Abschlussprojekt - WS2025/26

Julian Huber & Matthias Panny

Abschlussprojekt

- Gruppenarbeit mit 2 Personen
- Versionsverwaltung & kollaboratives Arbeiten mit git
- Umsetzen des Projektes, welches wir in diesen Einheiten besprechen werden
- Abgabetermin: 27.02.2026 um 23:55 Uhr

Umfang des Abschlussprojektes

- 35h pro Person * 2 Personen pro Gruppe = 70h pro Gruppe
- 35h pro Person setzt sich aus folgenden Annahmen zusammen:
  - gemeinsame LV: 60UE á 45min = 45h
  - Vor- & Nachbereitung inkl. 15 HÜ: 1,5h pro 2 UE = 45h
  - Kursumfang: 5 ECTS = 125h
  - 125h - 45h - 45h = 35h

Abschlussprojekt

- In den nächsten Folien wird ein Projekt inkl. minimaler Umsetzung/Anforderungen vorgestellt
- Diese minimale Umsetzung entspricht 60% der Gesamtpunkte
- Die restlichen 40% können durch Erweiterungen erreicht werden
- Von diesen max. 100% werden die Minuspunkte der Hausübungen abgezogen

Aufgabenstellung

- Eine Anwendung mit Web-UI (`streamlit`) soll entwickelt werden
- Die Umsetzung erfolgt in Python mit Objektorientierung
- In der Anwendung können mechanische Strukturen (mit gewissen Einschränkungen) definiert & eine Topologieoptimierung durchgeführt werden

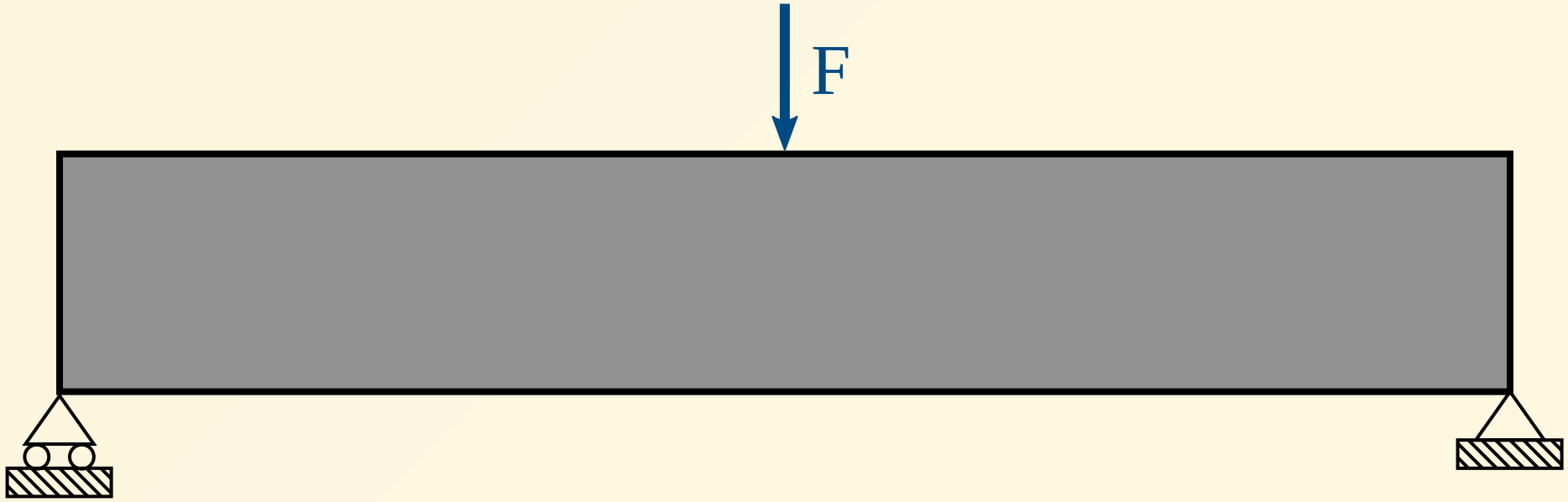
Alternativen zu diesem Projekt werden zum Schluss diskutiert

