Notes

13.12.18

Testläufe mit unterschiedlicher Anzahl

* Iterationsschritte

Plots im Ordner ..\beispiele\Balken 🡪 siehe auch Results.excl

Ab einer bestimmten Biegung des Balkens findet der Löser kein Ergebnis mehr (Solver: SkylineLUFactorizationSolver) bzw. jedes zweite Ergebnis könnte noch auf dem Pfad liegen.

(Solver: eigen\_sparse\_lu) tut sich ebenfalls schwer gleichgewicht zu finden ab ca. 45 Grad Balkenbiegung gibt’s keine Lösung mehr.

* Balken mit 8 Kontrollpunkten findet mit dem SkylineLUFactorizationSolver GAR KEINE Lösung. Hier gibt es möglicherwise Probleme mit der Matrixqualität. Die Vektorlänge/das expected Ratio wird zu -nan ausgegeben. ??
* Ein Lauf am Ende lässt das system mit dem eigen\_sparse\_lu Solver verhältnismäßig schnell lösen aller. Allerdings nur für kleinere Lastfaktoren. Hier muss ich aber noch den Plotter fixen. Problem nur Bezeirkurven plottbar.  
  Zusatz: im einfachen Plot werden gar keine Verschiebungen angezeigt.   
  Ich glaube gesehn zu habend dass da der innere Load Vektor immer 0 bleibt!!
* Modell mit Sofistik erstell: Der Balken knickt her irgendwie komisch am Auflager weg. Die Laste scheint deutlich zu groß zu sein. Fehlermeldung: zu Große Verdrehung. Die Verschiebung geht gegen unendlich.   
  Problem Querschnitt. S235T ist nur für Querschnitte <80 mm geeignet. Könnte ein Problem sein. Vielleicht mal insgesamt ein kleines Model [mm] Bereich erstellen und vielleicht auch mit einem tabellierten IPE Profil. Damit man da nicht in die Probleme mit der allgemeiner Balkentheorie kommt. Wegen zu großer Querschnitte + Querschnittverdrehung…

14.12.18

Fehlersuche um rauszufinden warum die Matrix beim Element mit 8 Kontrollpunkten nicht stimmt.

* Genau genommen ist in jedem Schritt der interne Lastvektor „0“.   
  (Achtung! Beim Plot Balken8 Figure\_6 treten trotzdem Verschiebungen auf. Allerdings NUR in x Richtung.!)
* Das Modul Beam\_Validation mit Yaml-Check, führt wieder dazu dass der Positionsvektor r\_1 nicht mehr stimmt.   
  das Problem tritt nicht auf, wenn man dem Solver keine Steifigkeitsmatrix übergibt:
* // rLeftHandSideMatrix = \_gke;
* // rRightHandSideVector = \_gfie;
* Auf dem Stand jetzt läuft das Element auch mit 8 Kontrollpunkten. Das Problem war wohl dass die node\_indices nicht richtig mitgegeben wurden. Bei einer Kurve mit nur einem Element war das kein Problem, da sowieso immer alle Indieces mitgegeben werden mussten.
* Jetzt muss man die nurbs noch plotten. Könnte mit den Beispielen hier funktionieren:  
  <https://github.com/orbingol/NURBS-Python_Examples/tree/master/curve2d>
* Problem des Solvers bleibt bestehen. Ich kann hier immer noch nicht ganz sicher sein ob es auch noch ein Problem innerhalb meines C++ Codes gibt. Aber wahrscheinlicher ist, dass der solver nicht richtig Rechnet.
* Jetzt da man hoffentlich konstant mehr Elemente rechnen kann wäre das ein Versuch, hier mal deutlich mehr Elemente zu rechnen.
* Auf jeden Fall ein erstes Fazit: Alleine den Code zu übernehmen führt erst mal noch nicht dazu, dass ich den Balken Aufbiegen kann.
* Der Vergleich mit Sofistik führt dazu, dass um eine 10er Potenz daneben liege. (+ mögliche Ungenauigkeiten die bei mehr Elementen weg gehen (hoffentlich)). Hier muss man mal endgültig das Einheitenthema klären…..
* Performance: schauen dass der andere Solver zum Laufen gebracht wird!

