPm/O27WnnLlA1pu4jK8rR/RcLjQ99MBlOC+yhwHeskhtPyFTScH5
h5G457jQyLv0o5Tsm/ahM/bjs1N/FzJQ4k82wDxa5pb6IRRJWrGhmefAfNuWkb6hCyrURqp30YGIsDx9JEKD486mK4mtPzAx9bxW5MR36ZMe5SYLY2Fmt1VNcb3N1/3vMJuM+JMqsV
aw1nh76g2yrM8y7QUpp3dP7MfZUBlnjYKJp6S3VvYXoSPJ80sq/dkJgd4gH7HS99pT5XRprKaWvFkPb3hybrkg71AfZ6DNlSSaXMmuXbttnRUKYuigQLZgAgkY/wJ698bn6qfJQFMK
d32avXJvkvouU1ELax3GO8fSxS+ah14Ejc3+d33t3/e8qHohgX2PR8iNH3vFXb5LJ4ijZzuNoPHVCes5uEYl8ZQGd2dWp5MniqxwcU4Od0vmhV+lNOMNbuNzQmENcH1InMa+tF3j9C
2h67Cy4lognJWJmOy04Nz3xqqQi0Zw6KfW68mcNspFvYIZSJU9HTnu94fl07dlvjvCRfUK6Uj9dLefjweuxkMtPVjOxbfDwEwJbKXpuENHZcnjAiZ8svma+c/AiMdI0Jcw+/KeMLod
X5n9FZIMhpEL35tlwQiQOah/eNn18UZXeYchVpAhHQGSRD2gZJiTnts7QTkW6NB1OtAlhB0j8Yx+DgP+UJCyHNANBQfiOAxaEIGRyhJRLxYyrf/k/PAFgMJb34lyvwjhqYKyrb0aUV
sWcl7jeJOLQB9UBZU3ZPsEnyDLvsQ0NE73Wu7X2bYKDqmHp6nzjtWX/Rab0jMWKI7fFwFWo/Scr0o+ptEZ3goy3yyzrJoYJHl5bO/vf4bVwrSNwqR5E3m5voPy7X9VaphKb5K44AD4
K2laXyT78G3GJ9NIvPfysSTALpBEJTti+hIwLrH0hnSfMadSqJgu7KN2lIwAiZQCVFb/CYCQFGiNGf8QIF7Ib1cwvpty5V87RYicyTLcx78N7GNq/x3RTcqIpAwPsFmN2ypW6kGiX
nRZlwYy0Xjyqptb+Ll7inwxa6Sqt3GGz4DLpGWBIB47AILNye+Jx7fkQj+1EYDeHm3T8BjFndff3ZxDn6O0QBvExMC3SCX5XEgc1D16F5FffuLu9N8OLet96i1pSeXwif5U0lszFo1
surzn1NjiPJob4SAn635vpj03NW5YSyAQ0OCywQnrmE8vnDurZIGLsqQ63ewBCKdZHGy+Ue/aA6esKmTfNIdPX4OcBwThucAxD0J+o4F+YG42SPEHumg0+p8DcvrYj2J0s9XJA/Grd
fYQP8PpXh4wVb/2xO4tYCL9IXAsAG3bAqMlrkpBxu0rhCJRWh0GPxb0XBHqesF/Irlq1QshJo2jVOQ937e8Fe8bPWJKrghfNfDTbAxvAeXWYmv9jVHE85i8xtr1Uv0bR6a0BMux7rp
GNm8DNKeLBF4Qe8VmI05bSbdX7r0FEkg1Lk69mHEwpi/OxjkXu682761LlMmS5yi4xjkQi2+YI998OzTSid18fyA9FCephdrcKXl4lIuOWeL3psITSIsWkF+8T60fqajd/5ZkLGv2o
xasnE/yYEWAE4NyB+EYGfnUlg3xb0pwYOdnjQNU3ijD+o2DVqO4aV7cJzf+bkBvF5n46OsI8PzQ2J8241tFockqRIdCtATETb3Zz/wONqrzQY1p93QoI5UM7nlJn5EvdUjc89WTM5e
oqy7i4hq2tQvSw2MBKxt5Q5OoJV2Phy8PmTJ9NkegxfpN4aJiZ1hc6J+yUlNbfkguYda/12k4yVXJL8O7w51vBC+3oBcyb9eWtveD5QZvnxnz0EmsY1kCNcKsVOl7Y2vaMc7wvccdP
v2LGSnJwKPVkFxdTGFzqwycs00fOeVCb22Chxrspz4Qi2YFUKKRRsNNPKrJscKCLOXaJeKTZjkyCoaeG/+XlkSzmV/7zNNX0usWWs8CsCCiUGtRtLtw/lAiZmZ3mE/E5z4p1phADUR
pbcmxGWDSh3tXIx+siw5ttKOYHOAaav0BwUEofFY5YQqp7yjvSHKZc0qdv0CDwKVQmTgUvvSGzZaPntptrFfGnVhEaecyxAErlW+WIjZQMH+7XXvjhBqTdEkXOtqFPciH2X8w5PJ/0
6eQyl/VqfxnK5HKQqGQLvTtwQIwOO9qORHkNa/qlu/J06EtQxfKP5uvRHbBzkmefusXfRDp+YcScR0UQ3/BKg/6bCb6ZDpEVlG1IpUDb3PRm3kbW8Dv/Sug8hjt1OPGoUIZiW3iFhB
BDjPMT/IpfIEc8+7TyuZ/V5tf2Mj/jGOrXs45eu9ZH6ThpSjKw/jurqdImb2QLIc44KmR13VUY6vfBzzjweUBuuL3d5BmL4vy87QFmPoTactRtSASZYoNuyCjdWQLF43j85XxpByJQ
kq86mntiWI/Vn8Q9SDS2ZAQX0sUOo/ZpJbUaafrpvd1u3t5tAig9atzAxmoeDfWDJ6tLwYnoMaUqwgNWfFLoduvAGLV+fIkOOA/iV6b4uyjkGWed6C9tfUqMn3OgZiaIBYfw4Jw9G/
/Ag18MNfQDY6+sz9+VXenB8np2ECDIiaJnbaGs+Pw9hxpD4mW7MI6SkWEHeCWb+dsV/+b7uBOfJYqqM+FN9cp8HDIaIXr1OUNoN0y5zHO8e2xvoJLMF0Vkz0k9WRuPwTdz7zrUAxH8
pUtqGjEQAe+Wb1h1SEb/Sw1j6oYQcrlgD5cX1VDWO3FDLRD1Sqm1ft6FtLzu1rNvuf3BkKSkzLX4HwPYfwNT0qINnGpCidJMHEPNP7kATe3L8VhLop6TGolUNu8lljaSFr6LLQVcFV
I/1ZPfuoolQyml9cf0X1gfzS3pVzqj9xY6XQH9f/0W3PZ1QE10MbR8AR1+//rDEKFj3bcYfiiXtr2QCLYU5okkS4fcHglMqZ3BNgoQ2bJ1qUXSHCgxGj3XfxI7Y31yI+EH9HjIdtZL
mHzQ4M7r9iLIJsVCHDwasIQYlp/JeEW4ODlpLi1i7gvpm9h2v28g+QoVOijG5v95D0ZW6ZwIpiU5ZWSb0+YXTgocJ5oDwwcOGalkRjlINrUUHNPrgXBdmXphkjuuOiHGoNgdzOu82s
FtoG4A10wf3aUByhoHqpSx+ebW4gmcIOh4lUJkTUo+fFT2nXRfBMb8IoeqfEhWgrRfFDyCHkt3K5DO+bFdShouVbe/VL0sn88qjxI4u0Sw6F5gAcDPAJ2h3j+jRjMiUsctd8t4ax5K
5NP+4f4SI+YucEtSsUCCJmnRc9pkHLtAEXy+xAq6kA6rEGEXf1WtTtup9LHTvNRdwJIptO8LvCBD5kUQaCarmSvDyQAW6fYtBQW2wOHIoVqTfb305zxEoskYMZkskszHqDLnuf+Oaqp
8MO9owVuCFHPTzWr/0JBCLK/8OZmpdo1dP/XW00+snn3KC0AP9WKyRVqMUlqRobRarNXDhF2i491AANXogP0e0EJ2Bj4mSK3iyGH/rwahsRUukvHxdzEOjkaE4HcHIxj8c+JJr99hL
83HzjuS5ooktijPZV/LryPs2WX5ktpasIf3cbhbq6XUOqH4Lq5ck+AcwgpvFm43Q6WhmT0T94mWtayutKWyFncNI74YhMP5+tdZ2nfXWRuXaMORHHOjKk+Os8h7sxzWltOJRIqTgGA
Qkpvu/qbUzJpoXZVuo0ecUZI6RHZk2mcUeRR8dGNCxfNjXEosPWuUP5R+LYrPs+MICvSD1iqwJdG1SJ1zYb3hSO5NwcAz3h1h+0CN3eGA53Vr2olp88/zmGjrD+FzrYgoQl7YTzvQP
sn5S7sNIiJAoOKA/PdT+5un2r8PODPQIojPGVR4RF3Qn4/dnzE7S1uYld8eFOqCFn8nX+LdX/qOPulzt/eeiGSwYyOU1oVAzzbus9cVCGC2qbjysjP1qoScV3DOavw/+J/+IKGA4Q8
Cyg4qWYzEFOPUSSU/927WJN5iqur6vICADh25jyOryA4oMjOzDrXNDJgg4GpOLY9URJeTUz0ziV9XS2eEAPhYVZl4H37mL5jjTILq6WUyrxT3LLbpCP3CWoE33VzjFh6XeQuKVIyEk
3whlRZBaBgDgb1xdzzwIzAKvnKEId9pChzGmJ5tngH4N7/S51eHb3R5PlbCU/UuKkEjHvL7R89B+ZbhmwXWlKULhudcTvRxSS5rwOkE8bHm8csDtAlAtHpbIyWR+i+CzRK6bMXGs3
7UkennTFZ6+DMLHlXBtHzQtLozoUrH3sHZwJ9T3P2saJOly56yNIcAj7e3SgzTIPhKyJwFj/m3SAxzdioW7tFw89+xZIY7SBjBlGsyG1xVYvNkqYQlCzA4kmWnXgmfUFXdbjDnc7Eg
vdIiWC0t+f6S8AxhARwhOBrRtdb4j/41yGoBVtjyoUfYdyKYugh9lfv3wlKeUC+7KVdAHG64u3WQQoVovJidsw04PuZuijM4auqoSImqhcV1vgj0MAdg5/Ngzc6rvOI65CiOoiCrql
4ouRMxY/YTnp5Uc+fKU4Nx/JskLQYbSavnONQnoBxaor3j9usjiMcftTKPxk8XBZGdrCcobdlspUyIswk02l7pI+9vErOTR0oT0RIa/xKpt2QdHqU5b4bTwsNHNznFiJYMiulLS1Ua
Icw3u3PQWytmznhslj3pAiYIE4PZI8cLrI8jwMIOWq9q0hAkVH0ki5n5k6xPZhKWxrPVbgwD/qqH+85B23BgI4lWnOZLjFRBtTEIEj0ilueuO/notqNj0zBbcbStDfLVpBuR5E7iim
r77cMYK/uQPnZD36rwWe7a2PZlcxqmpEtEVBSKBNmmNVOJZIq8smzYm5qRZhj4n7gkWeis37uDtihSbHIBYjbyyVN0Ntb2r+LkchpdxZI2lUdbDV28AgHt7bZnstamd2KOWcnqIXjJ
gJy852DO9rZzf7lpYMjWxNbG6GXWUGl/9H6iAtjjrZLXRkCEKYqbZFQMPkgJxS2EYg5nxzRxpNHKsZOqQRVBQYUXQh2aR04h0Z/WwwxsJPANoQx3NbV3dmB6XfQoZnWxvKBEoeykD4
7Tpk7+4IPcXsFS/9IS/wVKxw3uakv03EWL6HYf/BZyD6uoxqZsjINnObBiE9GyADyG+9euzeOhHe8Hx25Ux7d2gdUgzcfmUeRjPYakfkCNDr9VE862JsMv99vM7Uw5ofLPrURvBQ4O
p3zeuh5XkkYiH4rmtVywUeVWtnAxU+1DZRtebWkNGXIjTxXGsB7oOioiIdj7WVkWggNYPGEQXncDvyIM+DVlMw55jp5Ssm1AH2MqGMLpue2fj+sraD6olnE29sWEH1ca8novselKLi
Opoz23yfWq64klLb8dHuKxLBSJ1Ns6bWH6VqeElWHaZSNTUFftZZ2oE2FuklbguETSvxYUyJGCdGH+N6GU6lijI7xk8blT1Mvq+DqKH9m7+DPsmJe40Ylz4+ey7lcSqpdF51+9gJH
564kE7thfsRbxInBrh5ovq6dSMOLf37fQcGK7MO5iufXObMjVxqHpLVys0RnrNV/UOpolVJYQOakEMO4a5if4Ju88qMbENM+ZVKZRQ1pNXdJHLEqakhKuRMmpZvdo3GXRgc4534HwU
/pkMDspUTxXfvFqECsX27cZ9jOXS/ZsPk4Mu10lxpYw5PK3k+G4CwNLYN0MluzYPxmEwEcXOjJc+vRNSySD+T6FHXYUG4HbMSoYvyc3mzpmOlRBiwwsNwYKyUOls4GRZBOtQr3Lwin
GQV2KkGkd8+E5TEP4E0zLyHROM8TKxDBDNGhXUAp1bl8uCO6IyYzkxg4HiECYjBO+Z0wsZRjdFz2LburIgUsA99yHWvbeFulMN3oyR70iUlwCDkXi38K8pEQehTe6IDq1WSPHtOWJS
4NqJVEEuQQzLIRP56CWaeK570/4Lzk7A4zV6mOi4Cvt47R3V8EJbFuTzohrUUsWfiehpY8euIKih6gPhtJpcSxThlpm3HOmt77QZ2PJZxHiwGSwMrr031NUZcEijwhZ3L2Omw6TX77
yUzXW4ciVkLf9mZCYEKH1g2ZMf4Mom9NAM/L8Ojx477PhBsjrTe6/LFfozI+r6dKHRBg47lAVAF1z9oJ+wXQ0QmTCSxBVSPt2ifZLZGMpCrWNfkiJWGlaavIZQAYJz8J9ZuzKgaV7f
YfcGWUPZsnxrO98rCiCgoimuqU9pPyFlY87NUWJIv2yPyBa2r7hP9WLMxV0/a2aVM5CAdpqYONf69htoK1wjL0bFuIu0b+Lm8VJLFcOBGjmn3aBb9RwcigFW2+AcMuYL7FOaUmKUMj
zDtPkRpLGGHSg5yRWa0jIZ3Ea8QR5slIl66/4e9Unsb6nu1bGUCRiMLt8V8tu2s3/FTih7slAXOTWENnXN7reZ6gygIa7oAQJ+B2aImXd/WPe6xpznbmv+cSra33pYNY75ICKaz/U8
SM5U53U6ug/B3BM5HAN4YTnFeATAaBgteP/KQUeM0pe4r3KkqWadSZM0E+BZsjC9ENvtV/MzA2vi5wN46JAah9FVSn0bdYRi0PPgPdnblxSuR20My6sdGUcCcVPmtNJ+lXTmrpcEik
c1m8o4lfnQsQF0Awy9419KgYo3acKhZbRgWJglPyFuOBpTOZPqfwQtgzd1LrArDGr1v0mHDyLhPWFI+h4v5p4jwNGsf1ygCgYLowfZZ1gLbJ6JX/Y9Z8QFRc9H8+nABBoktN5NPc4r
zbYRgWmVHS6r76vzgbevQVHd5Ytjp01fvfzX7Lomg95KV6CDMtZbCF0010ThKPv2GSp6fhJxgTbDnJZfcDVdk3PmfLhOSHZ4/2FAGGZYDxYgP1PMA0u3dADFCmuPxphTh9n72aP1uI
14uYrVNzL/lKXYw+rV+aDr5jpo6aW1tO3TptWobpZ84Ty/CPKn8jH1HEN+2VbRQQB7ucJCkSlFRqA5JsrWM2T/Ycx6Dx9umpPzyXAcFR5sqpcLJvAjR/Rv2Lx/456+V1BvfdaNJQLW
HTSwkIZDbpEpQ6GimgIEUTbJ/0M5XfCdRCVe0Lix/Jz3fx0WCXoR4ELcxdckRCMaCQlK4Yzv26Eg5QuPvI2hzyIqGsm09h/LQ2jaERqSymDQOo5xAq1WBSOOSbmh9EkPZZxOOEa4GO
```

sQTOZIyvluVmZNFRhtFYmDsAs79s00UxGZ7PFVCJ89DE6iB1JzMSOKPrIyRdAFYJbMVAU8nC2EE2pvueyr5m95RxP6HuhyTC8c2m2FHAdy1ZGLwI8s42ZTMAPOplQQllsi6Xoe+DuC
NClT7hF2GppaFRgemXQE8ZAoKeJH6vWz5JP1qpZTJ7f20ktPRdQyzG2XbEw8ADiCxM+FUvvPQeIMSGU+AG4jzygLWV11AbPLJgP7xsU5GRVyaCejVY+1gjwtHaN1WGVqbs8HB4kPj8
lmvXbzqi/9MC7ChipKBjtTMF1gTRxSur4HndpcDwvUvP8Dq8QB4mr61sM7RWePuhhHc6CWxb+6BGep0kcYl2KrEiczR3pE+FCshIbl+Wm4QeP1mHljqGExQp/VrNCjqPJLh/dmm0V1
COgDUnao7YSv/2iLXW/lMLidBpCiChNhHI/PphQVA60dG4JHbvH/c9weYxUGftmtmP0yMQFP7csZheYEz2UpFTlFUnw1as82OheXbRinsBZAFljyj5T+celg0x1KaKVQ5VlJRGTCng