Funktionsweise

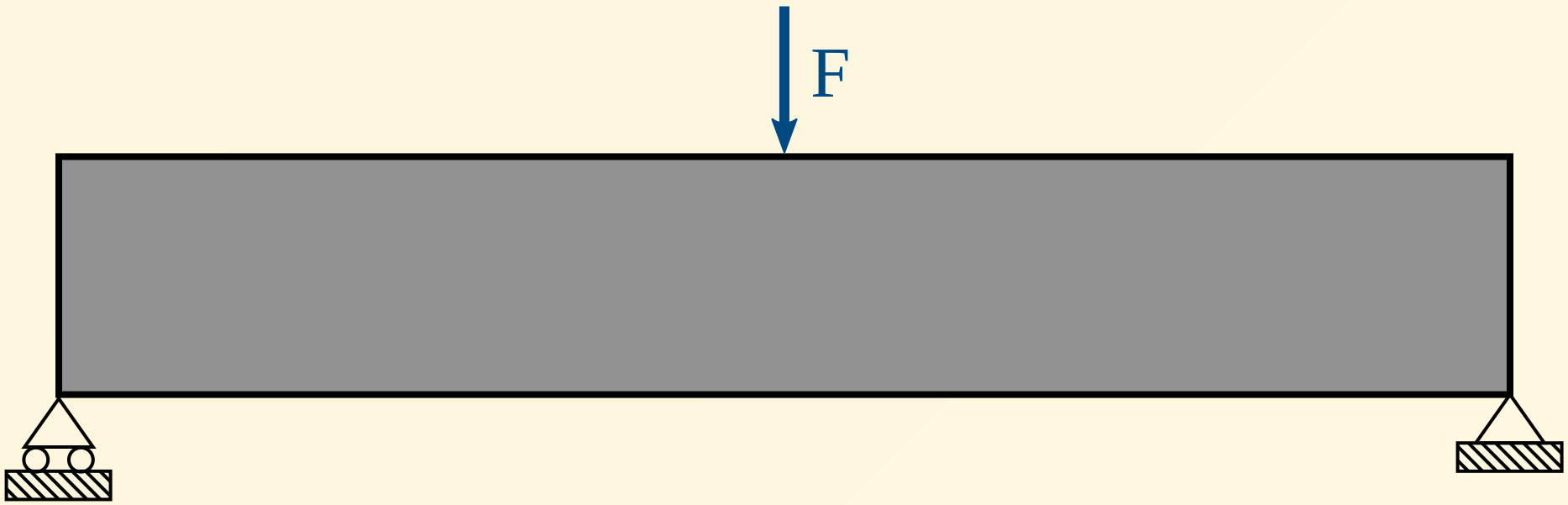
(Simulation mechanischer Strukturen & Topologieoptimierung)

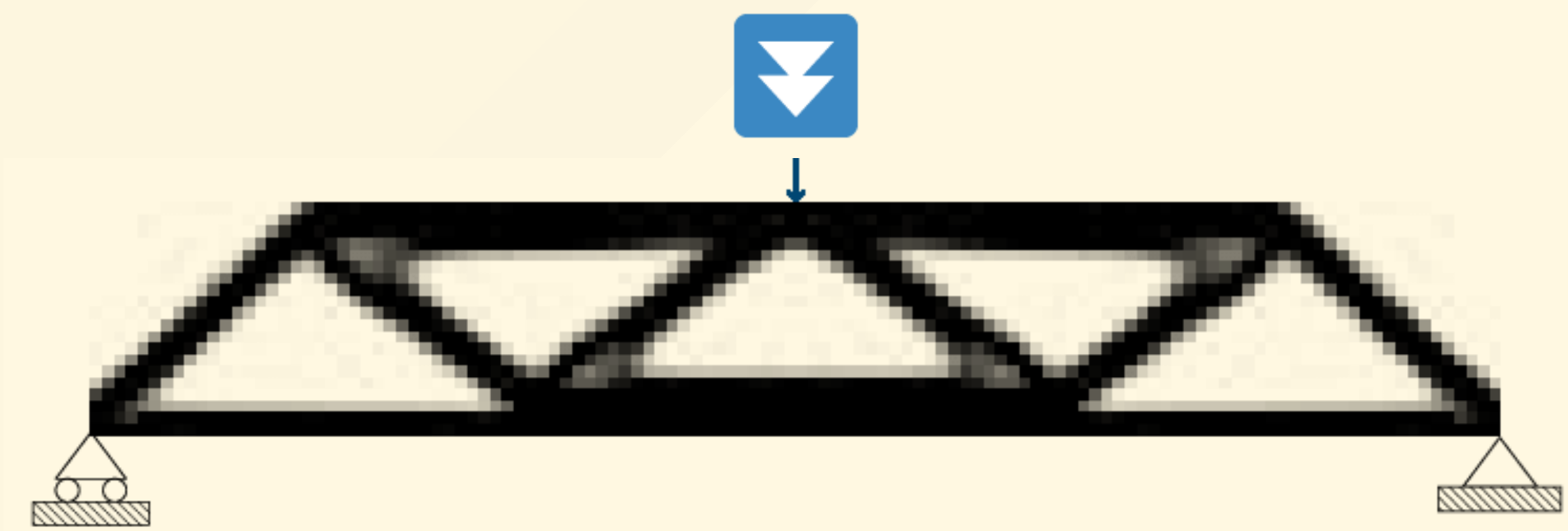
Topologieoptimierung

- Problemstellung: Mechanische Struktur mit vorgegebenen Lasten & Randbedingungen
- Ziel: Materialverteilung im vorgegebenen Bauraum optimieren
- Optimierungskriterium: Steifigkeit maximieren → Verformung minimieren