16.12.18

* Das mit dem B-Spline Plot klappt soweit erst mal für ein den Balken8 mit 8 Kontrollunkten.   
  Vielleicht noch mal mit dem NURBS Modul probieren. Sollte aber vom Prinzip dann das gleiche sein.   
  ein Schrittweises Ausbringen funktioniert nicht so Easy:   
  IDEE: hier einfach Die Multi.Curve mit in die Schleife bringen.   
  Dann wird wahrscheinlich zwar in jedem Schritt die alte überschrieben, aber das könnte funktionieren!.
* Danach Versuch: Wellenelement mit 16 Kontrollpunkten.   
  Dabei kommt nicht sinnvolles mehr raus.  
  1. Vielleicht liegt es auch am Plot. Die Frage ob wir dann hier immer noch mit Elementen Degree = 3 zu tun? Eigentlich sollte sich da nichts dran geändert haben. Steht so auch in der .iga Datei.   
  als müssten das hier ganz normale Elemente Sein. Trotzdem sieht der Plot Figure\_2-1 ziemlich eckig aus. Aber dass kann dann vielleicht auch an den zu großen sprunghaften Verschiebungen liege  
  2. Wahrscheinlicher ist, dass hier einfach der Solver nicht konvergiert. Bzw, das tut er nicht   
  Expectet ratio liegt teilwiese im 5 Stelligen Bereich, statt bei 1.   
  Die absolute Länge ist ich größer 0. Hier hatte ich jetzt erst mal 10 Iterationssteps, aber vermutlich helfen mehr nicht. Das Ratio springt hin und her.
* Bei dem Welle16 Element sind auf jeden Fall Y-Verschiebungen nicht mehr 0!
* Festhalten der Y-Verschiebungen aller Knoten führt auf jeden Fall zu einer deutlich geringer Fehler Norm  
  Das Ergebnis sieht aber im Plot aber nicht gut aus
* Das Balkenbeispiel führt bei geringer Last zu großen Verformungen. Hier muss ich immer noch die Einheiten bei der Eingabe klären.

20.12.18

- Vergleichsberechnung mit Handergebnissen und Sofistik (für den Kragarm sind diese beide gleich) ergeben eine deutlich unterschiedliches Ergebnis als die Berechnung mit Kratos

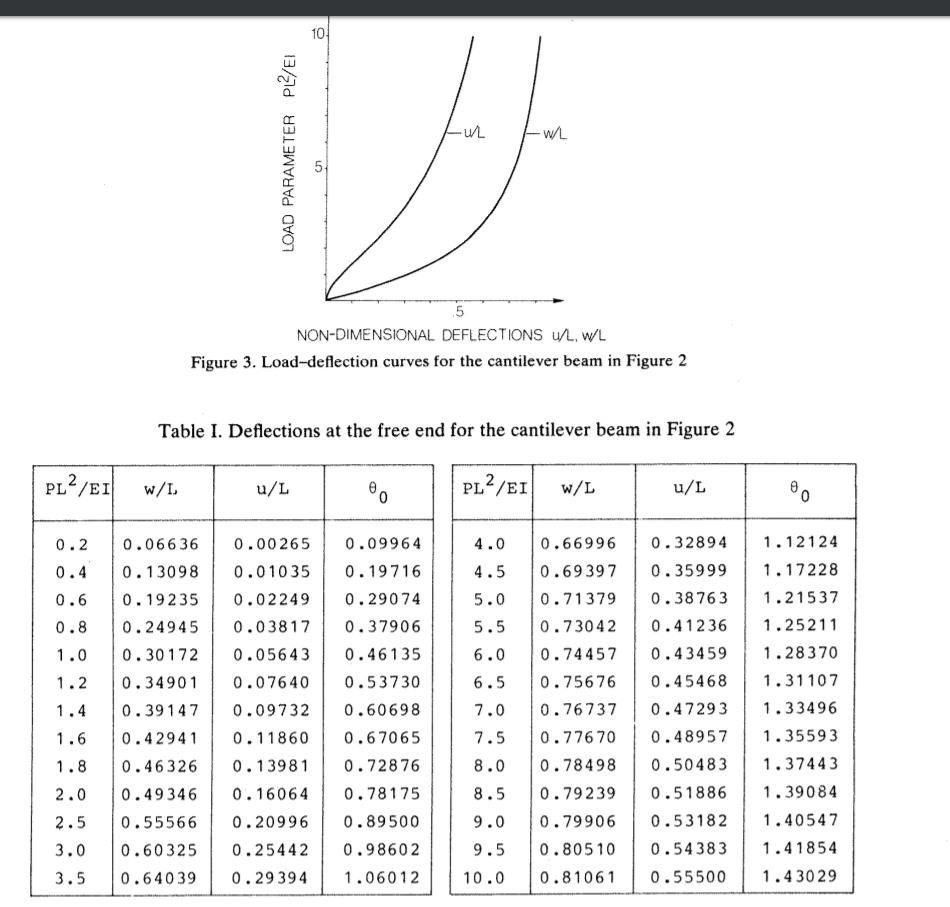
- Die Einheiten für die Eingabe müssten jetzt eigentlich passen. Mit kN und Metern