bj8/5LLo3Uj1tvec73r6meeMCi1vGFPHXX+rQBmALvxoZh4DUQFd5mi+7+Tro+EPHpZS/PcmiQVteHDVFM8bEu+m0T5VgsDdU1VWQvB0H241qHRc8TI/Rzc5ojiFgLddfV9UDk65Sv
Qn7NanhPWgbQtqz1Q5kDKJ7vDSlYEzatCTReUUKJ+cP0RNWpc4/kzrVg4aQ6DXbAKj4gDp+RQ7wW7EUdZPiOZ3DJ8ot9NEp7MHsgHmyQIA8aKdbUY14FlcB5iXAcjTS5vdgqg8yuS1
dkaUq6MnF+QuaZ9te/lwj7U6VIz5GYbzDQWwM/XxxK4hogvuakWzIWRRayMHhgHBYMA5P1SyQ1/6HAB1rH6t5pex+4M9zaYAYewr+4lBwkOXR/pulR5PggJGQRpy2PPYHEVpm8YR2I
YRpklpmcC7BcS7rQMc2mTj+ImOmfNVF4cjI3ZFWe97mWs/wtyOFjhS+OYxyto+NUkr8HzK4uGW0D5jJkY/moI4NFJ86gnp5uUEnBNIZcHHPCEL5clPvxhPZVtAVmjAD7rXun3vfdKy
JHeB7hYTCRKtcV4AjIdYdF0uJ60cT/ISF3KGzp0mdtwPr7jIiAyp0OVRT3jkgXwwp2/BguD+pWj4IpVFMwZua77bwvEbI5SbkXWXV/SrgL/tJU73k+XnH4BYQRxXsPmtYXWbskprVV
7oQCxSiW0W4szknNhVk8wRkLRhwjA7HzTFqfExV3gp67O5MEwM/h7GijY7Jc3jhRM6ZADAXn6JIO4uDxQ3ZdSkbQl1rioffP52/Qt78odMbMfRp3aOhfRWzzwvrRJBC2LUj5XNP6668
HtkjL8TzUa/lA2S+BWKpycMEXKyZSeisNgsuSvUNYmeLg7cqBPp2qFcy/XmvqRR7SveAEWICa57Ho46c2MyFN3TwqIj+oPzQfsmOyvoRbV1AJnW1P6vQjJGd8OKEcsEh91HE3fr2SV
Dmu8rQWxTmkNDltHH35y6/omp7CdeMHNb6BHMoQqYe5OM1WbI5A18IloQPwhmH1n1WMz16bRiFQqNEpAPjaZxh1YLdCzFryrxbL/qdTiMJS5RKc+nMkbFnqtMRTYg6z00mt1DmSCUz
ro/T7KI+Q2/t/f/1GIGJD8M851/OaGGu7DP2j2By59wfGOPt2+gw2WV0lKPxKRJ7sCX5v94RyYgcF+SaIUr1agNVLx4EZRUwG5D0EpWqh4pZXQAsALpCh4k2cQs65CwuuAeydprzml
c6aAiikm8ZjspJPerpNN3Mv2la8wMHl7EB8p3YlCiFV6jqGKhGAdp0xfkOtpA8vQkEvmnD1OLuAApqMoVqF9Kb1w+AWfbEO1ZYcG7XGG6zgwnnfv3U022dRtEAgAKfF5ZQgiYpPYdy
w9mokTpZFSnva4kZDJsdqRHD/tD5oWtVKcv1odyRxZgO+YXmK6N4+/usRI0QXSEg1EfJ6KuxltPEwtIGW2jYHHm4Dgblut9pBObkqsdFTbVlR2WWAxHHwCH/Gi/1WwuFcdW1UMcC0V
B0N702w/9E8Uo5IX04a3FdXpWo+MVEDsV2/avMTLjC5ytZ7yKu58rgXtlTnyoHm/J+bgq8kT3K2iQQDqoF8n6wbM5cmrtfprJb6wVtLiPIRTki8vd81WZVLqbekTf3Rzb8HxTMx1ok
Coi79gqawFT+JAJUTUQ+V4hVe6hScD8STU3HA3bPUG2ggRNVQCYshFGU7lkYXwCIV1hFwO+SeE/K0EaeJ/VMZUPDRUVzFm91olUuX3H20E5v2CBLUTozL2kZLmFVZ8jCoRRkqDno/y
BPxp8XngaGtsaIve62Il9AbqT1AHpcc1FYJC9DxvrPhFDzgI+ZDAt8RdVjbXjh2SfszATk/6coYDWSB7kGBVq1MgRVeT/l3dXAfO+/IPbbG6mjpO382ANTXiL4k+kusMqp14FF/e2V
K2OnVHMrXM8tP1h9/xAfB4FJ+8tT+BiLXbXzMt2J5nCVrAxZXPi6dSoX5p1/txieFoLv31+Jb5imL7xZhi1stzN7YCBd5pVViCpyMjeUvdOJ1XiZ2Kvv7YM7Ypt7MDxhg/nPP4enmg
qC29JhDqXR2lHrAR8xr0o9Fru5jWFZSiyBYw1LHTmg4AnfSDfUIDyJA9mGh/vyq8sPOBxEen5yhLGbdh02ZQSZ/kdRoU3azO4nLFyCwgbMKpy/O/8CkTXc3ciW+9vSMBHbHBDbC5m1
wnm8uJf6jKFRZ6KI61OII/3u1MxFc+AiIBC2ELs3fzq4czCAqwpK9qzTp9UUPx1bSmE8yuyicxRoT2GnDgIS+5/F6neevK4lGeujlMHyAlkCcVcaQ+jJyGBHmfDUUHVmsYr3aD4QH
BlSmkPjQJh7HQKfHMJZjqabkGectjYYjZHIQKJnLagOSF+n1o+fqxq9R7pxHNLhtGPVcbiI47WgSP9QLrog3NEWgT7yfyZs2g/unuyVx6oVP41ZMj/+zJH+HPCl6PpizHzii08mQX3
q1FrYNBt6RX+O54Ky/GgHfZxz4oN/vPsBVXBrYKut/5Lo6Ik27eyDvJabBHwaxSyYiFqCl1El0HHUxKzSVyAI+W21kEml/BK5BFK1vJ2xzVkU5nNpakGDsvvfng/YSnFXZlim9NkK
4M2AGvtcrn2vXTEquHZfPPqNSY/Cv9ajCSu4fMVL06r6fTr2lKV89fxBCYLrMTZcJqprh93lklW1GVq6J0hjyK+QBhyCjPQ2YibEJMkDSvWXr7LhqDZkH+w+KRxapzR47tcaYaZZi
28THOS5wD0yR7+GTQoyWHZ1uN+7rnu9s8H/yvRfbHoomvcVRgLDsIdgr8g9ej8DDxf3+8GRUYqIZ+JDZNZHRRKnlpCEuifNrvcKvwyjCn2njOjT4zyCS5mQzUJO/KBC23rGJF9nxI3
SqGtyrJ4dsgrbQehy26GKMs1V5hOcPlkznWse6xFOpoo5V/CDj0CzLflRIped0IrNp98kQbwSO2cr/hyuKy0z4v2pnneNLQys5dhsEBqn0LN1woi3lrkLfvbrpoxUcZeo73HHWm4i5
R0pV555QgNsQBqYIkk7/uskOMempPtSU97tQSpMgule/Xaq+msvd3SfW2XDF1bQlBc/4yukVlUdbjXI78MBpb5xYAUzTWgNE6t/Gv5n8Qr4SmN7+0gC9AUD+Wi65yeRg8CAKj/kFRr
8PB8zDHHkak4erZ2YoeAYKLe1D4a3JK/oSNRY6D0x9ryhOU4qeRFyG+3ic53Bt/CSw5Enjf6e423N8jK/1C1BA7QhQZQTg/ZPfSNYtaUJqgKJkwfJnoDHIorialBvqJtxfd/xp/F2X
Zf6DhmUyQFwElYZzOkCQMQS/1l/MeoAIOkYh1mv7MuDxSiXxKtivChZ74+y3v8wTr71YXQigErsHcilfVPBaaH7Rl6w3m5uEw946aRNZdavn6E5E3lUejS0TPADY+eBxGg4g1Y1p/x
8Yzal+xuShbGgExFngf+2rN+uVCm2xcECzUy3yl1ggVVCJekxru+g8q/9arW1o77APg+Ny6y6dO1aUTfoI5NYu6OQSbNLFsn9cyjw9/58LZTQm3TUm62N5L/Jm4sxHvFzb7WPVKzAh
GlESbJUbs4Q1vXYUySAU+ExBc2DGCOxjSRFDph6r5SkIkLY3FcFcvcXDrmjsPXLyaN2BBo/lRN1XBh12pGzkTR/6YCrawQSlU8NRzlmsSRiT9j7rX0ADIpATYLZ1+Cd9aucL7hl9+t
Sb8at59RPHYuuEOOFtariMH5TA4GpjT7ZDAI6UaDz+OBaEJHv531bhtonC1125+jXh1+aj6vFoBWU5OgkpR8s+3asUU/1K/1Qbm53RxlZkBbluqVA5udbE9KWv5kRGgTJDr1xqUFq5
m3kQwgo++kfwX1yo8+twx+hRH/OhTdRLXZVmAveRg89rFCzKIMZ11uo5XTicCseZmkENJPnRiip/TNU3bIwHr0GmLsnUMmrNVZ7M6wyRaV2n0tD/B25BfNuRp5SABzDEWj14V96eS+
bQO1x9583NO423UZlG7+NtvL0qgrbEZTYFuQkawjAjFvH4H6xCcEfA2zOo2Rc753FOHycOv7QMr1GYVQOoecp9oW7HIm87bZEklepXz1Tdp1EAbmkVNN0GZ8ezXbcULmxWOhtNdqqH
jV4R6cgrC8b3iFMUPguYCbgZ0WmSpCpFdUNcB0HDyz8yWjQgR6uLzql6L/brq2JFPRS0On8KebThmhjTu69z/ZXnDax0rIlPOozdkxBDTOoG9fWzWuRK6/8J59OQqoUFz223kdWrSYo
s9QHbQxATHpCt2aUMR1HYp83JDE9bVrV6+lWmgqFphgSoWQFxsMRg83FAtw9fCRiNwApbIHGNi4WiEa74DmIKIt/dbulbqSSd1rDfYSKAOpa8Hx1NbeS2BsNOKB803msaQy35IacVt
VCJv8GJRX0gmx/QVb+4MWg2xub98Sx6uQiuqvMPFI59t1hCoLglcG9az/v4EMqp+aTznOTch1CKZwKqa8IPIpm0CwENLZwN2FRuZxerpSKJ8HKkTcy1mxv197N3dzKRPA7BGYzQAif
xP79hLGl+RL5FTLWvTLGa5JscZQi7a+sj7/JBW8I/KgiilVZo4xK13sSd8NUpGt40ay9dpTzVSLzd+TBIDmErO3ED9SZ7YBq5XFjf7k0qAjEXdtSlRaXQEQkQS6CLzxkTs6OiLWCeU
Dc7QatHuWZ9bzfwQD3FNFVyZudoZ3+/OrhUHlDu8NW5uITHsGb13kIighrfxhvTDo/qJ6gxO5u4nB5N/x9+Q2c3aXsyQWj8FiUxfDRcxPjHZvVE3R5MwNfCYs1uT+tHKVf3v0ZAOx1
FnheU935CcgjhhfnHSUspbHXkqHRkeixVwfEO5dAjs9dAyEyQte+HFKz1tiNpkGJmsFbXU/ET6vO7g4gwC0hzEUQ5F8OMrfPqcxN5Az0dZGZN5CC4XP/xeT2RSqH+VEcyUSPPwycE6
ZZkwnTWso1L01Rj5z151hoMCe6Qv8HNvbssSoWKikVj9OqQylCGy2UtoI90Ol/fBkl8yh1Y376xR12X/6DtGmisxPxvxsYf1obFTcQ1z4/8vTuN6PCv0ieaY36UL4ae/D3Ck7hex9d
ZKkXs34YnGHTdBfEMaFMgnU3ZjMyRlalAuCpB9pSZ+d7MeGx3UasTcO1v6vjz8cY/8XUPy0EP576bptmT7PApqnDFIAvUy9fPla8i5g+9t93xl8z1MI0gw+51e2CZYAOiZYkeV2cqg
EU1TEVWMfgPINOFeS/5GHR0znyo1V9twxN+s6FBZqFf4fw6LuScBuLjGBiSCZQKoI6LQfVjGwf6jTjVCqwZLfgNPMMuCNqHkqQJFPhlYhRlr6s9qyp4ZkIyoRN4GL1hM/1c8+ZZODT
e7YYb6cVli7RdnQxyQ6pVWoe9I3eHgu8RFET7bGl1tsx91nSBu19QvSCrS3MiV+lgIutJqkQB4cecAFv13qNgVHQF4dxqw0iyYNHPkf4xi6lV6y7JuZ2haZOdgggcmeROBh0FGUPWjp2xMX
CN+8nhkU7JSWs92OHFBKICy729OygPC+aesPsgRibqmGdvq01TrwTI3tlETeR5wQ9uPFC9toXCjQBOy4+vSMPEb+EgjE0uCWvezM+tmrEa+h8OCV55iDNOAYK3LVKcyWEcblIhZv5y
SqQc4zQ14DE2dkgYzhxHmf5Tq8RKS/DRZV/B8ifimDAA6aJuqNbivbUODVP6k2nJaBo/KO3iyjIp4sMJ16EZ1rgOV345TJ8cSjpptBweNoS8cXHteAOr5Kan3r7UD8ADq3ysTuIocu
wPz03u+wa8RuR1JuCmFkJjcZ5dxwZH7k2smJz5WBdh7aFMNY5VsT1fNxK4aIJ0CDmA8fXl0zrQNWcEcrbiVDOtwMGHjk5+CpssJVrrFaTk1Az9jjfwstK8ugX+UBix9z3aw50tumn5
aEBrOFJWxBdxwm8KH0eDuUS1jn3Ur4/nKn73FIRXKhN0AZ50f+K2Kx5L6QF5KRqOBIfM1sgkDOm8+EOW+CGjIucsRT/0xzCghV/5a3aWZIpoc2f/dUE3td/iaolN14pdd3y4oKc4ZR
Bmw8BTHJGkxXeIwsMRmdHErlq/BOB+Tj77i+LyOvjK7JOWII4rXEw07wMmIhFTxIgm9Rl8cQMti+chGqynbYXQgZdVRvgJBQ6I2qF14IjhHGiz0J7cpxAipR/VSGixxFKeaDrms416
rvNQ+6UOSNOZp1hppKFrkcTMjmZFkZDZcswjR0qZoADkL+iwjTWg1NxW0e0DDLKav3ykTfu1iLnC07uqXLNVu/rksC2nG7V83ix4R067gRpMBixJr/InLxVmtHYvBJsvmJzWqwoDWu
6EyZgTsTqoLYKBNiPJP8Bv7900TkJLuReNqFDeHNYkpEf56yys3XWm7Qll1onxgv9H+EmquCu9RsO+hTjUlz7ilUQYT6jIWP027+zxQxHGBjCERTrjhyZLqOrs3LhEXQErIC9gzFEG
oPHuRdOv0INFMEzNvogCOuG22cNW1BNXCuAw1YBgVehxpl0Sg23toymmjKO3n4/28Xe3qLHr4RdbiGIOI3KAIwTzCSoH4JPlN3MTxiv26bfWD2cSuMbcLpoLwSVjcgQ1kJVAgUbUI9
ksUvsCjSuyadgywFfYj1pYK30uMj2L2JDc5IhIpgp8jOPovLKhLDoV9t839TbsNj96YAVzBcGxJcQZzp6s4B9MaDcxGrEUrN1p+hVknMMONh6xWGg7fkwi+45ouoieFm+TjjezWCkQ
Di1BQ2sMy2JlfwTezkBhDurlm6NexVMJSY1pm0XaPZN9+LZVJYXELMRiI2CAirIP8qlHcg2F9PvAHDWld8OWWrY/qwVUaTI4KnxXV8PEoUenLUyewmyUV0uUVnoRkjFKzYSEl5rO5w
CHhySVx7Q6OCUnqTiv9IuywGPysPKeLrSiNx9lp1oUHtJDi3Td7cI2nCJrNP0DwZySRnI/CDWpNAiEImzb69z73uSRC+vt5pq8PS1KoBAJM2XphiT0AETHpkz7bMBgxCJEbI7su6nO
ki8niGD4AnWG7Ul1saxq12cQ64G29Ykkxj8ZydtVDpz6Bz7yacsb2+VRajrLV4wWrKk8iv2s91l9GWcPKGS1kMxfY19nc6EGfvUnDSoaNY9bNEZxUlCJFTnNbpckKaxcLV2gF9DpOZ
5+nYu+PfVzIRfDICpAC1sVppW6LqBqt10dKhQ12mnLn2+34pmf7mZAP27AK+bciMu2JR6Hmn0KxuFDN0DXHo6YbjXf4YGofIOC+JpnYEiSHhjUNuorC24GpmFv991Ld1hziy5Y1d/c
aDJVIb8LTXzTJ/qHkXg+DTWyjs5hwBR3W+pqf6LaO3fCZgAamsi933zpFtY65GNAuZ8RVYrmsvJK2XjtkO7p8128zd/1qCMM4XsslhBxjJBj6OX287UAYN2UI198m5hZmpwR/yLj1n
QphSLNAjQdALV8xcG0uqEHDsdi7xO3CI45ctScjLSo5tnJZllo6H6QFFQUZVOtOTl2/W98ghYYRYyt1Bn4itRxj25Uhvxvz9C51b8FS4nl2eHWhuO+F5qoUKeDL02qJI7px3sdAy/5
TAwvwOiWZUE+liMWYK9bdv5ENxAig7QRAkTpAdhhps1ZnNQji5iCBbCiaYHi/CQSepqDZmi5+mEG1GgjI5sSU7fIXUKBZOKMRHIKMXOG/z6VLOI7cumYo0T05IYZa94FdKgdGGPKLS