Topologieoptimierung - MBB Balken*

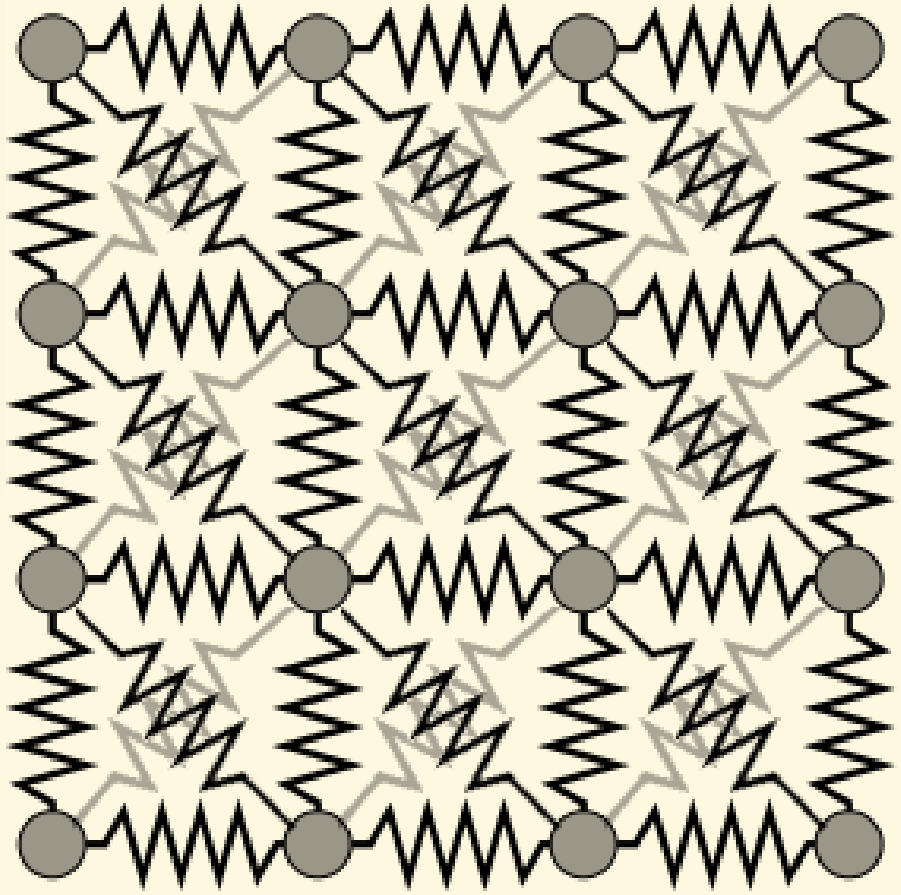




*Klassisches Beispiel: Messerschmitt–Bölkow–Blohm (MBB) Balken wie hier dargestellt

Modellierung der Struktur

- Massenpunkte verbunden durch lin. el. Federn → Balken
- Jeder Massenpunkt hat 2 Freiheitsgrade
  - Horizontale Verschiebung u bzw. u_x
  - Vertikale Verschiebung w bzw. u_z
- Kräfte wirken nur an Massenpunkten
- Lager wirken nur an Massenpunkten & verhindern Verschiebungen



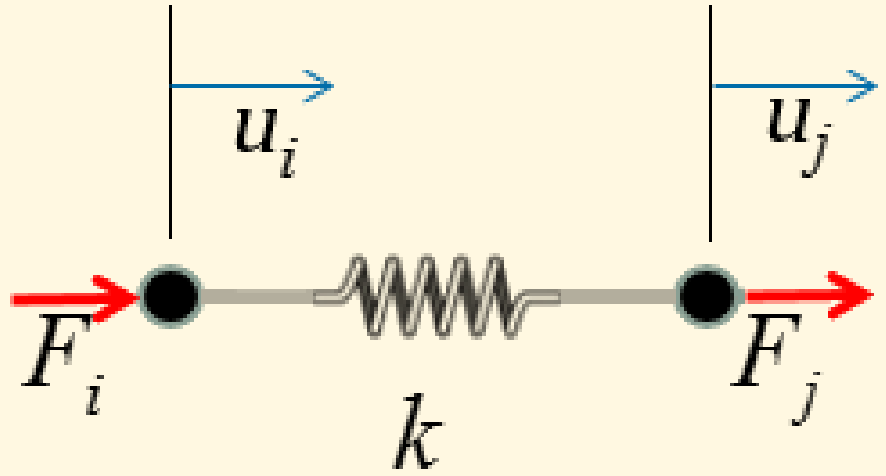
Modellierung der Struktur

- Topologieoptimierung durch Entfernen von Massenpunkten → alle daran angeschlossenen Federn werden ebenfalls entfernt
- Optimierungskriterium: Minimierung der Gesamtverformung
  - Federn die wenig verformt werden sind für die Steifigkeit der Struktur weniger relevant → Massenpunkte mit vielen solchen Federn können entfernt werden
- Wir brauchen eine mathematische Beschreibung der Federn um deren Verformung unter Belastung zu berechnen
- Wir brauchen eine effiziente Beschreibung/Datenstruktur für die Vernetzung der Federn zu einem System