- der vergleich zeigt leider auch dass die Ergebnisse eines Elementes mit 8 oder 16 Elementen deutlich unterschiedlich sind. Das hat in dem Ausmaß sicher nichts mit der Ungenauigkeit bei zu wenigen Elementen zu tun. Wir sprechen hier von w = 0.0129 (8) zu w = 0.033699 (16)

- ähnlich ist der Vergleich bei der Integration. Hier konnte ich noch nicht klären ob ich die Domain des Knot-Vektors brauche oder nicht.   
Die Domain sollte eigentlich nötig sein bei der Transformation vom Gauß-Raum in den Parameterraum zu mappen.   
\_dL dagegen sollte vom Parameterraum in den physikalischen Raum zu mappen

- Rhino exportiert mir einen zu kurzen Knot Vektor. Der Vektor ist nicht richtig clamped. (ich brächte (p+1) in dem Fall Kubische Formfunktionen (3+1) multiple knoten.  
Manuelles Einfügen der Knots in dne Knot Vector hat nicht funktioniert. Statt dessen bleibt der Vektor beim Balken8 immer 10 Einträge lang und schmeißt mir dann immer die letzten beiden Einträge raus…

23.12.18



Gedankenexperiment:



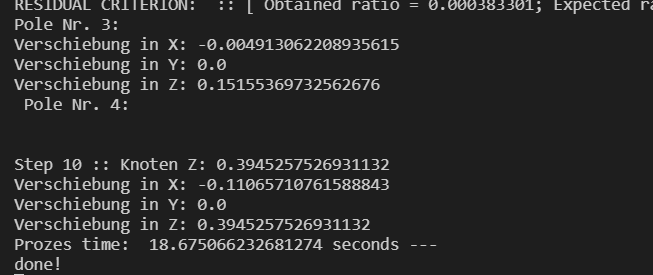
28.12.18

Vergleich: Bleibt bei normiertem Tangentenvektor das Ergebnis gleich?

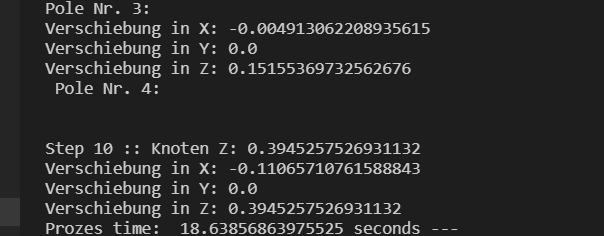
r'{"solver\_type": "eigen\_sparse\_lu"}'))

Normierung innerhalb des Pythonskriptes:

Knot Vector [0,0,0,0.1,0.1,0.1]

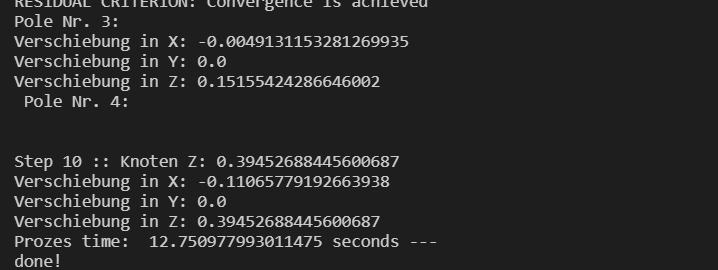


Normierung innerhalb Kratos

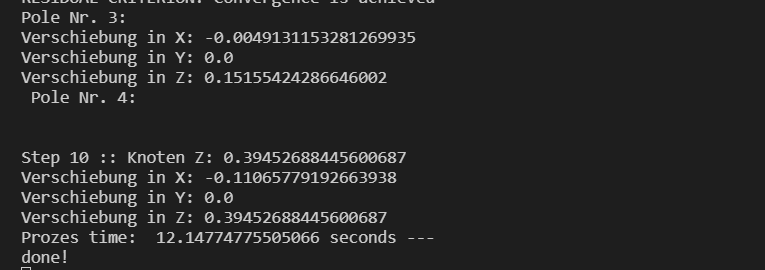


Normierung innerhalb des Pythonskriptes:

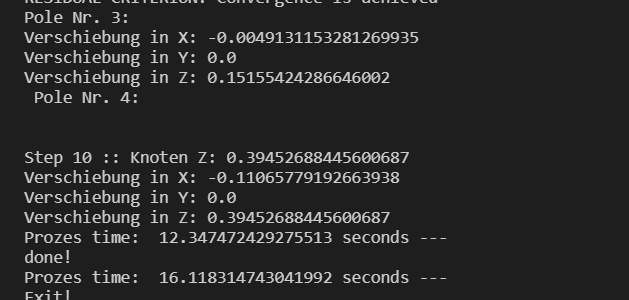
Knot Vector [0,0,0,1,1,1]



Normierung innerhalb Kratos



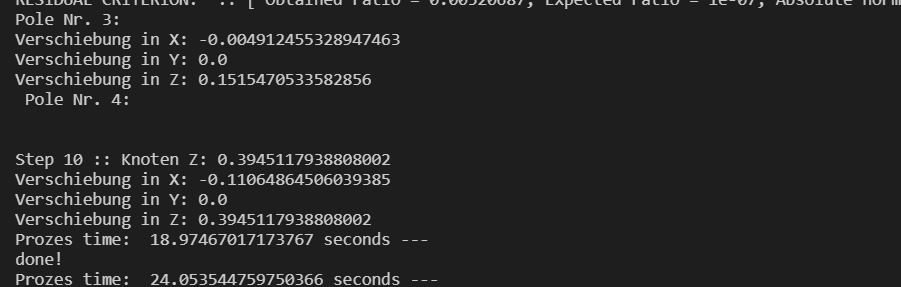
2. Lauf



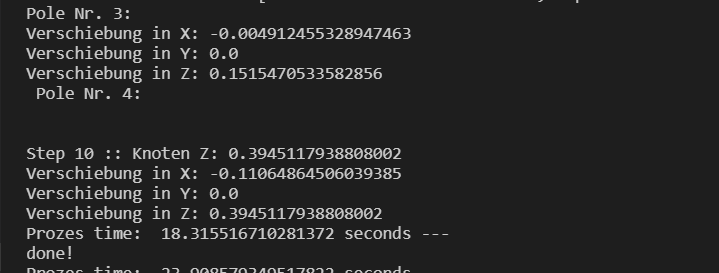
Normierung innerhalb des Pythonskriptes:

Knot Vector [0,0,0,2,2,2]

(miserable KONVERGENZ)



Normierung innerhalb Kratos



Zumindest klappt die Normierung so weit. Allerdings dürfte der Knotvektor keinen Einfluss auf das Ergebnis haben. Auf 10-4 ist nicht ausreichend genug.   
Frage ob das schon mal gestimmt hatte…. (ich denke schon. Allerdings wo liegt jetzt der Fehler!)

03.01.2019

Heutiges Problem:

* Die Summe der Ableitungen (erste und zweite) der Formfunktionen sind 0
* Das System konvergiert nicht gut
* Die Ergebnisse bei unterschiedlichen Knot-Vektoren sind nicht identisch. (Hoffentlich nur der schlechten Konvergenz geschuldet)

04.01.19

Irgendwie scheint die Theorie über die Knot Vektoren aus dem IGA Skript nicht mit der Theorie von Rhino und dem Knotvektor den ich in Kratos verwende zusammen zu passen.

* Die grundsätzliche Formel liefert die benötigten Knots. Ich vermute mal dass diese Anzahl die beiden zusätzlichen Knots für einen Open Knot Vector beinhaltet.   
  Anders passt das alles nicht   
  Bsp.: der Kragarm hat 4 Kontrollpunkte, ist Grad 3 -> k = 8. Der Knotvektor in Kratos (Ausgabe von Rhino) ist definiert als [0,0,0,1,1,1] mit nur 6 Einträgen.
* Die Verteilung der 4 Shape Funktions zwischen den 6 Knots passt nicht. Als ein Großer Bereich mit 8 Knots [0,0,0,0,1,1,1,1] würde es funktionieren.
* Die Eingabe des Knotvektors mit 8 Einträgen funktioniert in Kratos nicht!
* Wie genau soll die Verteilung über die Knotspans funktionieren? In dem Fall ist es einfach da immer alle 4 Gaußpunkte aktiv sind.
* Was genau ist jetzt das Element? Soweit ich das verstanden habe werden in Python für jeden Gaußpunkt ein Element erstellt. Jedem Element müssen wiederum die aktiven Gaußpunke zugeordnet werden. (Hier immer alle vier)   
  Außerdem habe ich es jetzt erst mal so verstanden, dass die Kontrollpunkte im Element nicht mehr verwendet werden, sondern durch die Gaußpunkte ersetzt werden. Davor gehen sie infolge der curve\_geometry in die Auswertung der Shape-Functions ein