k91ytEzuWf+oL8b/dTH5v8jL+snWtwRf9CD6tKFW6c4YnA+yY2iAJhtThX7Bk7j+oH16rSTHV7TbJ0A1gdwQICw5RqTpN5f/hNOhm9aY6OMT+xJpb14WPXet3JiyWkR/hTDzKRVAll
aAsc3iEPaTAvVBND2CP0XRXjIvxk9rI3uRCzbQj5adhfFPa4d1waWvbMSmAsqdy9Lf/dQXtCOnGeZRq2Kr86ZQxE7Tfq3YKeQqbXlGm1bO8tXXPGwOO69IE36/uHc9dgZcoRakmnpT
H8wV8ACD2e7MPM8hThznf1vxVEpIcEdXgU2o70kDZdkxdpmaWEEh8iEwFT/e5XZf6F+6aItx50HE+RlzLQUv/dPzKA74fyMm946naNgzw53IkVWQlh+I7CRycLUcUHkrsH+ObD0x6i
y1Y/ek6qlHIQtBXxw8Y9QG1eyw2sY2tLYENFToSsrOdQ73mOzWfg8f21fKb7Gtk9j6XnJ2m1yFBBiSuhku+BY7R5OR0lTimy19m/VuH1Ye06R1EaNFh157u+f734nw0Rn5uOPrrWz
up3RDgdDRSMYOEMOnSJHLnt0Jf5vRwiFHiTi0WdWpVnlN0NOwfj+TmdLNAJOuuOy3jjqeyx5spIOELI777CmWkFtG9S/Wzew5SU9LIZY+1quT3oXbC9gLJh9GQA8tD9XgyBXBnbMbg
CqQHW4IT2gU7+Ff9B0Zmw1sa1Jb9zpg3v0pPhQ8d3lxW+AWu0PD+LQb5/C+xbOv7YqaceaiBZpXeQiabM0087DQBElp7GLVhNszSIZV11BozGyyx2kVJy+kqEDcoOBzksHpk9dYKh7
xpSfyCFNQHpLCemcwJPqnlcMFqedReKeAV+30+0hIkSofQ/V5AyCY94bmhLTBIALuMzgdsq6I8dYW8dnxLB4EwLdE/sTuWw7Oihs8iGoMbSNgQiaLZ9qmWxJsr52iReXlCsMjQ/x58
ydVA+4K53Qu5bXJ5WlXyt+aT3AGSl6hQFDx4Dyqxf3gD2A3m6KCOHJ59H+OrG1IiwrorL8UiqNvPvxsqf4GPL6c4zR0+NZW62UQS4fNl/jq5ueEmCSzwvfSWXd6DIZtfuuu4I7dpTs
VqTS2yTQ4AQAFy21vP+afdFpG/ox3gwqx4Avum165AQkDOUtbOfb2BBUV4OOCWWjIYxH8aBlYe8c1uJLDib58WjdeLJXHKqqO+TfGO95/jhzmbpBn1vCNHLMxSoTgmEO5rsylWXD54
OjXlkU2j7kOGnjc9fZr8ydcc9SUfh0bJxKoSyPhanI5jMX4d50GtaVhQiQsHqhLsC1m54uWEaBVyCcf9KLhr7av6MafOTTBD6tejFABYH2mglTLKiCFftMpSgUFsW6QF0JXvQ
IZOkHq5qrjr3FNBJWVaGapUFx9qVvNZCPiAy/JfYFheh38Z3RRpQXAMJ2f8k6fZzGSJ1UyckZ0b9xrpfEHyiXiMn2qONqEkbf/crX/HqN6UQLpOVKwri9/aKxoQS01KIntFJSzew13
W37Q1z3Fc1GLWKd8Q+bcFrDJckesudUYFwct0LXwJgsIJ8M53ZN+WpRUJJLKa9Yc2rFVxq+33EUPRNhqgQhvvoOcJztSx8zhAlKP9Ui+MJCOaE3vYDR7UUlAu2n334YQ1CECUWhKxA
MaI8MFmwmvqAQBO8o8kh6v7ut1M+Z6+MfcZsWPw3KNJiom/7wgoxKR7ZTBKt4v2Vj50GjZcNbSeO4l0RlrTcVmCWLHtUIlkN9cPmfPtEjOox/NmhxZE0bix5tCDon0TT40jFZaJEKn
f8k9qkL7k+QUh3bB4jL/znvJmiauRCfP3bW4pMIUUG2rQGc4gnFBX+rKiduEk/bp+7QSLJ6Z6XZhhzoEnBuiUxv1m3QKAVR8rFBfiaZnOU4ylMQSBwxMpOQdxmyXQ16kYbjL24Akom
xZYWlgmB5LNPq18MF3u2KkdH3jUOAOg/SKPKjyVckyaCXcPGq8XsdiV1F2Tvh+U4jF0RT2HQrO8LXsmqutV8FPlcT6Ic7hNsN+cduCOSa2K0/k9zffFNHcJSOp9KNCIVJ9nMdaYfA+
pBpBcInwJsLwm7WGxkzkZkA1i4oaKwz0jCgpX0be+IqLn6kdzoSg5FngLIwcvnuJkA0Y8E3jeV5i+3Eoa8HCArkdDW8qvOIWCaBepPSXi/wb2T+oN2Jpsa5RIrNd5S98Y1o5qlbdzRQ
TKKkpppEW0+aIxO+VJCu2tnCiRGvl95qkGJ8Q0BfwiBAUe5XxcHi0oYHyvoBzoJHXFVGOTxvxOs5niketlsV7p8scd/tx0yh0rm6WxPrf8l7Dpeld/eSwwptEaTJITfZev+cFpX/zS
uTb1GkRIiYvD9G7WZmSRBDdYp8CBgzmZANJ4Yr2dGr/LrBWHBUboc7/nIDLGKuk4pgX0gDbrF0ORe9h+PoTq9eHluseZvABgA+obqWLY6sNVcIv6I3TpR2jCtll/bpSekkiXj2Xtt
pC7bLCauccPr3nGmp8tR4lJIfrXDq1FPth1B5WOlhBygLvvb4l/T4R4H5kfrRTe3tcKfu6QimOxYHNccODdc3wtliqhJZ3df5oMbflhXLJsR+ARTNZGJ5Cmy8t3R1Br6pNSp3xSxc
3b6Ak4WWDv7rHhzaZhm1S7xyTxGKk9OhJDs/9SF0EhVR5DU3lM1/BvGsx6HBSBujo/zf+t+MPGt5RtOJCe9YDEcIXOvlA4EvNFIul8roeChiU0IuUBw8+LYV9OWZaIsAmGvI/jx7v4
V+TJFLChIG6bvtR5KmNMFlapz90KmzJ3BNoWee/ps08NbYNM+LDLFuOgqt4gQFi/tgY7MX/dVJPcHrC7VBbg27EJPE8oSZ5Fd+oaozSm5FuiMgSzfDb+rog+jrsqdWct+n4HnSE5mE
wsaSqBvPY37vWcQDIkOoVggYIqzgRIGldaOaXTgZTE24XnS6dwTGzBSQ/hWo2SnFVQ5iUwAXYJU2ye3WP1pVI+q/ztFl/rdFpRXqrpNq73EtQvU1xYC6ZpA4ZeDNY5DI3+j8gjbTJC
HpEdWMkF6m19xM1uGjyaPteaY+exdl0JhxEKNJVJmBO69VsueqfuYjbiZTkwA2lL/DkDxtn7DpEckcN2mUgb09G8d26ejWt1pDiuMCjPvgHqe0AOTXS5Da7UTlcNyrCv3c5EvYPWOo
WQPmcyJcOJWINxe1CphFoWwKqrWO5pWMOkNO43DZr4a3ZDKoeNOZv2Zyv9NC8m0Cmei9IKHN1N5ZZqeW1s+30Ur5Xp3s3l3D6JrUmq4XJj+aPf57nwrdlgnjDRsSRMk7FMdOin8gn7
BvQnr8ctQncEONcUnZ6tPa0ZMQ6gzeNIzitrUspj2t9Yg2+APhWaeNxsUcH8Fl32vLuoqubVmvsb5KTb2BSKPG9/Gw3BarzHLHep1NTVCpK/0XncDVrRn9I/qEB7GMVo66JpAeE3DR
cTCr92wxuODRijd+TKUApbJ28IqILL7mJ+BXxsF3dmjJlP7xsVZ9+xF7smclOgtn7Q5D4RWVwTHy1TTWDOC+fKl4I4gl5BkLf3dUOvx3aJjJBfSsRtVCvwr+MukyV7BdY2iOfZ833U
1bYehRqNAPfpb3r3WvFUCqhB8ES3xaeTiEdzFHv9gPeKniD3SLFuRiIOKsY5vC3/EBLnvhw3sVc6N14Tg+QaJWs09/krw2rLkRuy3TrutY2TImZt+cAjBqpo+W1y45tTz84RLCvccg
meQA32WDQoLN3XxQaPVnHLebOLZETMxLRkS7kKJ7xpONfYRQCWM0M2IHmmrvKh+ShHCod5SDGrYcAYE/uCo8UNUNmpGBW8LFgYRGYhAH8pJkqCw1lGrwkjqe7+gGoL6R5H6b3r3g/
e5W6bbaUURyjdVuzjXGHBYCfxd2eQdZXZXeIAtoXJh7F6obpzXgynZPYMvKKhEhbOYref+FAKDlyjzHJvCpL9dpBqCqh4ikB+3iADQtBOKNfbBB+Afpkd0b9vBjxlXNgTPRqNO8Gko
xW6ySbNwggN6Bm04VqrTUQbnEhUKGMmCqTELH8aXqIltBByCJX5e5PfnC74Pu7kCISMJtWO0HueMw+yIuvPJfT4Sws2dEUWAop1di8G/EP5LFBCnxGARI8vVMFykU1mweUQBGG4bav
tPt1Bi6ZUBHM2ZumquRBnsJCEG/ejdqEtFXyVvCSF0R/heVDo4FXNg5dpBUCK6Uft2pWmyEJ3iJDR51+8Bhs4izuJbbvurG2VJzcWEKgf9bFHRqe+HbXlbKlVvM8CL+EvSI4bGl/h
a2LRlysoApEWqVgjzhZJPyLnc6gMTQdzNEjeKx/fTYEgqS04Lb/WsBH1+z4CV/07CmEe2OcmSeOuPGXYH/JFGxAjuCgjMu7OJVSF7WQFmmwfpz+oRKYTSXNkumT+dp7988fjUMLwS0
WHOnQwESt3e68STCzwCUZYybdEkGDzOVNpqHBYcds1Rg6DahG75L7+MZD7u1y4o/VggAPTAryiAhagm4kNEoR3EHU6eci9IsSEaIJiuMisuhYxw8jenSBaWXIiP+SiPUU5dwKysjhG
t3//D4sDt0rnkMQoZkCOolkJVybSIIHyV14h00eSs1sK+WTC7Uar8z6sOjZ1STKUoJwDLLR4epzPLT/nv6yFjya8ynuInUrafgQT9fQGJA7A2lOlG9UjLV6Xt0A/k/aNFXwML7OlE4

EVjK7jVMyXGuAHTczqxK8GSwbt35eoNiE+fyZfY2BtjsOhT++lzkIxBwJWN+aCkQOosXKCvt9d0vYoANCC87bVkZXk4UprZ8Ho8qgWnqgGCQD5gszI6ZC9FYbCGzNKfFOIkabP2wM8
1dKVRnqHQPHR98y7/RmsatpegWb+KyMhvZHfdG/5L1izgiHe0T5UKvFbJCrjyXNOZs8wuNtxOa7o+WyrEd5i5yPKVCx91Aeogs7u6hjfpgKvtNByZT4cxGMVs/8PFcwBfOBA7Q6/RR
ahrCiPUdO+mQwGvlIFmKLu94NjsPLlhaqYlhU8rrGlwfEojYDFSpG4vDDcYHBm9I0+x3+1X8SoIqEvZWJYmBpFrrFBfT5brGGmOGyM330ry5OvR+bzs7BKAnNmZgC9nsq/LR7/T+si
VOTU8Rbej8C9eQLh1HekjWu3FNIQ4hbhJOArCavX0jewXJFK588ilp4ohwsLEMDB44BainouBC+dirf1Cw2kunfRbpmT+YP51wjPj94g8DGHI73qJ2y3d/+6mcX95NPc0d2A7tL8j4
EDiHCqz9Urt2q2rVDHRRbY8jvuIlIP5/hxdBg+jiQzsZu2HlYUns5O+oRFqDTgEU23F6O3OBqxAeWXKWJhI00obmpp7eN2zv62dYkMTZ2g11MLy0VWX3yGPvHoctptGzGupdRsqdxN
96vqDM31mdltuJxwQzO7IWusZwUENljInJuplCZBZpqfA30wpep8OdCOVMM5DnrlcsWXBzF0LZyP+LfBu8CxhvM01aH3eWHxQ1e/Ic4aCPd2MLxsD1ZCRRUGfK6+3eDBpX1VD8pYoi
miS7NG8GBytf0+Kiv7rsCFu4CSwDN/MCfWLpx1KQtTof7rIegwciIA9nC9J0hEI5vtYOZoPjsbnK36FECT6t2lll3u2TUUU9o1NXB7W/LYmASA7CSyQnX/Pdvyc/VhouGwm8NO+a52H
eN+6z4fyWPy9OA0Mx52cuw5CLCtM0sXqe81ojO71Z43d5VFlCuWfDX6LqvxzN3FXKn3hby33X42IhQmAPjUxHvqibjGOd/IIR4Tp0EZfrlb7nhg+kUpfP8BBNNq0fZxL5cpOKBChbh
MmKs53rx+I1mu1PGxQ9vLDg9cH8oLSIVKBG21oJvTQu0AMWBQsDSn2qkTRs9DfruEXW93V+Y+IQjpWI0jZ1isxj+LgHbw9yoggRyL+adr6Y4crL5IlKvBAZyRxQATckNTKwttqBpkA
SoxapUBjY3mYI2BEkjCrEtiyLM3d5Rl4WVuWKHOW4bX0n5frEJZe4WThRGEh5RAaGgoI4KjcSnz8u2lbKXMLu3lMeS1hZkwoiB/lkU0nwzDWwR0TaeV8Pu6Bv2mF+0mcX95NPc0d2A7tL8j4
f9lg4tD11j27jhzepQr1gGxIGXGCRS1f4lNeU4d+S0r/o50xZuouxWDMg93Ii8AqqzlHsnKc+xFH+9kgSoOIrrwt3iOyep6gaM3iml2NE0hd6FVtRZt78t8Hy8PuP8ndDPmVskux6+
MnEhFuZF2S1/M5GNuORdbWmJsZi6Izg4/szuDOd7IfF/NVwa9f4sENsYtjnxHy+WpFtXNj2d35OHqtBVAVR6SI5ZpHyEQfXDfmU1dbZfszJ3cBPl9c9MaZgykd0/
B2cqQte9iRPNnrSVgyEgYl4qLcfAl8yGIO0I71x+cHQUc/xhdj7bGa1VP5Qa+kx7aEa9kSG6FfTldgbNq05ofxktAKL7StFDx5noc0dqYUdvY/ijDfyLKEfwkLIDKgMoVzfhndPiUX
BYWM8ou1HASGFEsF63L9BW4MkKT92P6y7PrLEUmd1rdvLYgA4WUbZl9dIUdugg0p1Hnkrk84WITtbf4kN8VomLkFdE+kedqZurLNsSj6FWDkKDfQpbViHCKzSMtOERDDbORMKUNUR
C01CszE9jK505NLH62iujAIh46Vig4nIDeQN94h1a00A1KOOxdttMNdOQj9j2QFV0IPa/GU54SqrWz9D12GZydcdlntx/mW6BJvThLNWvFA0wkoe3mHayMcseJb6EqaXoc9ba1SPGx
cVPo+yDDOwd1gN1EdiCUdE5LZG7m4RdqNkuh/sE2vSflD6YHU2xlItfl0D1IKd2CmKIjOkbIQMN75+yDmBO4dP+PEMURGULcvrNxwPUyA37bYDRxoZQFHgAeGtU6onH939gkf/s0lZ
8KuQ1alIHyUDcnf6vg5QGo40oY7LVQoVE7WQ5qTc42g0OSc6649mQDBPAXiq/M1JXt5U5b/IzxSY6A5qMju0zrRzBY/4AP8AG29utFifE23IOB1kD/uQZ3Uc1u/MjgGP8EP/V6TZyQ
77jwQfZ4NrsP3WpjlApm1lV27ffgTvW0hWmO4kf8w0cUiEfLLM+v+mj7nvZZ+f/nxutYq2SYo4FtPH726QJzc3ha/GrBRdTmuRVqe2WncUNJ1XLD7F1JWEL3gTV3iYZyZcj9Sd9SZg
bCUjmM42wVULCH7uXNEcYwdxDkPutEUOEQNpMPSg6texEfRPe+uWsrmS2O7tp2li/tmIVdyL1dt+au2F/gZ22qoShqb3q77VUUhHGzlIZ73nCgi1TPI4RtWwp/8MfEv6zlVJtJj3ye
OvsqVG67Nm+uP4gCXIqJLeH0SLF7oaJVfrzHqbJkNGIvNbgLwWqSyvpNCNlTQ9kV+WdzFAaKFZr53icRB7wRP9GIk4LcxGSNmnk/6x5uECzkH0kyHJeD/PJhiY/ssYhJnBvo7iHbYi
Rp/ks+YWeO59FPFw50ZOb03OTJWWQ3vkyikCElqkpcwQbtvU7dCHknXrN5s2c6wLlVItn4KcQx1jWZaDAE6C4ZGFLzVSwz3PW6OUnsfyETdM2raxZItJHMv0+A0IR9S0wacxFfjQI
bVNBauz8oT6IwGoZ5s2RudMr5DiTcqSR33dHGWEyVceiutDDegPKrJMX9kJMBO+C/ZdmgVak/b5j31CWaZcjpM5adX1/iPuzyG2b4f4t2RqR2066DKNMrfyoLpGJNDxcTnRLBNelz6