Mathematische Beschreibung Verformung der Federn

Mathematische Beschreibung

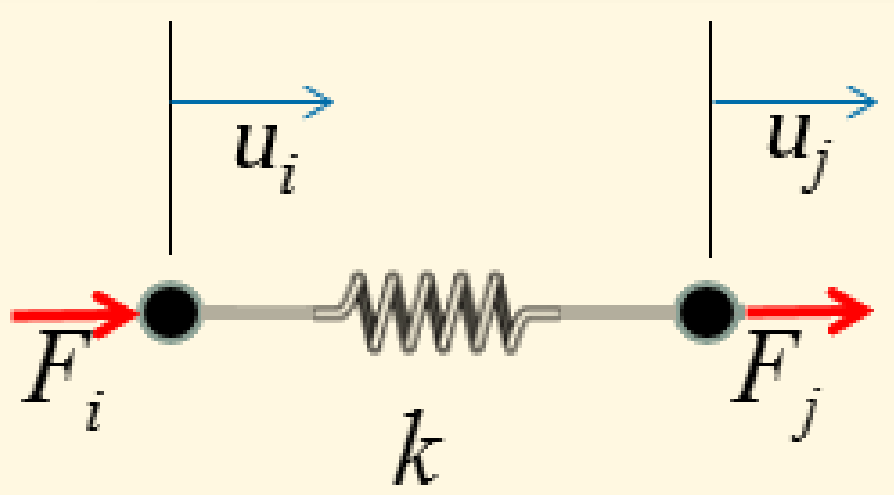
- Startpunkt: eine horizontale Feder mit den Enden $(i,\ j)$
  - Verschiebungen u_i bzw. u_j
  - Kraftvektoren F_i bzw. F_j
  - Hooke'sches Gesetz: $F = k\,u$



- Aus dem Gleichgewicht folgt:
  - $F_i = k\,(u_i - u_j)$ & $F_j = k\,(u_j - u_i)$
- In Matrixform:

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{bmatrix} u_i \\ u_j \end{bmatrix} = \begin{bmatrix} F_i \\ F_j \end{bmatrix} \Rightarrow \boldsymbol{K}\vec{u} = \vec{F}$$

Mathematische Beschreibung



- Was passiert nun, wenn links ein Festlager sitzt?
  - $u_i = 0$ & F_i unbekannt bzw. irrelevant

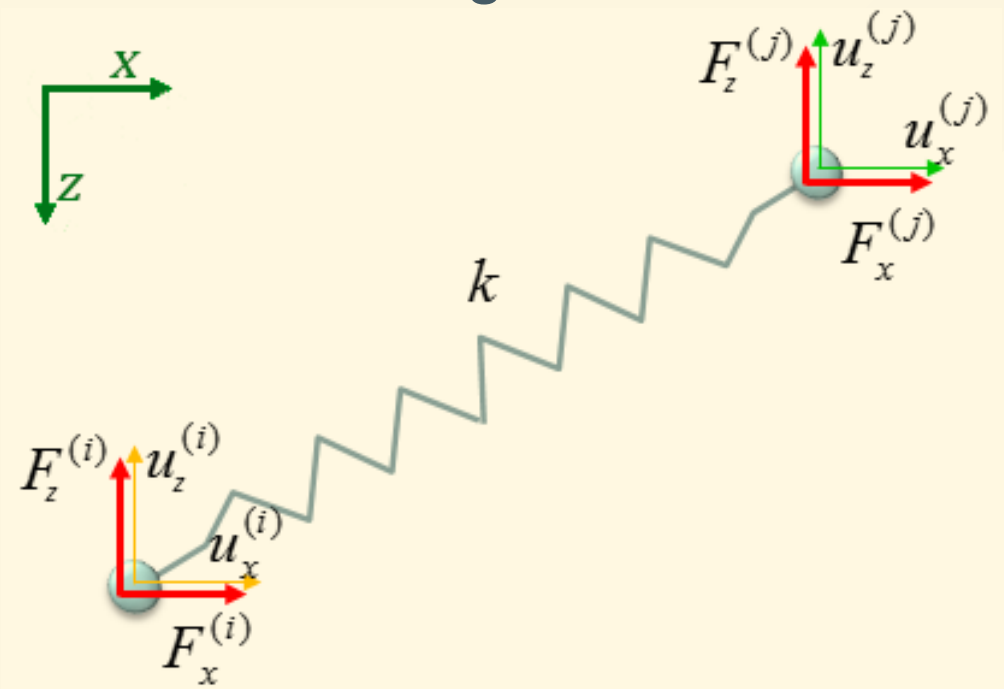
$$\begin{bmatrix} \cancel{k} & \cancel{-k} \\ \cancel{-k} & k \end{bmatrix} \begin{bmatrix} \cancel{u_i}^{=0} \\ u_j \end{bmatrix} = \begin{bmatrix} \cancel{F_i} \\ F_j \end{bmatrix} \Rightarrow k\,u_j = F_j$$

- Es folgt das bekannte Hooke'sche Gesetz → Gleichung reduziert sich
- Alternative Beschreibung:

$$\begin{bmatrix} 1 & 0 \\ 0 & k \end{bmatrix} \begin{bmatrix} 0 \\ u_j \end{bmatrix} = \begin{bmatrix} 0 \\ F_j \end{bmatrix} \Rightarrow k\,u_j = F_j$$

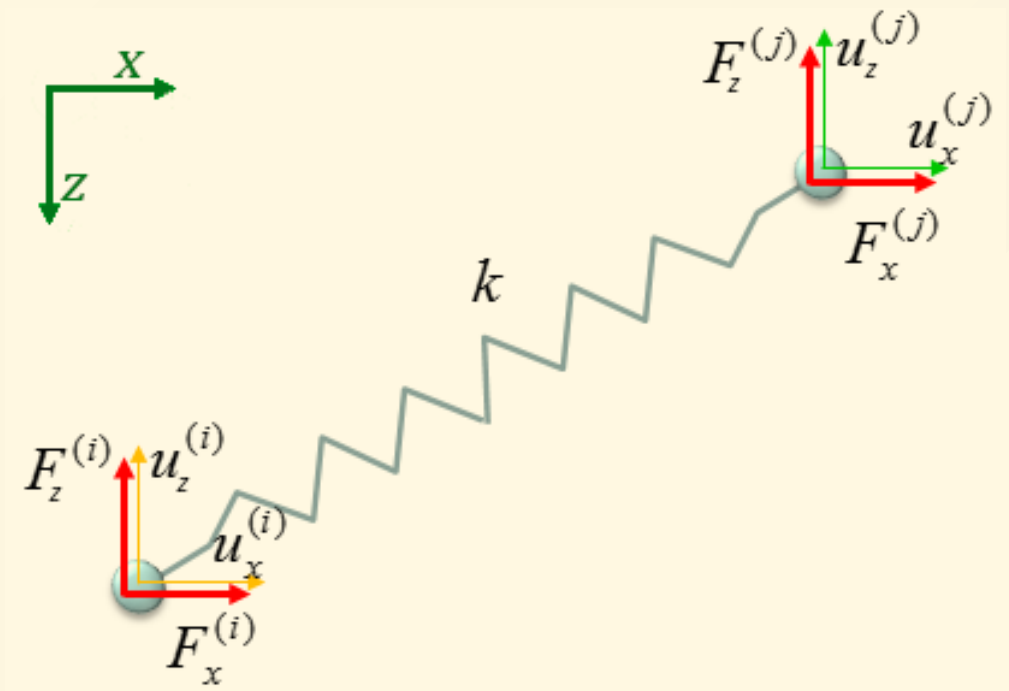
Mathematische Beschreibung

- Federn in beliebiger Orientierung?



- Transformation unserer Steifgkeitsmatrix $\boldsymbol{K}$ on zuvor notwendig
- Richtungsvektor der Feder:
  - $\vec{n} = \begin{bmatrix} n_x \\ n_z \end{bmatrix} = \begin{bmatrix} x_j - x_i \\ z_j - z_i \end{bmatrix} \Rightarrow \vec{e}_n = \frac{\vec{n}}{\|\vec{n}\|}$
- Daraus lässt sich die Transformationsmatrix $\boldsymbol{O} = \vec{e}_n \otimes \vec{e}_n$ ableiten
- Für die gedrehte Feder gilt $\boldsymbol{K}_o = \boldsymbol{K} \otimes \boldsymbol{O}$