bAvKJVrmW/R87dRHYPhjeNWZAvM7qPa++zauLtOizICDobKx28LwVqcW6EhxEzW9MiJZBpd2jQpKtX0KqTC3s4wP5lVRyZYTNmOUB1qLD0mQI+J4kF6TDLD5o+LzXR3N8IEUme0wXC
zxsUjM7BsZp77yvVpjplW2CPMicux+AcLs3ueRSuFnVvh3uEUxeRwxW1k7KDBqC/dIrj/OpCAEGDsbL7Q9ATHHw1vU0q00SAlGZM48qJbFRWst0RuE96pRLpC0lRYniK00h6Y62r
ncL+JMQWFyRsK/QGknE6/KMWFOnGFQnmhkf7iBKLNyCtdm9eQiMSQbL6rEydeFhY0wlUT9Yt V2fTjR6zBNAdPVoZaGLLeSKA0e62CfNHTAsmwUf99vTNBCaKsk9AkxtNk6NF j91WDo
Jyq561Ab6u1QtstqdU+HBJkEnfp71tOS8+9GUM3jrJ2rFrYmlA28qx0CyrVPy1Kg5JW2okSUx8K8GQS3Bp9MEpmnBDvq1WFCDahikokIwRkD4OZU8ajxjkwc4VchwSVIsEbazCt/5q
cJSfc/12y5LzyGBxqQacst5kVSIzSTPLhwIoBVWYtwi9GUJys94UpK0Atkbleg9d1V0QnJNGr+NpWIKGmeTcNyu8uwKwNvkoR0LTevN3CrBlr6HoxqJf64ZMhgWNybMgKxwjZ4f++y
JJzgGjyZs56/1HsYe3VQyVcB/WR+xVaJ+8gQ1I6P1uN1GFVeP+C5cy1R2/ykq/A+YICFcP0UkInoUhjrsbNUzP4z7Yu+otGczbJ9vMKW9dFKESh5CGfmD/GyKA7UWhaAAsOP2xcgib
nx8lC0Z66AV7jmxbzb8MLK0A20WOa5+q8U3gGPjiWeDFhRknrDnvauYHj42TfIiwCQH2BLYsswXUfKKsviWD48JQhI2PBe7bsmhCWPVyL+sGXIohg0z8G4LdBV0MuleeqA1ormQvEp
/WDmRSInwQXYBZpVoW0ePjdeMNC37qfKANJbrF8UYA+kxTll8K+j1sJyk8JjsTmLMJ6toqExs3BsNT6aiwcxAiP1Jv4fLW76vJTkOTfzg0K6KMM0EuLu1qpTf4PfzBAzBZZ4ORS08qQ
1mV8TELxvFE6xaSisxFYE62E6p6ffUlkOachNFvladU9VxkNI/P3zPfpesCkQK+Cro8XmFqjHlHxsL1gKo3dPTgwYkaGzO1M1YGqU+i3u6AyOqWnDzCTtGzZYeGA0NUOYg4tENr09z
GJGWiImoks+OUc9m+HnX/2ZeWgxtfT9j7MXIgDTGI+t1b4MOOEwZpBT7RbypH27Vrnt7fhAOST/6HVADt0cFagbFIk0n+OIeDEwBs5KIOdZCRRL1LDFIk19GBlyy1LEEKRCutK042l
2yjCIkXPQPgQehmvksfg5sshF0pfU/OEbzBW7jBcyIR63Tg9V45nzCusJBsVvHTAv9EtAbdXbipOvQkKyC9hkjErIjMrB+4tw6jcu9JPPCPeF1GcrAmGr7PjIil5md1FDTBduDA2OT
UseehxNxDlNiH8OeoL6w94zliv8PxiewGAAcY8x5zieDluEnmGhbUJ4p7UQO7rs2ks4CfXPEO6hxW8/bMOD88ym8mrlqiQoKfLtG0wdFFv0HnGWB4nFu7kHOWHwyQxpxqxOybtSUT4
zJZYChzU936hfdeOOF6fCIY06L/r3Qy/1UXm4eLshbdjt9JYxz9pCkgiUO3Cj+xTlQMAzRzh0ZFu3HYnsVGqLRaXMB6tT/CN3ctwVWbvrPZgZbTLgGfa8jruV/ux5lOMFflvEZ9Qkk
wWQ3O4lTNb+Xc1q5ufVd6THbc/wSDiJOLOB14tewgTJgdkojxHbNFdVsy5Tc/rBYMaczPHv6eJKCUy68BPGumvUsFQUSPFafRLOOD2cUUoSfz6ETQRjTMEOQGFwt45dcu2NuIWg50O
t64aAxtPUVYfo7gX5H5I0CUXzAwCM91hTZ7mQA0dErst9cQ6aO1iIJ9xX80oq96pXoitzb6HoDufyw70n+QloARRq02gVRuXh+We6lIrTO1gYvaku4vpMg2eHzHXUW4LRJXFzvi+Vk
vqlvLt3x6cdxUIzzTXoDBuAdPsNAl51U0o8SiCBzjbNUpb7Ol5jCiatjLood6kC59T5TY7N7TxdCHGwZOtoU7mxvlxwvpuclbSuh2OwQsqYVoOZaSeq1tVIpg0ar0RsrtSUyzZa4k3
6JPfQAO/0wLpqO9vnRcYsIRrxjJKgdDFG2QzyhKiVPgjqMFIYSm96nJJ12BNQ9UlRZBVgwUpJ+nHr+vn1 7KsVPrAI/xP0HVEA6okzKy9U9zoOk5686F8YyDQ2koYXjOU4XRB9w2W31
/8ewTq7cksn84Q8g11Sa2tF5LtFGah2mmVv9oiOKs8ItTbNP8TxpmXpEn5JI0Jrh6AfQpTKsUhr17KnNyMr0hmiKyuOgyCxCYAqm8mHYiDIj0wb4+qs/ptLmCIW/SgxM5qzk6UnJOj
Hlw07e4a679k60hZ3YitRzLzkThCFLBFx3zpIrm5AHPS1Kyn1Vhw/R5YR2UeoOqT4A3ACnRaUMUeXL7RspnWYTxJJjhqNMUkyVGUWdocmGC/81r66wXlx+qjwgQrc4jcGR59f4OkZP
IsxZqUMrnsHh03wSaFHeWjA3JkKNI+0aj7LTvXMNrre8W1qCuVORYgVCEDNgRSRPP2fPYVz3911/orJxHIx1HAKOSSdxJxrYFrQydxRrg80dZuSfeIDotvi6AkowPGGc9pcfuMaUNa
AiwYh713Y9H7S/PL4kyFxgytAL01g81R+aPW5HvmsiaDmx2dWrzxJThZVexWVUNmCAvrx40V2kW9Y8wIhx2WEC5fil+AtgcGbj0B51exn/wyun54KjnvjMT30nkuwMheukBml7Pqka
CGJxPOkIVakj8fPNNt7hcNgqs4svHnUJGoqhQNDXofk/3xkqO8Gv8TpUz/9P4t7xcjc4g3Hmd4S3gLbQGStrmbVTUrpY0L+FDni4dt+eyxrZqOt208Qg/F2iRsfdJ1Z8X2cUG5tbyg
cF4bkLNr+K1dZGqGTd8zAQQcukTfyaEt0sIpC+Mcu3GPCWuv6qpX2aL9A8gsf2T6/GP/xEcmMcDG7fSLc3xwO5nWqCk3UAfEDM6kZJQXVqBwCPa5OUr8PBVVFZgbSJOWbn2e67H8/c
T8ISlhYniyP+a3FaWt1F7t/gEiqD8l9fYH94K4PoGk5bPOGhRV5gB9rxQhYUMtkvx97xlEeL5pfR46aljISehElJxjB8uga/g+Y/hRNev27HmlkYCMbMhdnLx9UovoNTvgPvfDDlLq
pntgK8hED2kj0x7HMI3OQq9T182K8ASFWi9+uDtLN9C/EoUJBdZscyuqBphoSDf+kC/+JRasgpRcI9PPTwO/mqdVLD0t7mGIaM+QrGqcEEwcH3TbAj/Z5BrAt2JOT1snoKnBv2mK+7QSL4qY8UVhw1cXgmui5y0TyF6zRCKesf8hwc
qfXat+fZWw7ziqfN0/P/BA3x7q8EmSMcUZu5rnILSgH7hQmBXQHIDaraitD3EF+59/4YQHbt2MRabXZQ99jPG8at2lns6J6QuFZrOyKc+gqxQ1iWeYiY+cG4BlC6td8cndseDlzk/F
9aRp1X+SXg1idGKlOhQsjNEsl+NYr4wP9ld1SyBuGzjj+TQdXVxIph6UoAKkfrSBtBe9h6WSGHRN5nbIEnWVYQxyzu6K+DO1kIsdIB+RDtRTb3ATnAOxHKVvepyA21ZQJgLA2TD49u
eb2w+9HOv13L009VyWrrL6aJU5Pc7KaHRFL1tnTYDQkOPi3VkH22X/GTQYwyUlDnfBNcOSysYHhAT5+sMG/6lRSLwCN4Gj4zPUOzSTqdSReqca5h47KyCJpWSm78wLnD3p6OQLgf7F
pq43nSAECHln7vxwpGfY3L4OzFd7juibdsvjLjAQbURsE/gjEBOM7FT2am+iBIqpEl56TccMiZK157lf+Vv8iIStYcXdVNQOWC0AyDtrO8bmWia0WLx1Fb6hRFGHFCskdGsIsTQLjE
cvRAUeolf4vb6G+/LUKNATOETEf6D8T3ON1sFNSHRLJ7X6OkQIW/QYL87NHhqp34E1AMVuwqfuv1yEKaLlz0gJ9KJXTdJ5/mTUM5mJWyPXi1WR04D5T0bovopO8z30rw//IAQsm+l3
8excqGOsvl4j+HNln5JY09+94ykepPdYWVRpZgQm5yhCH8WS7NAt4A6OOmQbf0c0fOAjCsBOD7YZpTn5AiVxoeMi8y+GOLp8v4i/tPEVFX0qm0iX//Cfo4kjbRk4A38wd7/ershJab/
ewUkvNwIMQNmxMO4qt4MEi3FPVLd6fFGvu3LHjao1Rg0PmHyohQwAgH8wMZfGdTzLOuA9oTCxOEpznOvh6s/cLFwFSyKgPNU9qfP6OcfRGHXbV+vNJMGVT+PdwVuzoUHmKpMn8idLK
x0M2dj3Wp45uo+eoZt6FIm6fzkLIKUqJ70IUxW5Caw7iHAnIvGun39W0pC2hze8Xlgh4Xg4agP3R0qK9uL0Uqorr695Yr4dG4540D5yIEAkqAJ7SsLLrFd+dAKe73BLU9r+nHaAFwU
PfNFvgNc3z5RF3CYXzvFNKf6COfcrHqOxcvl9qcGXoVK0uu90UBpEgJlrEOcw/Q73Wg2TsgkkOHQfCYADIAWZpSiVD1snoKnBv2mK+7QSL4qY8UVhw1cXgmui5y0TyF6zRCKesf8hwc
17KNtt+9G82NN4g6olOKHRz/ES9k2S+FOpKOeBljj5BA5+yL6rz4fiotW+FbxoKwDM3i5ucSIs74mReocg99LFkpXCKTF5pHGvTSUfmHzc7XeMPul3LSzZCmC3aadribnB3kJ2yl7R
zmWh1CJ3+xBoPRtcpLtKBviV/q0aB7PD9MUWfGCXc8neO6wvzgvBHKi+ww7aHVeGfcAEhmjiWIZhBjc0EXJ8EJxEQj5hqi5wZYXM+R5bWXFhGl/N1bHzZ+2abGLlAwFITu6Hrofemf
xe1fZrh8yZ+1jSs1kw4lOVw72RWjVqG2BzV4hik10IXzbpsNhSuHxY4ZWZaf7qZg/w48I7YFWEsfM+iuy8JHkRbz8QeLTRMkgtgAxYm1EPfGaPfHTLXOjfhdGQ1GT68NpPSzY7wpZD
y92tThFTYbAAuDoWPBc5OZdHou7YHt5hm+6FYciqo2cmeitKTzDkE4QD9z2OGIrbupKWWZL+Ua+5NwrrVoB4tAA0a/G9M9Fd2jjPMRAfq8PLVURbge/nrq5WZDfBRdVxuJ4RsK0XuZd
I7hwHlyo/oWcOa+mLGjKhzYn8qv/WGcq5cBZdgG6q/TzoWw0ce4X4Nw3IBDEhIW+jz0k5+5iIS1ySJavm4Acwy2hpn1pg73o40OP9AXTfWK9cktLCYamjPeDickId66Ity7l1GmygN
hDhC57zIIRZW0MPdTqlsHQI/gmUQLs3OyxcQo66AnTu/zQJLO4nd5Bu4OIrag6HPj7u/2AtTbuXdvbOBOghEaJtm/rhWD0gza7ZbbnoguB5QPJ57S9UfSUui/Pfh2NkKpognLLaKpg
RVtkakZD4n/qDT6CCWBVvc8qQW6TX761m5/HpG8CdXaQBMU+BNM06E8JNmEYwX89vcjoqEkQSh/gakMeo80sgfhr9YqM9I01JRxMnZ2/uKAxFfGBp0+4OO6fux5ryH3WR3GwbzKHRU
LOwfW4Qi6j6aNrI5q0IuF0n7RZFjVvqiMWkvj+detyC3czZYZbgWasQ89ECgbON7CFLDGEiD3pPgb2kdTPn1UbnHPTWn40eDSY6THU6uI8J+vqvWWMu6Y3jpkAOJHbgnhiulwdpt2g
Si8q7M7sPQju27NjyXRzaUfZFOArwDvTfzZEcSD1fMwZ1bRWJBxgJolj79QCcLuwFQv8MnIvO+5B/Ly90KSJtTPRRYqYtVlW4Bz1/L/bCYqvHoLms6RzZyIRPJGrhNGKtIbaM0ZTUf
JdWmqnQWZnZV9I9pKSb/BqVW1b/5CL4J3Q80xmMDRZku9cq0cLH4S4jBiQT3sTkTdhWZ7Gm6i6PC0d736JJ8YgsXayy8mozobTXq/4O51JcKMFcW/F4tnz8+1Fq6fDCbM7qz6/WySa
9sTQJ4q7awRxmvPOKSWmxp6Zi7uEPP981cBm6bAxj+BslKdOFo49gEnNlwGxu/fI5YhGs9WDUPGJ4swSlgVTaeO8+XXldn/gFl+ZJ5PcDD9JFToFJmDhUjIXN8tk5ekEn11pWW/Frn
OQLytYJvVKFhQHox2cmKGS3TGUDWoNjmzajO6JWmaeYbEqNO3HCYcuXB5QjzjnuMu9p2JgTYUtAgaR0bTQIyleKXdAm2iwzWdHfMMFwqvB/TdP3UinEZE0hq8SscogsT+u6fkEl3
DxaganhRW0NNlOm+HNEFWK0O0UnhTM9zQovr/2Spda83IiQEWGfRKGfDu1Zy+qh5xQhSzXN85FxJ+5IcM4huOJrBKIJlPVLvRL4Otfz+OzcZcsJkjbcU/ugpvLwfhcyFONUzUzIJEb
bA0OWriJgU2gPP7zwsjXXfT0EO9PPIJg7Hq41rc0qNTDMCkJ/cDx6+hpIa0kuLO8tChZr8Lclu7KMgl0pAAQHFFdV+39F7g1GAVvETn9glMb6J17pBPsCazZcPfLEl+N1HnB7LAtph
h8aYSJLjUJt9lg9XqOXb/6Ga2vDolA625e43vUKPClo51Yubl6OdglRzwC/JXClHdupiNbM3U3eBAdJct8UbtgPF/bZbYfubjFT6yGUxxG2DfaUjqW2rYJnWt1MyH5kUM0LfClde+C
4domkqrBUmOGrqVyEV18/vbRHyljTWDuQ9J7wswHwwXJc8NLKrsq9v1jAQEgUTGRxcsPos1yXJ2FS2zCeYTJlAWTFXxP5aTeDDbjb2VAdFusphWwBNL4ajmDBNGccOpIN7BEX5nBty
/VlYly3Nzv2HjAdyiXYHmzPGbT1n007gnKl94fo94baKC7cNm++a8MwmsNqA3C/di1tYSyCsMZK5kfHFeUTJQiz5xoTsK+gPRrt11flnw6cJ+BUf7A8t6F1p+c/42YtEWG5gOYEvtu
gtmwuPEusg9ntan+juHaG9H8TL4x8o3i8TZbQbmKFq+DfCPKuVn0/aWFAid0/CDcPn3g1JhpyxMpqiYT9dq7X53XoUiLSMrxKvoux67rmrQMDEs8rJGZIYBQNcv0nOgkevNQZ0GGt4
///c4YbUi5959TG3lIke+koP1VYNjb3MaEorJdeJTjIggiRRCgY3qJ89XgMq8So85bByB6Vo0w7r28jFhVlS1YbPYiwNzHAO/8E9Y6fdwwwDQlqUqKCQgqL5HoplCLXbmO+/h0jaBm
LIwvnteFMcQREm269IWWrgXvuiNluD9WWLr6Ccwk4FPXfT2e+6D70KYikw1fvWOoteY6RSTpqf7+4rnsV6Hd2PCe+2KApgiSG2TaB4D9A9sfdEExqNppsNwhcy2nk2RXEh9aHeQ7m
RlDpfhMZNP0Bz0nF5sva87pFrOkc5XlzPMQbpOyv4Rcks88eetOMh/CkWR3s+ot2XMkwPUcwalOR+uycX6RPjP5rwPA7nhwGLxZst3XFE5X8MN0psXjDmqKcAjMjAB2uK7wH3RVkIH