Mathematische Beschreibung

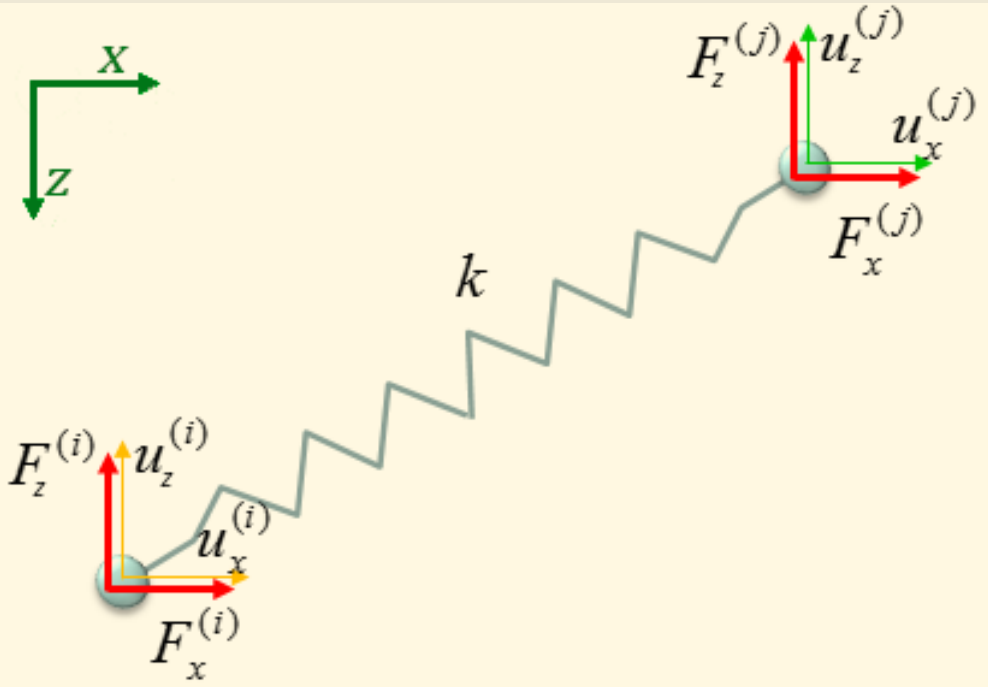


- Dies lässt sich auch auf die horizontale Feder zurückführen:

$$\vec{e}_n = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \boldsymbol{O} = \vec{e}_n \otimes \vec{e}_n = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\boldsymbol{K}_o = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} k & 0 & -k & 0 \\ 0 & 0 & 0 & 0 \\ -k & 0 & k & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Mathematische Beschreibung



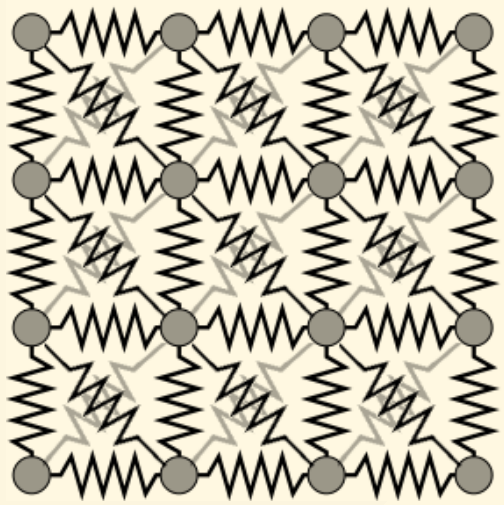
- Dies lässt sich auch auf die horizontale Feder zurückführen:

$$\begin{bmatrix} k & 0 & -k & 0 \\ 0 & 0 & 0 & 0 \\ -k & 0 & k & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_x^{(i)} \\ u_z^{(i)} \\ u_x^{(j)} \\ u_z^{(j)} \end{bmatrix} = \begin{bmatrix} F_x^{(j)} \\ F_z^{(i)} \\ F_x^{(j)} \\ F_z^{(j)} \end{bmatrix}$$

- Für den Fall mit Festlager links (am Punkt i) folgt daraus:
  $k\, u_x^{(j)} = F_x^{(j)}$
  → selbes Ergebnis wie zuvor → wir können nun beliebige Federorientierungen behandeln

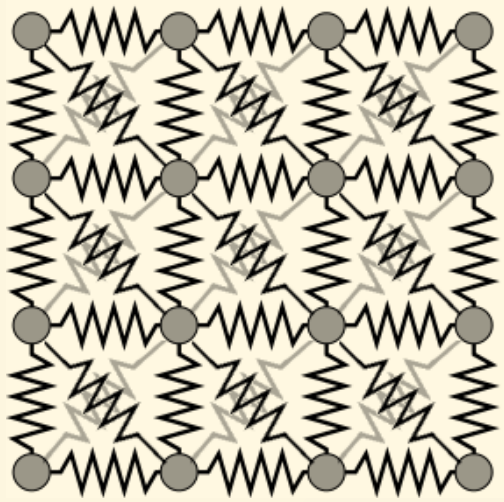
Mathematische Beschreibung

- Systeme aus mehreren, verbunden Federn?



- Jede Feder hat ihre eigene Steifigkeitsmatrix $\boldsymbol{K}_o$ → Für Feder zwischen den Punkten i und j: $\boldsymbol{K}_o^{(i,j)}$
- Gesamtsystem: Kombination aller Einzelmatrizen → globale Steifigkeitsmatrix $\boldsymbol{K}_g$
- Dies erfolgt durch Superposition → Beiträge der Einzelmatrizen an den jeweiligen Freiheitsgraden aufsummieren
  $\boldsymbol{K}_{g_{i,j}} = \boldsymbol{K}_{g_{i,j}} + \boldsymbol{K}_o^{(i,j)}$
  → an den Freiheitsgraden der Punkte i und j wird die Steifigkeitsmatrix der Feder hinzu addiert

Mathematische Beschreibung



- Nun kann mit der globalen Steifigkeitsmatrix gearbeitet werden: $\boldsymbol{K}_g \vec{u} = \vec{F} \rightarrow$ kann nach anbringen der Randbedingungen gelöst werden um $\vec{u}$ zu erhalten
- Somit sind die Verschiebungen aller Massenpunkte bekannt

Beispielimplementierung - `solver.py`

- Enthält eine minimale Implementierung des reinen Gleichungslösers
- Definiert für einzelne Federn die Steifigkeitsmatrix $\boldsymbol{K}_g$ und löst $\vec{u}$
- Enthält Assertions um gängige Fehler abzufangen
- Enthält Regularisierung um singuläre Matrizen zu vermeiden

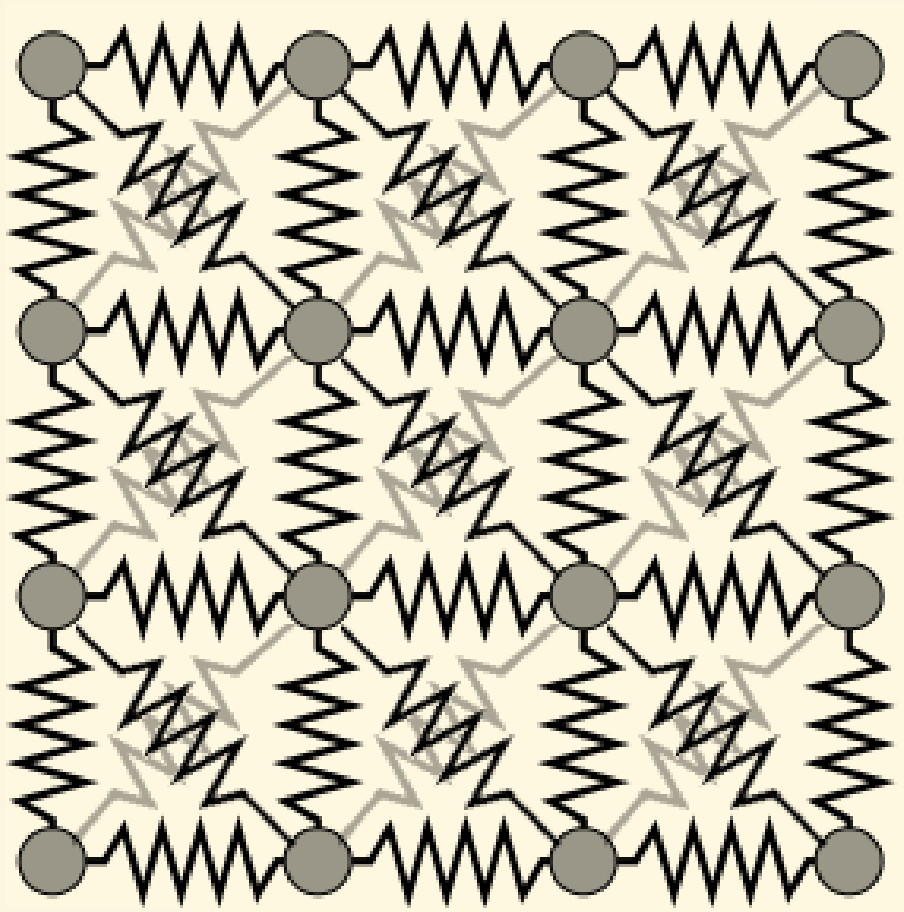
Mathematische Beschreibung

- Topologieoptimierung: Massenpunkte und daran angeschlossene Federn werden entfernt → Berechnung aber immer gleich
- Optimierungskriterium: Minimierung der Gesamtverformung
  - Die Feder zwischen den Punkten i und j "speichert" die Verformungsenergie:
    $c^{(i,j)} = \frac{1}{2} \vec{u}^{(i,j)\mathrm{T}} \boldsymbol{K}_o^{(i,j)} \vec{u}^{(i,j)}$
  - Federn mit geringer gespeicherter Energie sind für die Steifigkeit der Struktur weniger relevant → Massenpunkte mit vielen solchen Federn können entfernt werden
  - es werden Punkte entfernt, nicht Federn → die Verformungsenergie einer Feder teilt sich gleichmäßig auf beide Endpunkte auf

Sinnvolle Struktur für die Vernetzung

Struktur für die Vernetzung

- Folgender Aufbau sollte uns an eine Datenstruktur erinnern:



Elemente der Struktur

- Knoten = Massenpunkt
- Kanten = Feder
- → Graph

Struktur für die Vernetzung

Vorteile der Bescheibung als Graph?