wc/ptEolNJ78NRRYdWLwdvSWk1Blfu/7JBh5gJ2ysRpJ51sbYkUVFsOlCl5lPDtyzdhYZhtN8/TEHUpkBXDfDEo8o7nNcoNdLJvqmUG/PS738upYxl2ksZeNkM8FWPpMVaYh15+agL
2ZZ6rr6YiX+yfLbBDLXzRzmrPIHyxC0OJUaao7Xicd1qWjkIcX63P358a2Z6oyiNppUC35LIVKWMHhDonkerPBT1hwqIowYBNADDHY00YTqECR2p/Fk2B9hnk08qkyBmnTrqM1YonX
mWI1D1Kl5roJ6/PdACly6dpVSU6RuVVeYSZWle7JumyQbS1a7BBInrK4OcQ+w61UCD68RukSkE9Nry8bZKyATuf/0KA6wzKUoEcHbPPQtn/vLWCXnh7vac/Fjh05iC0If7nWpC8Zm4
wclOzRm432HZyyVpQ+dXWiMRuikfK5h/miQTAETberZ0+lAn1UtDdl+Xv+P5ump4kQv0QQEPnx8uX28vqkOg6MUGyyP4Ga0k/ehM3qyZdbKVhgmCTpHTSFNsqB2iPh8QfFHPnlMFSV
V9Tc8is+4d/G+btXWJo7LbNQOyPDCCsJz0Zw/VCSrc0y0tu71B21tncmX8W7qzVfJ7prCboyvH7Pd9WrU4K77VmoUQpUA+FY20Eek6lEhsofVuDCa0zB1S1AFYet6yWsJWiMrNUAw
ha24xp28QFeyRFFGend5pA4FLtoGFuVKk1btsnbVqYdh9mqxAxe0PBp28zso8uxKTZI9CqAHE8rQT8m3t0E4Oc8DWQlxp3sM7Hcq8+poqjSCU2iB/Mpw07mIKaznFyZb+uMgoGaJQz
bGzrVgoOFEfksXMioE7S58ur7i5qE1tCdqe1ittdc5Pk7JmeMr3meJA86wOl1VkUuwVgRSsRjqvfow9rgl1oTNY2xQbRWfx9fTFpuCyQjX/c3QpdGO8V5w/UvLBuaw/qLO2DdZTvYU
n+494NOijaTqZVdowzNhV2g0CUP0bUGzzOmFBqqWnBsIbfMVJj7Chy4np8bdi1jepJdDV1N0lZFJZwC8EGJraZUYm4IYSVow8qRfQ2d4cyzV82ovi3MwChZGQptFLFEeomZSYCwq1Q
sx6Pj0tl/TCkZmtSDnLi4Kzz6+mmZGiGeAQIBU28HdT8yXaCCIZXsoF7V4Y2f2O9eL04Lf9oLB4QAdCDWS2p2kEOt4tqkhHc/DV3haPLqEKUOPTs4hEweO7gnWFVt7C6jcgjQEeQrD
Kq832aghWxpon+0znS0Om13cyeGmcC5umuc0VX1l9nmyomYcSk61PhF3R3FDpaMuRHG+FZytLK5e5S+TTvyt1DsSaqqcwRJ4CXVyMI/XD0aRqaLAg/okSKzGyEJxo5sTQqWIS1rvh5
p+c10dklsLT7rkJm+DKRlspZ6bVuLrglj/y24aHUqZq4NmsVzLX9ziJtY30hlUJeg2rMQ+hF+UPC2YNRuZuOpNCC5JreOF5J6yRYsQroewbIbjrFEJXvrMIWM9uUfuPdCUNttAS+Ce
oW2AEkSTaxHAJFJ0upV9VbPJ1T5f2Jy9UoJoQqij8gOsnGYp3RtaCsBqhWL2rHpdeNlW0cHnVtXPAnaTjOW1TJuMjZfKzBVHyn/ZyMTC6ix2DeiKZ6GtXBN/354JwRTlkiOqr9EPnh
G2eAMDwlAE47r2athNdtKtDqC9biF2Rn6L3m8TF0JyRr/b+utUu25WhVGiQD+tuU7GCSjmlffCkUD1PNHxAEx8vG4g8s/iQjHQL/q9P3pdldFSJXR/AGhJJaQrNPA7NdHasFdIodirr
jyG6n5swaOJX3Zl6/jtE8aG9aLkvrMIT6KOkOsaFhgm8vVyxddgo9U6/pd7rjzR2mjGUEFdEFrCkzhReZpZQx0Uu7HGlhCIFMr1Ch6gtcSEjvLQZ5dS6vkXZkwZGkQUKr0OIybz5Tf
vqid0RNtkutReFDpevmkm4QvuGtpOga1ggazLsrNwdLDZcqG6MnZ0ZDkP5ydo55YTcgH+gwyDEH9n86rnPW8vOD+C+SNuwO03CBdaB0bp7iuJSOb8VU1wjUPtNXlWkcXtGTa9reIeq
3pIwtJj3zbNGz5bhYTgBpxg1JiyDEw4ycFYELrkTpLlN+ui6bYINOWE6lI5y6sGkS7ter+bVCLXbefgGfXRJyBxAGK9QLM/yMSN8VpVhi/qmbh6iuPmd9CX0zojseN+1zzRida9Rd6
n75jyIsIT5R0H7j00UbNP4MDpZ/HQNmGK71yMwbWdNNRXW5FtYV4SJFaIuLV2EY256SfeSz4pGzpVr6XEv0PkqxHJhic7J8LAx1u+IzPoaFyW2vUGspQul18UlV0JR3Wxf0cFdazym

```
glsuV48fhovl/AR+6FTL1VIGRqNwixDUlDhbOfSGgTIgsLoB4Q+o8EuJtikyz86yNgRfxCHS1/6iKrUopnNq/w+LBF5RmzvaFMTAdFTLaP7QjsTWmmHBGCznUfEdYl9PuYOluFgP+6
pe45tSPZMGoo+pmlvP0aQ22++cRTOgksL0Rbfr93v4WmlONwUgKkYAS6lVGVCJ15Tpy2OUxqaE0dXDzEmX9ZD6i+sC4yfyvdS8ZOmqs9fBW2Bb4huttUyyWz81fxnFcgq5DbOyncrr
WMucqO4mv3dTQolcBI/Vh1KNXo7sCsyyUlvdS+vmJridJwVhd1RU9O+DGBX8OyAc17f+6OvLo1YBik9wrAHUTuOGkTGU02GRVFwgOxpNzXCffSJaCJXqT89VtvJ1v8ZEfJhoERjckA
mc1qGAD3dsug/mKdIvJCj+3YRzpagjzXLoh6ZsAZZCQ91FAnUB33GO9MD02d7+Cvx75fURDSNTJnMu9gpp9BPzVpUuiKyEsXOVRsvWeyjgZmw9PMbFwTLb9Ka9oRa4sqp09s7MQFtv
4lx4kQ5r6/z1vlckHVDmj5WuVCB/L4mS7X8LzJifs5WodGDtuSxSTOrfwcU5Iy33TYxc/2n2p1v+CsbIb6X63Cumv1NM8HNaOI4zepqUvbxXfFclCWVarG4PKQpCDXbW+BMJ5+iaag
KpwKl/8bqZcNjhLFBrdsqXTi2ht8Bmvd7n++dClFjSnp651eKWmr/LBNnSdED+yHIXYer2Ad7PurMWoMJdD1UKNNI16S5wfMlPmRQStIl/q0KzIkY2Qr2eq1BGWWn0uIxSQEDM5hD1
ur24IRfaAKutejsYbDyHAjUo1EVx4deWaGVnSaxa9dYsOAD7vF9H+tR88MCkYp8jWvCc3sLehiO1dw0EspPZevzLK58M26AvvZeRfiNVuazpSUtTxYH7/NfF80X4VrDJ6rDWYPq+/L
madzU214xV5hsddCvAO31QyrZrM3fRuzd7Cnnukp1Q+UFHntqIZLi3q/BlJrvEOwPI8NaXgbWAwFGdYna1VZ9+Bq618Ruryog42wCu4Fla+joK7KrPRPQ8mbPC0yuU7PkGEOwdDZT8
V2dYzLvLENdwGjb2+ukGOub1RRS044GGxsMuuzOV0Xhb+WvibN6pT0gNpRmgsP/lDF7wE1ecD9zoknEJxptp/MmhqH9nGxpE7xJFMmqKQ0lncrm/YoRATMk9MNirqzfrAeufwBH26q
8AONxAt4uDCSfCgulI6XOVgV0qZfjUMyiKjnyCsVhUS2TiU81ATN2YT8qVS+GoZfm06ootD4mGE01O2P6czK8P5xYXt5gr8UYLRq79kD7HdqyI3MCAGpllUZdr9kO8jQWklnea2vIb
cIqHTc/vJocJ7L9qgFINhlFvRbzD5dS0i0yJ3O5/kSy6LLd78rLY0ffpfmYMAyWUV+xl4fT3MXGBK2v8fPqdT0vQQNZGrgOaix/gOQEQjhvaiRYpyUEjg6ikzxcKIEc+cUfxATgWM+
xaIc6rh8Eqq61yBIumK7MWMFbPnMZS7baOsDESLDPNufDdoLMU19XiDM7YUP68XSG4F64erJu8aVPfbqhlklgXi/kz5r2Py1QWavUrReOfo83E3vnQug/Dx3/bulCbut6ZxFnCcUE00ffUy
XseYv/Yz3oqpHM5nY+Y5KrZrWLu/b5XaDJ3qA3+clWNs5DaGv5nfZp/+6FObEMlOoD7wUAMEUfuSP9OJtlvLTsybWOr3VxohQxn61EvWAN9kGTIorUnLn4FLCHxyw8Z3kw4Kj5U2do
e94Re291Bw5VpHDV3hu8X9bYuPny+HRv806Q0KqKw+zLeVUOYMSUPaZcNmopHiiTUOLY6luhZBC2zJGEZB1RAHDc0iwss7zwczLxdXNCrmMr4UPTvPHEYeGM2Q0UZ/GBikVOzylbRq
ro4ahl3AC010/ho83nUry9aQxvQ/OLITDtkYQ+cJeK1cb+TMK7y0y4rY3yMb0WEwi+OC2T9cnqtlk9xXNzPADxPwN/fnxyb4rDRbTX/rJsEXWvrHm7D6BsAW/I7BWg5zaPqSXQ1F+I
L3N+nmgyQE9XNSBa9ccK5d7Stge5ordHTtB2foCRxidwSMfyAicMFCpimr6pBvMqsoYQgS592VyIPKY6fLAlL9cfnc9CdaV2IfEaSB+03GjhEtMlxS58m5gconlEtMglkgqdh3IX6R
PajruNL4mHDCRDjVxcCwB5pP49stdT4XamonbbTXXtyKOGSVJuZXF42EsGpPK/vGdxTtxpBm27rjykaRkA59zF8UaNKch1t4sJiy8ok30bR819njQdo0ZDeCWph16j4+7PBaJh4V2E
Mkgjse+/HN8XMqp6KO0pX1prS3o/nP99fT+Nad+2xDjE37TW2HoMJKQ41DXScKYpgv76OmxPP0jFZLlXlKP3DAXttw7VEgU8qlY1R/W1O9da7y7ulK3yUKhJMRsZwTqVEED3ZbR2eA
R5TCjwfVg214f7ijfLBTWLo8TEZxQMxxL/csnXmeGsBkDCrU31oPAVbxnfRe3Fn1QmihSqpEWFm8oFYJBlbPYNTz+ZgvudMRyI0A11G9XL+M7bz+EcyctrXiXyKfCw845NFunbMDpQ
RoBWgh9N6JW4+QHpfiWSnkRnM9aeQ4dnCFz+tGhadvMAtgOCgxevPNjDkgGRXOVHrJ7FZA1/p3UHecccwoNALMdCef3/x1SZI+l49PLwebNAWVBuS4jBRDxzZqyKK3uUHVeDdTWtFr
FQZioN4hjrPOeKFuKLcY4Q607w1nTsHd7YlpfYImUbS2iVS7+sK/vnVbRf/OQYKYXUrJUpjX/EbfcdLMlJwob1741reWwYGfw2XOiChxuoB2rFQpq7wf25nBlYFPNxi7MOA2JlDpZoo
5bH+ogoR6BETbbPXpj7FHnajoqI1BEgyTkDsJ5g6apCAl/gVrkdfC5W0SNPN9faLZ5eVnronjMCGBz1xUcaut+YPGKZ9BD8JIMr3r6jM36AyRmTMCwKdYmT4cDXV4FG5cut9SjxYTp
FzlQtdmSpMTNzmiSBnMO3rM1gOuz/zcTyGWtdIwMVCZUHSj9bn4MI0GtSUKf+tWZz2gjscqielLPipL9QwPOsEv2rHQa6sgSRyNfgyYuC0QUAt0QOxBklIk9536m08p40M+LL8KFrm
14IRjqpLtBgf5HKUYd3SG+NbxNnsuYc6LskU/u2bN4mPyhO7irFhJdKvCArGMVJZLa8QdnyiZOzZnk2Q9T9EPyUVoZ7npIduAXOuuJf5RVkoa8cKCQOhBnVpzhzxzkSsFSUsDLiOi
r+jebusDpJD+nxNQRbgdwxdNU+1AVSa9e8snE+oYnWcCk3ifkCQL6Pfwwky GfjYSew8WimfhLuprS6LeL7AhsZUtiGvsVrx8RseuiyFM5l8mOfayx6NNz2zcjiIYRn0+UXEFCLcZv3
gOIWMZ7q1JvkUHt0Gnq4GDtSEn11MLp014CoG8G9Bbwg86eQR0jVXonjfdjit4sI4ShvxV5rqWhEGlkoy83pRakEMvhBBnPcRHUb3fnoyU8jD65PtYn6IWeP75/Uk5n+RygIeYpEUe
BM5e8M9VqUZRssmk2308cIwtIITaivE0hXYxnFCVKZ6wD/ww8R30XhEiYM/XLEbnWVeU6CczzdJOaFRWirgzYrBBHlH8gCcntrkOvbFW8Vy1SF2UujZGPijY35AecsVKkU8iqiebg0
n84R3uDBFLsMddi8pC5+PFvZFim36A7YBrhrwl+3vN5xYdnSkR5cvh/yQuLBS4LZaqZ+1uyVfawyaeVP8X/Dw7y/IKzI7SNzjUHtgdKCaxDhdx6MSUEbahSApxzyDqkr9qxRganFa/
BIlClZUhrciEic+TrakdMmFTguSBAD9MtuR4I3YAMs0gC2mhPwv58gszsAl4ARwKqbt7HlEmRK9LwAKjhgmb+GGjJW8VIYwx5jNHiXgo9M5jCRV8NDBGVWwEXui4RpdycjRRGpWyfY
peuJJ/V7ZoNjPkndVSBTQ6V2DXvXK4xaV2bgT3U9T/MemOmW6lOMVBOGF8EaR/bgRzC+9uCQVbtPi1k5DPrFjbeXcp7XhRacWFdFUQQQwPfup3++0RylYM3KaCXZ6DMaE073NeBm6O
1Rqp8zCw5r1iqbPK7Am9xQr/S0DJiUtSAmhl1sQAniWX+wZxT62alfbyzfaamunIoB8zm+HSlLsxxxomnK0XaGy+kgnqsCpbn4XksvZMiVk1bosXWJL+W0hj4tFP5uyS6fhOEw/5lb
htbNLNEwEowF7luXi6sN/5KxNlHsK2Fhostos5Uzq6v0c03hSNc/QHcW6UM8sT1I9p65t9xuSrCR3MuZ3zLBWPVtb6C6HUh0O6amzuqZbe384y7hS79LZAWU3YHKn20A/w5HikKIsP
BIlClZUhrciEic+TrakdMmFTguSBAD9MtuR4I3YAMs0gC2mhPwv58gszsAl4ARwKqbt7HlEmRK9LwAKjhgmb+GGjJW8VIYwx5jNHiXgo9M5jCRV8NDBGVWwEXui4RpdycjRRGpWyfY
peuJJ/V7ZoNjPkndVSBTQ6V2DXvXK4xaV2bgT3U9T/MemOmW6lOMVBOGF8EaR/bgRzC+9uCQVbtPi1k5DPrFjbeXcp7XhRacWFdFUQQQwPfup3++0RylYM3KaCXZ6DMaE073NeBm6O
1Rqp8zCw5r1iqbPK7Am9xQr/S0DJiUtSAmhl1sQAniWX+wZxT62alfbyzfaamunIoB8zm+HSlLsxxxomnK0XaGy+kgnqsCpbn4XksvZMiVk1bosXWJL+W0hj4tFP5uyS6fhOEw/5lb
htbNLNEwEowF7luXi6sN/5KxNlHsK2Fhostos5Uzq6v0c03hSNc/QHcW6UM8sT1I9p65t9xuSrCR3MuZ3zLBWPVtb6C6HUh0O6amzuqZbe384y7hS79LZAWU3YHKn20A/w5HikKIsP
uRunMZraWveTTAVVUjfzxg3sMmoIHKrKx99Ifdv1Qlb1qLZDzqX4eV+ikCurybGf1mXs6BDQSZQvAMMwR6LsOW0jZQ8pQ7Z5oyyFYICSy7XKs0MKlci++vcMyYwPQmTwnakCZXaTxG
JZaIe90MDmd5U0853euP6a37dm3TMVkFjBxmrH8SczWMhV0qnUhf00ckBat7MSrmgmAC3M7loeJfkXxR11Pf+Il/SPYoEPZ2Z0JiX0BGMYLd1rWAIx+9I/DH+f8r7LMXUqHHvFMLx9
/q9tuW+kQllXmpuLNNfJFc/u+albkXf6KWL8OFsIiySPo1PQanRabmUfWyDuYk+fX0JnZoKgrVqUqEqQpNFfMs/pBncC4v54YDPVoW4Y+cnr/RS10VBkIuk1ciPmMfjDXJfs+Ihuwg
XCUfGe+0vhmmFWRueKywtxtE30KJQ7x5XYGmKlvvWaRoE+Gux4LCJbJqFu990ghBunRpk4+lFn+UujI95pReAbhlzZf4F8Gt7Cg2s5mfwauCdAem375JjoLxBkl3MQKLPdBYKO+TeF
PERRjP5trzMD2X1YcZ1s9ZT1NFtlprwFxrI7FU71i6gbU0QW0yCGzsG1JqBZ4ve125/Ny4UDtLYifCIuQb1hU+8ZnWDQRvNCXbHGMKlSX0urax5vdkJCOKwUaz8VAjtkgzx2/lg8+f
L8Tw4E/qD0OduW0511CwPn02qVkUzo+uiwL5b/eAf8fzL0Xj4Tvy798CKY+V8i7QhbSeyARLf1f2rZODxp+NTQA61zER1aPRYcv8b7tnT/DlpSP6smT6iMaErRaPY+rYw2FrTpZ+0DmJtJm54JRHUyx
hI+oP6FtLz/aZC9Stjngn5MtZm+bB5yL33qO4xPSYiz2zBuaGVrHnK75wPi0qITy8yMtyC/WqPAQAQxgchHoK/wiCkR4wOZnP/AID4QHWwDLdfR0bLD7mpL0uZeRwylS8BMCe/P/qd
WkGEFk1ieDwtQzxJH2tcmAjmdkV7I5RH0xxloKxgVO76Rh6ayCYW+OzKdbrM8Mo4d74tL6B+oe8GoiuGexbj181AQUQ93Op5m33G+nLhOUSYp3QhrVfy2fiUDIApIQCr8dHMeOBWxr
a6ju7ryzK6XYOy1FifjMxViNgf/wk+T7b6wjcdJ7TnE1tuXMWwC7WjM607GFd8NoA+oTTdXZqF4o6Iak6gKzymAF/feTeHwVpMe55H6MiaErRaPY+rYw2FrTpZ+0DmJtJm54JRHUyx
xvxHQ4Jr8WCtiMVXaBZaRJ6rqVWQ+wY+pRclRYuIk6GDwG9qC4/mzWJMRaStz4qMjMyvPxm9AB/UfQH06KdeqSseTb5P3x+md9VKXJPgycuzH4BqmRtI6O3rEc+rrXjWODLPVHLb8M
KlnvVXooR3ItO2JHybd2Cwmg7FIZBUQeObSmIJlXqjxi45nSYn3TETqrFfToDBbQQrzyDvJ7phbD5EdhMZ94QJ1GCRcCOy+rWUQCzJzdbfN2kmHyHTuDMX1oHJPAv1RZLdHcezfsuB
Cu54yJzR/PAUBu0hN/hM5x9hwqA/hBVnp85YdnxJYFBeWucWBG4jgLNkG0/HNRmHulWezVUesqTYa3w0vWvbr6PXNxKf3Izwbof9mgO/advGsr2Vwtmd+G8jMhcL0yCCHXbIz3dXgK
ma9ZUjMYl+z6VRRgkc+2j9et4MC3EdSlcmJ4wrGhdeuAii+8B0k6dRa2oSCl5xWzHqM+4kM6R1OqoQeTf8XbOWL6Sn6eW0vikTOHvusnjRLabyQD4hluZPCJHP59U15i15t632UPQt
6vutAfZinz73eE9QPrdJLUSdMqG+ag3gXSleCiC57VnqEHe7Pl+SC5aOZ7Oixk7GTdQLm74iCDXvfYBgZvIruZSfcA6b3hEz0uie4AfrpEikV0Xdg/Z6FGpt9cNhKVbbrGe2+rrUnJ
gaaAhcKE170KhiTejbN7kPMfnATjUJuOksJ6agFxJda9RXAmXxKPXwwtzA4WlJjmhTWuMW0TkGmABBlLigi9fcUwYhpmlfI8axRYcv8b7tnT/DlpSP6smT6iMaErRaPY+rYw2FrTpZ+0DmJtJm54JRHUyx
gpK1rFtEOAJ/yB/yQowCfwx0eCPJ4jQTni/vIotsOGATCH0+ERskFVtO9Havs5hRVw/TDmJUhkj8WbZ4561Q8soGtT7Sq+tf9CpL723InMLix7TqFdoEo8EdSgCIGOP7fttysdUsTB
OSg0/Xy8BBTRZL03XogtljJTXkPf5LF7/R0yo6ffXxgkDg1VYSOAo+mivRIdrz4AKKqeMGBB6dMXOH2VhHtrFzF9DqxTGycM2jIpBdJSic2oNddHmxnKrbluQrrbuybVEKw52SYLzW
09syRGSFSVle7xehqXWW1O0nkyGkAz8WDJkFHjBobmv/vPuty5qmLE0CYrizlz6rhUVf/2Rlrb4akaUiDiH/TXXcVKsDXBd3pyRVowfyPcU+wQ66UvkclpRh/MYttKKxnkf+6viK6
FWjvSn0O1rXVLdmnoqFSX/u5PwX9Yu6eMTJoLFNNQtt/INGuBJbRXjugtc7N0qoCZlkQH8ktreHjpDL+ZptxTdhseA3Fm1C9X3KN9jOBfF7GdHxgUdA4m2z/xZ6WdQXs6+THMjSww1
F9C+4aUzrcGQrOnospTOAJ0X1BOfBJuq8uuZy0mbzSawNvs2+xGk/5xDpdvJYvtC0jeCrIEAu57Coy+84KQmKxMudSODacRVmJQa7htgA93HCrV3kKKtOCajEiEDPCTdQUTo6T3jKl
qRK7S3YOfoi44YPyIH2Hblmq92Az1MS/aXJ/caorYwayCgnvW6EKivG4p4ihDyUdf82R1ZopAnkzaiOG3wB2y33PADhtfTSdMXKa8xCIWwosRW5sPwe1370M113OnbCCUphxhnEtia
uqSkHXTohczErDv0869WgCtpvkhq9HFyDzmKedWARkAQX1ZlFSrLaM90bRIVgySWcRj2LJfItaBOBv/Bo6b4ewhOtcdqYYPlGn7LRz/BqVYz2KX6ufhCU3eDs34Dklni/asS2PMKzM
XSF6gjBneTvch1tZxspv48J+71Iys5WsSiTj173kFZgkUoyzK1RSfSvJr6f4gohD+xuST0gWA/hfqnai7js6tRgTW/KXSQe9rJJzR5LElVd2A/e4d/JnYXjz+RNN4O/dq/2Y5UfyCA
jgDhT0g8EtWpbu3PXV+8J8h9lWTNbSn8EJ6QdLfiomPK0z8/m+WT8VcRBvkscdNz7nJDJT+6eafYYh02joBgQOS6IWANgtoY/EQryPnFY6/1TAke/PUEely+FmEvxupsumX0xfEFhpMY
7CdOhagTRFXp7eVu4zrIsvPgbjRTINjOICkzPLpwbeBgQt0wyMOt4+V2lSJ6y7SnSxw4E3A88GBwGfmvfPb2A1w107a5rPlNclM6iscdpyh9zEwV0/3rFu5oR+q+OXHgVVMzS0dexa
M6KR4Ihs1N14MehPO5sTzG++FT/f7x343dPpUjdsM81PU5kJL5O5UgMKi+MzLn0SxnlvFufkc+yXj/wT7BIVaz7JxI9ZCAxZGytubrzVNKyKsk4lE4iwd1UPo6Dc9fBqZM6hk85K+K
U6UykA+K2tXY6I/iZK+ZqAEGh67NZdla4JxySbEb+MUY/qiFwkm/+YYZzycEJO2QmEiYbu8LcIXQ0QsrOcA8j9w4ip8M8TJ/8I2k7DbV8xB+NA7w2ka7pDx3gK+L6VNuIipRcx8jQE
bTuu8PXllLwY/3FUwchs6MMDzWCUYgTDOnZrynCLWqFDkwDXEj3M0xEyWCQP0d00u0m1eKYCPdSNG9H9gPeiHq3P1P2/O8y+tSUdYIv/H+FdGAuj88ToCRHvBJZDfeGcoFKfbr4Fbz8
nodnEmokkLNAbBeHRIm6uZOMotjkcCNO0Q5XxNmUr/EKW5TtFI+e6QoXITzugOibugHo5pwVbR/lBtNwoMUwlBy48NW1s+R4G/VNgQ3IPSASyRuPDWB3xOgfTgNRFTGoMME7hAx9B6
LmbMjbWvRBuWcsWu9uTb63zUnx0NsbuybXhCj1Hf9oeoPKhzk8WSoEJDCAZChngiHuqv1CQomenFDNW2M2m+AJlNGENiVjKVoCMYRNx8If6GvVENtswA6qwCI0LnBVRUvCiDGihhR6
7Kczi33Oo2Ch3gjh4AaPc9OGFdojra92FeZh9oY1+edN7tn5P7y9a51uy+zC+AGxBbHpOIt3eBiT5RV771Fui5dFbZDgI+CXq7Mxn8K/qIU1SQv6Rejnm79nqVZ1NSC7jMMGeaTouc
kCgAVd+8GRufw7A76omtloJW55T2gGPUOdx15/M4iexttmWntPm7c/wcOF1t5Fy2nNaqn5KZPjpqtB62AtHqIL6pR8S+mC1p+k4IjyS2167kVucsm0Bmw9tFyDYZhw/5k8X0MLKRZ
bYnm9lUrNLKuxGE/L6yHfeKeVryayEyBe/Nk24oG1LEELWxRIGCLRoJ9WXT7DhmUv67j73PowFXOPYzXTHn2CYe76m1l/64KzxnGYcEhbce+5Mu7JPZkUtNzJKhR3n2mA9hdiheqO/E
jCiCEpgX2zoraI3FIFIT3tsCU3H5t2NwfVj5cydFDvqdyOnwF+PaQSO9NKiOWK6qNE5crR/5RdpyiuN0TXvu4O4dqtnUjedvaomegHG6JHJAa3iwc7xHnZl2IYTo7ATnLwlG7oHGOv
HSnb7WSstPBEf0HOB/HIPJqtdC4S70OpvYHmnhSvZpzV/4uTzNhNn0AKE7ZpYPOy9VLh5U8aD3P6UIXbgjrPRqr1NYT4nNhTFlI7hwF+oiVAoKZnk1ACGBfbfPwziQfinuh3L77xnK
3sNZ84RSjLunT6BLdmOv0aChG5UyYyMDcMpCCV8uIFwR47XshmClqYbsRyoJ1T0qB2Yr6Qm+XzF4KP32B0NQdbcZhPp1BQlYYV1XW6bzPqNTMqbgzR+mv/eEycbvl0NheS8NqiZTF
3KBBiPyN25eeIsKXOvouvE+oBl4g3/GOB6k9Rs3eDEmcY2gvhOHAXY3mOae4+VE7NCuFJOfOAJfPis8GKedZLQz7eYGQ4qE3zCxQQDwTC2m3kA8NiC7F7LO7/EDmpm95vsNUOBUoe4
+ujYVOpOtipO6bbA3CJHHJUjh7JYsDuX7E0nUQeHbgV3jMV8eR8BnlHh32jPnzZkJYSziBR1s6rG5Si/Aw3Iwxm6+u65y+xB0uFzzbM4zwhzK2Tnq1xoAD9WbZy2GSh+CeVJWT3Qna
YAVquf74+W2NagmpSJKQ0xODLZWfF+RefwjddZJQtvIjWi2KcZLc3/4NWOHxzm6AXBYWo8TIB8rSOFpvMODCWzViL/l79cwOQABxC7BeB3I+rQDr1U2FStuJUdqPrzzzJXQOA7Lx/A
j2vh3UZ/ikvbDDvMhsRZ/QTzhVRqsLzhDYEjq2p2WyLQ5SB0UMedMLm8Lgy5/hpMG2bf2RqvbGlibcYRPYwHkpX8wCzfdXsoAdnhSqft/n3t7nH04xywAytzs4eJrCZi/WtuIpFVAJ
9Ko7IEcHzyoUTMF9MzhSnxbErT0fVf2pXwu1gnW/JINSvIGNucL8RHvyeKQfo7n8PtaVAuKf7oDzOlPx13Hp3mjI6onAYHfxR1gRxf88A3WdUStKN6j6HHEJ83QleHZ6ZOBODx8w8Kk
i76m9Y1IU94ZxV7Gv6jYrPuOxNovHMIsOLUARsOp0foKkj5tqjxSiiBv7xCMdqFKlKBU6Bte1XkPOL6OjWr21M8NqmrCHiw6r5LzmgnIb9Ejh663vzpdG9UQbzzqUNPpvf+AtfSit/
W1NGeg0O0/LN4ZjY6tfqpiPR3tv91aQsvKDcawMWuMdQcsmTamkoWJ3YCDEf+6WHzB9yiIGAvEhLeJxvvrEsHc6Cqrmy3O93BoX0/2g+eHV1inkCBCeNKxhU6iETd5zfwrYbLM2qtQ
7RZ/VgozwcAHS2f4EC1B15UNJpF9ZP8gsZEv6OhhLj5AHbmgvgEGE+Rc4T0MJOrUy7lBBRJ1NWuB4WDTYxS020zy7WOXEF35qwb7Wj4XSAANpx7k3htduy1ZACZAt3jRRXWImsGp5BJ
5kiE18ggExMAZETaxJURATl4TVxqcEyhS7vcgs4mMNR+FiVGexdyvJudwEFUB0s/6Nfj73GAK8ao3szUbm/qItF5orbjk1n5MUM3C1KKKQHY2/Wp013Nqdgtr/RO7mWfQQD26AO3eb
L51LMnxt+OIw4RykMhh0a9CUJgRpnWfM9kAeHXOhsqWR8utVscjA+RW149ZzX5kQIWeUPfaNwHbE/PXVx/eW4tOfo67Kp0Rnq3s8DpSRXJT88rOKtuB8uHiXfly1feHByCu/vbqbIH
MAw0ZzafClvIM2o/Ty7Mb9rzKqQ3eORhA/eFlIQhMwqnQoRNrHq8FQU+DGljzkNj9WwPICv+CzygLwNHmv7qL5poQZusprE/SMa7mCkXN2wjyw3GjV5b310g9/dmF+X7dD/WrpTtwJ
U6sKONrJFhZ8PGLmeGSXpd/aJqDWx4ftcRGSYeDsij57K7SaIMC4mXRt91WQvV+Qag+LryVNr6lp0VE+PAjKHVH3FXl8CdfJ8Wcxh3h0uemuVtO8E54WI/6uxpZz0/+gnLldZ9JfHl
/kQ2MIFPD6tjH4RPTsANO3W/4hUhJQ6nEvLCtpShbD+4U4nc1yoq/X6BUewELOGryMKqXG9RHyFAMJRU4MOVLurZH4f9YOQfD611dULAXy/Edhrb1QTwh2m9dUk6XX8F+HimasvU2p
MYo2A2Oa/Gkj1FkhRK1D2qku4LA9jNDtTZu1/VvLxPc8JH/PW7OP9789gYyQxpBjR5N4LI9MCQwF5W4h/4Ra/YuuBZUX+1wrVxZEDUQ7F0GUJ5MDmTvlr3Lxk6IwX913bWdOExWNQ
VMNwuMX8NrqcEv0mOiuAYNd4AdFbfQSjT0Uaphpvr9gUEBP/41cXzc03NMBlTUWgEJiqT61gPjb/ymgdYjE7S3W9OfTOZprmaLgS9qv+VXRn3R8MMdCs4c5QVJpql0J9jKZgdK0921
UGkOsaYfF4/x4tN+d+wt1nBDf7j/RXJofFJ5uWd1993A3vmHuzl/wQYdFWmcJaasbayNiAX63OXmkzouVUpU6WkhEs2T27Q7dD/SOiB4/YEjv4iX3uTme1Ea08JEXlwVOsgPpiIM67
NxnIFoM+PlYphYq4lY+wr9V2Xeo6WGmpgRWVmcun1ouZkxo+ETmw1vtGVrHCu0GEvUGJxh/IdC1Y07jXbnow+eLvKhqS+hNegkh05gOcrDkqfY9Lv5RkAPQNcUh5Dlpj0+WDewBAGA
KO7KflknNL32ioGC17X6H6dbYeEIqrbOXEfBkYrQJGHESwfZAH+VSOBo3n0wOhFzWfuuiTatscHQXRD8tt/ukbq9Q9UZBsikSCstSii08oNU4XUehaQfc0ORBhh58tDXo4+wM995IE
9exSHRPfbWR/ZC8D+rivdJwEnjV1RDZ+ofKF/yG7VZDr+jnRX5x7DM4F/p8SPjEnt8pFfE2MmsH0f8P08ZoT8Ag4zfC8BzfzYqmsOgxIhy2QniBoOKygcsrR8bcbvRWarignoAHf6v
DURS+3Rpzlvp9o3V2UHmZYdjJq9jIyYSXq+wHMiDjdz3p0xzQz4CpHgPHRndJOHb/+Mj/07czbpf5ctKX/ee6v7bugZGB9TdfLomezThNwoIocLDdjhz7+t0/+uKa31Z7K5cnPhI8P
```

SkQmy6Px8/f3/onXee1QI0wcjcxMZAOLYUQXqq7+BVipztXPYaWenJ+RE1IUGY3K0Pz6fGC8S8qflGetGttrmDPCP+ZPvPJmS+Ek3QpqWgk1jlSDvt8L+g9wdqVvv/TS2T3oVPz0CT
q6F4oy0x6J56zGW0ZqJ47oHWVK0NKRG6Mrd7dmbyUDSTKqzTKdmcukMVw3ruNJOAq3/XoHHCc+rNCq8dNNIiGr5+hodoPlL/lxM3nHk5xaNDAcshMip5UiKOAQbD9kh+3lLdQXTmUB
d8xq9QE7H07JCZ0dpJO4tnocCrV1SV3xVGjfnOuLd9lpBPbI2FZurHGhNOtGHYvP4R9c9cYBA3DqiZ7uyHRir8G5RbqVnYdlGMQ2f02qcsuS9n8TdH3pndV7ow/aoBvCAocgAZW3hD
Ealsgn/TPDeSp7GfW0wJtO2w2EX3b8SXFleTSCmfbuve9xbDPWBvQQqL/IW7YlFnsPgxpGerlaziVrAL0nr3LhHF96eCRnSk1n7s2OOOJiN4bluYLWMDpAzQL8p3B+RezytQa2O6LH
HaJjyqxGjcvMG7Ba2DuHpb7Xcg4rB7qEM14+DZMFONcAnluEjEP9pLlO0mBJldfYk1VTlDDJX3s3lhzC3eTDVmmFPMB4XAbtzeGY36mpds2FFx/B50R/0PxO6FZHIOdCsZ4Z2fLPRc
5csGRomSjtJMMYc7ELAepzAHPMZBJ/k0uZFjmaum1jl4fPi3ylk0ogvk3feNqMmYzGShWz5doSYN8/CzWAumTarpzbT4iZvt8xSM22HIPW/fWVyVscXuKRq6JNBjA/2FYJR+QcRRRi
utAxlfL63vdW1FOOPWTDD16EFDcixB6uqz4iugvGp+wWASfTzJInFrc5xAMW/T7Z8nnFbftyDeRq1jBUcc3eS6wHdaWiRmyN601hGVOy55R/SmToJ86xvQWTZnqGA/9VrxBox7WySZ
9xaKG/+/hjqj6AYrjXZWOYicYQ9xHyQQt/SfnamobLoTge1jzz3S/GICSo7IgCZLqWAzJXZw9vj+OmrgG4meNhfECgtORbuj6bGTeDjNCwsJSwIqcN8/VDskeVzK2UO3YLyaXCgxmX
IAxFLw6yBAZ5AGQugzxeAByoIvyItDvo1VQ1gGIj+eLLkp1D/nEq5GI2ck5A6pBiFgYwaCKFiyrZPghfBQasNu1ooQI76DiviX87WYZuM1+ddPBsfDGqxOFKKs93alPN+VwXhkbDXj
9RJLUE6wxnPwzqbCHTo5rV2Tg1sTkZclyaQUXYcI8kXkVeUKoEfiiiR1H6UiOCKXbmwTwekM0+nKYak29xk7m3mhjhldKg2SCgMXFohcjwIMLrMfAUoX/k0my4qGapmu21fdOs7ax+
RnaukyGqa6v0MyGFObrcJlFGjcjY8LHGuX02V9v76ElnXUf5/rneOCjVxtCWCpRHQfHq34eBa6745DRlZ8fjiBhyeyIA3wXNh25XbiYfiCpALNZZ1Xr6Tzmja9ww7DXH2NbGEyjFcc
RzhtbyCOgKL0ZdANlZ6DVMOORN7TJKZ37BgN6jZGWRBsxJR5jHgEtHnTgUccw4k3jdT5kmGSkdAt6AUpsu+y19MxgH28LPkjQGpdW4/lwtliILEW9B+ywaj5/oUdRCvBiCGyCwDNE/
asjcO9ONb9D2pbeY3ONS3eRYTr9KMwdW8q2/wALtXYuOfCaPmpMIMhFoIoN0wnRofMu/YyOgEPG5h18qvE8aC9rHAC1m7mpmaff4akmfv+ZcrjkJ56ULfim7MOr57zkaIKhg50oz4
PHxWrHx7gH7qWCFkybFHqgQUXpOD6JYd+WnWlWIbsQ17FGy91seWJOfbZpFgCFyRKDq/F8couniankuQ1lj51fntrH42yT/6nsLZHh33QDgXKTfc/GKVmCCHjuZJqs2uSMHujZP97u
mIYLjZuGPNWKw7h1uF4yornTlDHM2TIc39NY2ubvgQl0uae+B0xReMHA+fiCIOrY5hFlMQnr7D1+wa+X2cg7ttKYj1p3/Vh1FbM4iRmHEEvp/4PKjB+hfYV61ZVAzUONNxkB64KrZs
FGkWLFojc2z0+oTsU8AyqO7RDZ2JrLLDTZaxVxk+1vqAGs0GiyAiA8jwN8PGuxx780wTcqSIREUaHBXn+6brO3l2c1yY0UGkZgjNtbIWM4BqArT42uB8j4lWhhciKj62D/BI3V/QtO
mvYiCzyYuC2caWU0yxCZ4sIg3qCH9VXECqFHzQloPUuGQuMWFKqdVFGPIydlhq001qeePc11+xQjoyb0vOFLx8G21JV5xd8YXySVX7r+uj5HQvCEmHXnSvEn7DKPFJQgxiSO7daFsK
RcXFgAhDK7PLIvTEnagbZobX4aHLG2JNTehwWOsSob19OcLowQrJg+QSO2OqCZjngvCWDEooClt8U8KqIkeFPEsHhhpT4/ToHKexNhXWAej4Dn1McAkKpOalGYER0aC0deK4KsQwzs
3hA7ztnlD5Y2jLiVkK0JxXlChxJsgcovx3L0NEGzCSD8tYteMnKZCr5+OohP1OAcRxCRtvNLqA+mYFb7zD1DvK3TuqBjcPjaQqwU2JCuBICnt1QnaqbDasU+hP3hQbUn9TCMB2P4it
OshOyfFw1v8ulVYxoAyMyQiPYLqETxUaSMMhmdT0HVxkjr7bFWoHY6t8LnMNfxy72hbJIuJadshjTX/DK7kctJ7hOk+mfk5sytNgm/HCq72CZEAml6nfEBa0XR+lUxV59ClQVYec1T
4mo/70rCa4tok4aPRo3l9Q8qHFIe3lKhsk4VLK9OMuZ3pRwM4h6hzcebIt+qOAJgCgdPuOH/K05SFiZTzB/IKn+ZcCviw8Aa6LNUPjphD+LA9dJy+OM96Qy0q3TwCLaAUtwwgrUIUn
e01HIMmplU0v5N14FLMi32mWg6YYHZ0rrbKG44v8wqSdNpZ5xoDdVxLAcRTYSP3V/alBNuNvIbSllR3TEifqEWfRtqNIr2M5TmW5mUj3aRNnO7NvruPJtiGJ0eTE3kop/YVTGFwNaD
mg4p6op5FRmNSjGYrcjgYBiW+lbu6gfR1abHLuFIvL4TWr00qUDpRlzBerGjpa27tYxGibZgGhG1YRywZNKah4Hu4o/ScuTsumr22hsn/TMoaYUCyErWuzhQTz0vcv6ImC+kPA6gDG
q7m3gY28u41aUkiphJKQpdt7NIm0OQsbWA3DWV0+Tc7zt43Iqzw1h2wVKS6b8MqhYKOgdXjYaQz+Xpbo9BhfQ8GUAU06h/3SVChNJWEE8R+ONFMuEGKmdRDuUEYB2Px+dMlyBr8x3th
Uwv4VYH6wlX9o7L+iaU2E6OdUWwbvM1W0oGpS2dx2LdZNOOplZcxLqGlFnJF3ZvdwsIUZgaiOQYwimtwLEdxRnL0PK2RlgP1ANpbMPYvQjUZf9Btc1v/Ij8uhmW/969N2ghQPX8cU
Xm9EbFovfHBd8ptgW8KjOax9aY4wIi+y14fUbjaE7HMAnzgjqk4R80k+6C5DJktOeYeJnukeau027eX4XK7SCPn1ILrCstu9klmemmXM+A5RmnNtrM2YYLmsd/eV6V2QPZRrR4gBJ1
pUpZXqHiJTaRi5soNaAWi3dl40hDF8ulhCc9MdTESi5+kZFxC7xfiWzG2QN69OOMwwP/noRCMNI9My3cRTNN4vsD5UzkDH1aHg6g3AdixdZKt73i824i9IRpsB+vXmcA+1+6GBDmiK
/Oaj5BvcyB/Im4R9KFzpwZWyX1jPM37gZIm15mQrFFoTtPTPqFTdCIRT/G3EOgwNR6IBOFwXFrSLrIcDnnQ/6+mMztgw9Y9sVrA76QRtY1WHv5t61cjUy1s4CMvv5VDgi6rE58Bgr0
0VjHgHq4iv/XoiptrNySBYz3VJHekNpuk2IeYxYs8EWH5Uw0rnpeuHRWvBQoHRbWzEO4oIQ86krUqD0UCDCa372pn/VbgrWMrBkFRqtN6Rc8Wt/RMPtRm7R4y+Bfivd+yp+J91DIDW
dsWZzShVNcvkS4uQLLdGm+G9emBRzkg4ys0Dj1GnWUu7ognhKjMiezFvWrG+6bjm/dcUBxX1uEB9jHpM18LzGHbvde+H+MdBt2apZUYiTZUS8n67LNo8kq1noukALKeAbe3AjkdbTF
H7Ien3rvOyjpJwzLlG04xm//Z4gKaNQZXMz33pR343lxY+Qm6jQl/Kcd878QttQkfM3td90ijHdx6c8JAo7+ued5yZCxW8t6jmxIPA96mRgsvoCtr8O47Y3FV8vZhVxj6oDa8EWXz9
qEI6sg4IZDQCiM+pEyp3W7PpnqV59wBN3PO5XsBzKd9EQ+NXFuefZC3UZfs//WV/8zkmLihbBj3rp9xm0zPgAT/wAAyeDwgQf2SMmDA0lSYWnOCqrmyTF6YRoVpiRmUPSQaDHEhGNZ
+JvBa+4yS9HyoBO604VMC3e/3Sn29D5NGRhBhxTO6GHY68BZIPkTvnZSkhToxBBA+wz7XywN4hf5/JlrgATTKKtcNBH24arpBOfJRUpL3bWI44H+gQpwuxTum0IbVMbxaCiV8dczUr
CziNZveb222WT5M+//YPWbHztVCPCzVBgO4NgyagWrHz5niZ4cwpzbjHL3v2QpOh0ozco6CQzBjSCUYJ9JqpsxMYNgCpXlrk9faI+xmzk1RzGzUwP5swR9P8aVznkxpCpVjc4AwWeB
Kw7GsoIcftLy5fAyZqcdER605o0k21J0+5XZhJLtpv9ArvzHlvN6v930Wh5t2fhnykJGHic4BNWhHCyARVDUdopTe42Em74L+EFurhRgoWSceQby0/gcaf6+RxE02pGOE56XPlCIL+
+eVsfZMKWzzvIhHr5hK528JUU8s7t12Y7H9jDBevdS8XCNnSUSZ/ZN5hB1ToiG9chAZfLfEM0qtKXPM5bkpyomzedGNy/NYoU1ptHJJp/thQ2A1gFXuvoKaXfdcKobafGzTqgqIzNb
7FOgwGhrH/yKaMrHbQaq/K3YNDLKluo9cJ5b7JqdH77eIm0STYMSmA3G0tI38HKfoJrjPn7YwcSrX81NRzjjnz6CwMKIXOt9ufhUXp2VhcWg+hGWJ6j6TG1rqMa16KBEBTVW8aRccj
vDSnWsBGt2NLgi89T+7947CWpsMn2bzbkx0DHfpMv/5xxZSP78MMe4NCaCoae4y5OglMM5pWRG8tsJRYspwyZrGSMLsLJ/CRNggIzoGmqNj44j53utmaPWysaqmObCEQ6pvQy5RF5O
FqrMJvbanVHLMqEsbAkhvsoXadBoUR4xFPd8f10TjAGSYOLkO2oUFU6OTXczsSW/K23UPQ0gWeE/G9bsl1BFpaOHqGw7b4+YWUWNpPIKBzBmeX0sCg2Vgkh92DEKVX3W/Q9C4aT/py
auDDRh4RlIBOZ3uR964vP6pO5xA0MerUnENhnfmmOGLCxDudKlXzv0hqCHXfwApnGZmaga8WuZ/FkJ26frJ5ZpZ99kL1kw9FnhtPu4DPDnGATIPpZcVfBpWN776i5E9hiUoxT4nYFj
Q7/Nj9o5os4DLBFDgXm10XD8Jb1wlwyNwFYGBJRyVRnzSbA+Sx9PDbdJzuLSu3yB+l751PbQxQPqdOdmTNidJtM28J/HnvHiqCPbcSlR31V0sOkcN26SmrFysDRMoKB0eUSGP1DpKd
c80/trGuEgPGhtPE7vzpz7oINA8pDJKhLd735+6NXzp6gBCvTciaQkSViJGU09kW1Mi69FLgovjaxYvmgOYMqnG2Xr721jEV5DjIGMgXdQw4D4k8h7ImqQG+aPRKS60rBeVswO1gxm
CT9OMxX4W4dDJS7+G0ZC3JL3dFCPdvqlwwP0lmlHKx0qxSI6Re3FDbG74ICHFFMI/XG1Qaqzh/btJ2DCJTE0ZP63ZLsGI0n9/GRKFGXS2kjNOLQKzfWT3J2BXBCzBYkVCoCY/mPyEV
M4NjHH3pjFE9LyiDbLWL8LBhRLHFrCfzU8cmfmYQvaAUe4L7ezvYMFbVeGtxAapQhjJlWCwsp/GMIgolwBZP83vXKsAmiI1sJg2vnsQMa6ITukMkbXU1X8dcvcPGTuKMbXU1X8dcvcPG
F2+tbsw2tuOy7olmFWWV8OWr61GIi64pmzzkKEB3Q9EAQxysFSWFi+q+nMV4LFKSr36Bg3KtWKn5Jt+c6cnfaisGZdHKP6xDR99ue1bk14gvHuEO1WrDlBMKnYPbnjPB/3p6tnhM8L
Xbe7AxDnwVeWCCIAI9wMlfbziowgBdsIJgmMrCZhjAOoDQMmbDOtquZYTP+yI5WmUT+O1moGJDF2Y3CspYZh17jyyc4gNgkmwEdp89QligcMPfX6W1DCivdWuRhkLkg4xOjOH5Puad
7QP2Io7whbOkI6Kunga5KMVjc4RwHfjwSogNXSYbjg0tyA+nW0ucGOQ1xm3o0DYtEs3uP1A9GNFV+phfy7ZmG8FaE3qX/6R5HF5jPg1P2AUNAIzfQ2NXlYoIfWi8Qfho6AB9Ua+6ts
AH2vdfiBmEKDjw2UykKjXb3rR35+oaywK+hgTlQkZawhXHJSOKZLHtgtxCbKaU3RSXUgX0bK5O8n5pDJgQdsR3Pn1Zz8WlnKoOZ/lR5NeYNHi7VGtd9jENG/fTMC2HMoyBuRLE5/2
z1zE2mrUTs9nHlUzC6r29jexz61F3BKppU5Fo2GylPQD2NJPHZUZaoqJ+ID9QM0r3VWcoTjcKC/LqI3I6NxXoKfxncK113b4I3UY8THkKZEJoBkFXjhjkSj4Petllbdc/BsHEMafq4
ilZiCAj/ll/cJo2tG5XMkU6qG8DU/iqpErtDlA0bds/kEmbf/up/KOg2/0YNCcW9tONvSJHfoA28tXxY7B7NfqWfCrWcBC/SRQaHA/9VCmuY3YfyfYPgR4PptCikPMkKU6uunkrgJG
lBjzVk8kn4JE4jehZb/6qmq7jWct8N/mXQTP/y4JZ0qq0KeKes3pHvck7ksFD4gmk0y1F+X1jye2cMBow2xIy/INevZZkYIYhA/bb4PIjytGX4d0usziDZTc1vOC6ITCfas0mfpCNd
3SIS/chvCj2Nkru8etxSpgDzmEVxbzrInt+YbZSNXPo/eKFr7d2WjCLxTQNuf3I8AzX8BwrpD2DevblItoHwkcqzjul1qLqMM0DL3J6T1WWz1KKjtQrwvLBlCkxi1vy/1YWD9p4uM7
DkSYaAv8XJg4rK7Hnfu83C5JF+sKXdIwvmWUQQX02zsg/uk09zEfWsjHQcviocITLmL/ScQRsS6TzbBMG6UaaNrxk1lUMoC4K3Vkn4GWu0FLDNWxccnkPj7zIPSZR01Wd0K0jIPwIF
HpMZyXJem11NhH4iN7gSzN6rBqtUxV9+EEcxZLpyWMhvtIMAvoDNcpPGpz+Y/kIfDO7fnJeyvNLhud7t/MXgIU2p5oquIKZva1vevIq5yCXiokyFU9x/woPRTcNbb/QZEo0UrgTiPO
LP0ZPlI8IxqWWiJTuVNqacxNhJ0r7PX1ea5ZabhVHqieqoakWVoVPXgSGMb1EF3dGx/JVEtf3LqIpUymcT/4FKOKqZvVRZE9dvhNYU/zDRDv7i8qtDoHrLumTrUsaqNIuXxS9GyYLB
vCrl9+uU1dRZrKCGXpk1BD+Yed/6dMvrfpTxYrgO94vsdGF+YnunJDxBJ3fKhtd6roopi7tcTCUq1N/IxwXXREULPBfRjugPw2HnZAH6gBAfmCgK1Fim+/Zu9X0gqMpijBcWMqyRIP
BfUQid3Ss6jZ7oeajhNFuALk3ntJEZOy6tAz96AN77PqEFxsmYCvLTMeOWq1yrkuETDor5bZ8QdtFTBgOnWdtjRRoeJDZLF4YfoRaBbVvF6fzUDKZ146NDtYjdzCmgUM0J+KA70KZv
qch0rU+LMNdzNkGRThGAbxD1Zztcdvojp7M7d2og+LcdKSyIRXBkjSeAh8viA/gWRYMMm9y2UlVv72CdyAolD0VGTqMPwjcDH2Hpin2TrNP/was75xFq5q3Nhxme4/WTD2Aon/bGB3
4uAkf1hiBYV8Msccvg2y//VY0gGaDpRt72ml5jCU9G2juWIVUG0+wL3WFu678K1HJepiDElPSROw1MyX2XjDk0oQrr89TTZ/xfmfbb06VUNqwSt21S/yQHXFxE5lXB48+8idL3syfN
D4h8uDivc8JuiPOAVCXNBotY4SXRC02RB41kGybYKGc+7s2nYDvclda8/Ota5Swiw79BYNnlM0z4OO9bAuRFr2GfChtvUTj+YhlHDpwYRhh5dWt1qRqArc4TCm2ZmsYwLvRvzXauHI
sbmugC+EGfeuGEMSsy+5iaRSJ7PWmAwfm5NGigIgB27m152v45MeA5pArRHskIqYD+OeTuPHH6XL0mV2m2XU91EbpcD9+Kv9DiZ+ZcpCYtOfXW1hvx5ANNRZNoJtDafZz6y5fisMLk
UrpGbdf8xxg9KmX+AJp5rgGy55B1z8ro3cFcodTVr2lC7vydIGdAS9DLX+uUdYLpnTniFfsAbuzgzz1mgH6ySITOA75NEZ4F406NdSaCaQoEbRr7tntrfMj0Zffc/4Q/LJ67NftzC
WLT1Vvh4Le0oOzw4Af4PAwe6jHJ+D98DjyDnz3buItFzfQun/+bKaGztTCiGMXPT962Gj16I4GTl6nKaZDu/CqPCuaqvlITMd6CX0bR1szCCVc9gh1GIRybenv2NthRrOkLGhuju/1
Ub0gC9cvSAFb3QEgUZ3ivf3ZmW8VUao4F4ghVb2xr6cItJX7rv3YcazUdXDKWNnqT8zCnf9u/r474bcNnr+y2FcF8HVN/XLoX2H7sKmVaXYm6yWb/Uam+5cii6ybmvuaAxjsa4WSU9
fZTKGjII9Vht92Wqz9P5jhP0xSI1cKqBnKKJ5Aj7ZMqp+G0derNy65IXuvFVe+uCAqG5tzsm3iu63i6k/Ed/WeKC0ZbLKGOub/tGtxuxPmKP//jNoxHkAIOTvprV0JzH1oLm5oSAPch+7
pTT4HW8ykBT5VkkOs/OpBy/1yOMoYDg0tOfkNHj5fQorzXjluTluNu5jcEr92vZJZbiACp6QC9LPWQkNXDPemEHTZj5oBUd8IoFTgCzOH3gPMYA35EIrxN38xuiysxmi6ekIsJjoJh
Z/ldgBWSqtdVkb/cEITFk10kMxPLuyRtJl5SfuggEU0tz1++k/irKgP8IY8v1fYYCUikDaVXc+ob3io/c/tafYqCZd56DpwwqeqafSe/LprLd9putFz7sIZcSXj59zkgs4WcGV+Jtx
GNgkbbFyhQublaZ0vWDgPBK1cydjutHbqOz9kRl1RiOKVq6QS6hGI1i2kzpuFVrpiCHdsjd4iT+q8zYYo5Qs1zWOVTgjwt+WK1V3dTD28D78G/i4LfccuLcewgQkMT4vuCeJyqVzhu
wzw6Htl6/i2rE0DjJshLsEVhuxf2S198I9W9ohLgEc4yRW2Zv5lk/J0XZ3RmpW/ZGVBAbeZ877nSptGNinrVC0oRRIKkQW2ktlr/UgUJABl7taAjboiRGRXl/w691+gnSJT6/9YhJG
lrcfv6fo7SShYxIBpUb2Cs3OgtBif+2aVoPeyUQtnDFomYFepkzYd65p4p6sqLsK4qmc5qZ9/rd+spms+yu+00F75Mu78mDj9ao1cvgARe2anZdcioFPCNDxba8JWUMtWMRrQY2MAb
5tN2YQU44PP1AKW4nWOMSAvHJFXUu5et2XjmrEtt0B5QViwfZAb0rUqj5Ia50VRbr0rvPxFJvZK2uhjP7l4z12jvjbUscak6KeL94tyRHq3Km4xfEiBGKD3Q7nuX+6JKMMW9VGlJkb
ApRo3W4bBGks7Hd2izZ+hqKls/CFpMv4PJzTGIqfo202GWo/1zx0qod9zhXyfra7DoHPyjheDojQ36d024o+7q7qIIQYe0CSUe3mAAcjXXs7q777SXHrqJuK6sfLZ5bQN+zVEtS7WO
f2qal08qnHZW7J8vF3Qvo2e30mzg8G5YdTJRULmPzdrRQuglIeTAvN2ZnF7FPo//7J9hNdNuqHhAQErCQIaw2ZGhmK3LxP1XCODDJsovNgCdcrode0nrGRJTp7VfQvqMwOKfB3g1BK4
WjDbzvjR9a14JQROh/NhwlKN8Q2e2XvzAUL0ElEcBGtFNzQid8b0CyI4B2jcmR2eKFAc0/sJlRnbCwvUPAUbMTyuWgBGLNFPzQY76p4XKcVoUBtn09yAah80lPRcUbn6yFgQ3Stbe1
ipcG+fG5rUKXn8DyHumq4V1M0cbPcwcOtK2pSIms1h6RxNZTJK3NbJE9HW4tbAsTpqqgRQ2l4QAu8nKB1SF0Z8Nbzgrcz2GOZH76tDjduA1ChqEHGpFnApgP3nQWvAYTE5jSmhNLVd
nbCf0XB5icAw9DIAT44m2fby4ZOUD9S8uceX6rpMIMTgB7zZs6Re40iakXXnWWPRbP7oNAXlri6PH022e6Iyq3bsl8fzVW+Qwbia7AlytmxvO3SvTsjWVaC27d1iLO2Kz4MuV6Vda
Md3RejTFRt812M1gcho0Tu2Aj1unEmySJWjM5Fm131hQH7jfUWUFAqhd+rgb07VrZ023BPHdKfVPIElvWzq+xxsmqPLkS9ZgwAb5eoPkDOGR8EXeyjAtUuj+ET5t+7J0x5cHorPCUQ
mE6z/6XR2DQmoC2agu2XoHM2FrbcmJuE7vOFBJfVuEgQOk9SwXJva/phuFdT6C6BNHeLKqdXzhHv1sJkiI5EBt/9UyIjqdLsJawGJ5CD26yi/JaoY0ar1vaNO1IuyXQ7wJF286Rvg2
W4Fdu+o9ZZ+ykYi5jLN2cEg7kWt80MqhSbKsh4oyR57TrwjVSj/4akms2fRSP0gizB5WQaJiYonbHtk079tz5YlcCHHtezQeus7fZe/9tKGPkF++IU9pPShVHOCB44AJ79U7QecnM9
s6oAFsFYkOxKitcYmhn7dgI3Z0lh2x1QDgovu1O3ag1kRecHuFeF/rCm31Z+MUjNh/dX0cewtHHfJntLh3hEUyweaVb4dIIscuzo8UO7n0l0jPIAUdyNRzLvESQe47QA/XrFdZ8ToW
U2gT6PfrSjwsthT5iMR3vDtdvBHcrNYAd7pE0VArwAfAy8lAohpthzL3pqLaf039NWEpFp2aWRtQHneaYrdpP+aTNAkTBNiojcl0fOy2MPyAk8GzEVdBpiip5xHPDbs7tYPVh1oWml
4es5rTodrcCsqDerJSQ4fCVsOPDve/8os0eIsKSaQ02/qV1u4ISzSCGnEWsOxku6FB5LTyN9lWTVKxJnixR8Bc3LqrSDkVVCJ/161/LEmUmHOxyiycwP5da8z1/dttAI37nlDBx71+
Kx52Am4sOQCksxYElzx1JWJAEL3BDcl5qmn+iEz4i6JnvuCXXtks6GNb7wTZb9ku2IZT+8Ha6mkga0TZGdn+4thGb4iKq4kZciTuTes+/QHt5JpudR2KWQgW6YQJIQTJh/aKaNzVoM
/QSbIUj5TpAfr8+EjPUXV7esh9tQwpPS92gXUKhUimJut5qnE+aKLmnKB8WLk7xbHb/KSjQxRwpXSqGnYHtnmZecwYfy9CheB0fITxZTtg+ajcnWr6wrillIa7WOkJct6yFYCl4+081
FmDoTxVUC3p44WOmKvMjyhGrMnID0F/cNFyb21S87KVy8t3GaGTLDKL82irDmiROqivnNmnR7oRKQXjCidSqjux6qEuTlzCdGPDz9zikZ/D4orGa3CUFDX6gEfjc5P7heDKniX0odF
mSPowtB/9ckvfEgcHRAN+ky33MobbwNYyQWN6VQ9wdNun6EtmXY5lPKArNam4nAuRqlOqMd8QHjW2RPnyUR1BtXBwOeg3uEePdbLGLNis5ff/eiucq71iszhJaycFgxd8uindwj//7
YTW9rq8xe7Hp+kqKk6AxsdveDJvoJvogdo2NYNaQKDkgdvoajaLeBga8OG/Wwi7bE70uvB+UZd6Kob3sBl0OZFsCSgg7eNv750n0Slkii4PEbXz4toSPAHYy9nGeYB6FO6vRGLsYaP
BqRDffUFLHS4lO6vkVsL6X0Ps/sLsJhvkpTpfSrhhB9lOQ2CSOAyKVHcwduckp3eBDuVEAAACBg2+9BlFBMAAB3uECgNAUhIULLrHEZ/sCAAAAAARZWg==

Index

abapys, 178
Adaptive Verfahren, 40

Barotropie, 12
Benutzerroutine, 46

Determinante, 7
Deviator, 7
Diagonalisierbare Matrix, 7
Dichteste Lagerung, 10
Dilatanz, 10
Dimensionslose Matrix, 7
Doppelte Genauigkeit, 42
Dyadisches Produkt, 6

Effektive Spannung, 11
Eigenvektor, 8
Eigenwert, 8
Einfache Genauigkeit, 42
Einfaches Material, 20
Eingebettete Runge-Kutta-Verfahren, 39
Elastizität, 23
ElementSimulator, 130
Elementversuche, 12
Explizite Integrationsverfahren, 37

Frobeniusnorm, 7
FSAL, 39

Gekoppelte Euler-Lagrange-Methode (CEL), 179
Gleitkommazahl, 42

Homogenität, 21

Implizite Integrationsverfahren, 37
Interne Abläufe, 60
Invarianten einer quadratischen Matrix, 8
Isotropie, 21

Jacobi-Matrix, 46

Kontraktanz, 10
Kornbruch, 11
Kronecker-Delta, 7

Lockerste Lagerung, 10
Logarithmus, 7

Maschinengenauigkeit, 42
Matrixexponentialfunktion, 8
Membrandurchdringungseffekt, 121
miniSoilLAB, 118
Mittelwert, 9
MPO, 126

Nichtbindiger Boden, 10
Norm, 7

Objektivität, 20

Phänomenologische Stoffmodelle, 24
Porenanteil, 9
Porenraum, 9
Porenzahl, 9
Pyknotropie, 12

Ratenabhängigkeit, 21
Reguläre Matrix, 7
Richardson-Extrapolation, 38
Richtung, 7
Rundungseinheit, 42
Runge-Kutta-Verfahren, 38

S4-Tensor, 6
Schnittstelle, 60
SimpleScriptGenerator, 178
Spur, 7
Standardabweichung, 9
Standardunsicherheit, 9
Symmetrie, 7

Tensor, 6
Tensorkontraktion, 6

Unit-Tests, 113
Unschärfe, 16
Unsicherheit, 9

xMat, 2
xMat Funktion
Abort_If_Not_In_Interval(), 232
Approximate_Jacobian(), 236
Base_Initialization(), 236
C_To_Fortran_String(), 229
Calculate_Dot_State(), 236
Check_Input_Dimensions(), 229
Consistent_Jacobian(), 236
Determinant(), 233
Deviatoric_Part(), 233
Dimensionless(), 233
Double_Contraction22(), 233
Double_Contraction42(), 233
Double_Contraction44(), 233
Dyadic_Product22(), 233
Eigendecomposition(), 231
Elasticity(), 236
Estimate_Components(), 229
Export_Matrix(), 233
Export_Tensor(), 233
Formatval(), 230
Get_Direct_Variables(), 236
Get_Direct_Variables_Mask(), 236
Get_Dot_State(), 236
Get_Elements_From_Matrixlist(), 234
Get_Factor_F(), 237
Get_Matrix_From_Tensorpart(), 234
Get_Statistics(), 239
GetModelCount(), 90
GetModelName(), 90
GetParamCount(), 90
GetParamName(), 90
GetParamUnit(), 90
GetStateVarCount(), 90
GetStateVarName(), 90
GetStateVarUnit(), 90
Givens_CS(), 231
Import_Matrix(), 233
Initialize(), 236, 238
Integrate(), 239
Integration_Algorithm(), 238
Intergranular_Strain(), 237
Inverse(), 233
Inverse_Internal(), 232
Is_Diagonal(), 233
Is_First_Call(), 239
Is_Nan(), 229
Is_Valid_Stress_State(), 237
LU_Decomposition(), 232
LU_Solve(), 232
Mat33_To_Vec(), 235
Mat99_To_Tens(), 235
Mat_To_Vec9(), 234
Matrix_Exponential(), 233
Matrix_Rotation(), 233
Nonzero_Division(), 232
Norm(), 233
Pack_States(), 232
Perturbate(), 234
Postprocess_State_Matrices(), 239
Preprocess_State_Matrices(), 239
QL_Decomposition(), 231
Ref_Index(), 234
Select_Constitutive_Model(), 237
Select_Solver(), 239
Set_Element_In_Tensor(), 234
Set_Elements_In_Matrixlist(), 234
Set_Error_Estimate(), 238
Set_Matrix_In_Tensorpart(), 234
Set_State_Mask(), 238
Set_Variables(), 236
Sqrt_Of_Sum_Of_Squares(), 232
Squared(), 233
Tens_To_Mat99(), 235
Tensor_Partialtrace(), 233
Trace(), 233
Tridiagonal_Matrix(), 231
Unpack_States(), 232
Uppercase(), 229

Value_In_Interval(), 232
Vec9_To_Mat(), 234
Vec_To_Mat33(), 235
Write_Error_And_Exit(), 229
Write_Message(), 229
Write_Warning(), 229
xMat Schnittstelle
Dot_Values(), 86
Dot_Values_CInter(), 91
mexFunction(), 87
UMAT(), 78
USER_MOD(), 88
VUMAT(), 79
xmat_console(), 85
xmat_console_CInter(), 91