- Effiziente Speicherung inkl. einfügen & löschen von Knoten/Kanten
- Mathematisch eng mit der Linearen Algebra verbunden → viele hilfreiche Operationen

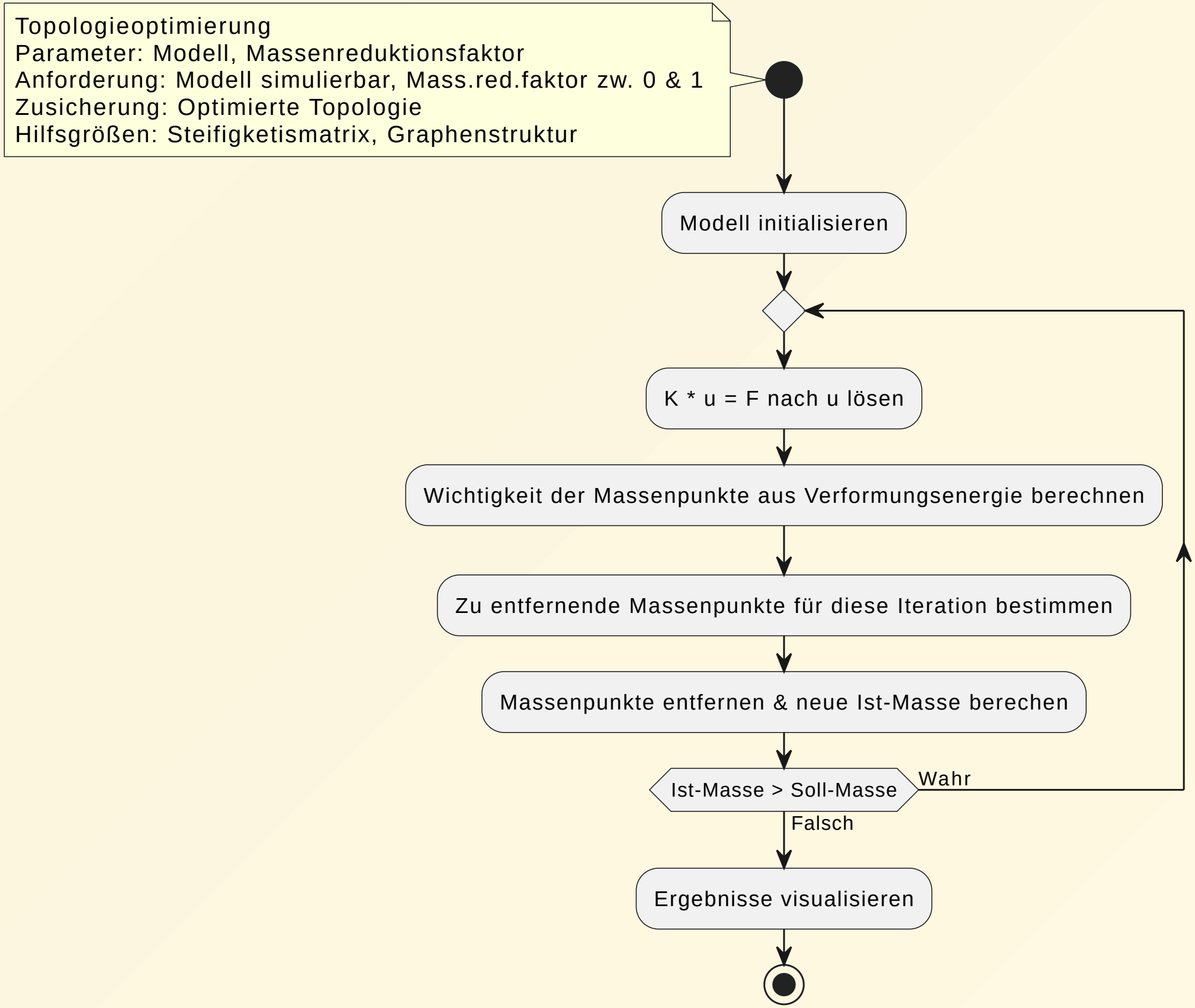
Hilfreiche Operationen von Graphen

- Bestimmen der Inzidenzmatrix → Verknüpfung Knoten & Kanten
- Finden von Pfaden → Struktur auf Stabilität prüfen

Ist es sinnvoll nur eine Datenstruktur zu verwenden oder kann es Vorteile haben mehrere zu kombinieren?

Zusammenfassung

Zusammenfassung



Annahmen & Hilfreiche Ansätze

Annahmen

- Koordinatenursprung befindet sich links oben
  - x-Achse zeigt nach rechts, z-Achse zeigt nach unten
- Horizontale & vertikale Federn haben $k = 1\,\mathrm{N/m}$
- Diagonale Federn haben $k = \frac{1}{\sqrt{2}}\,\mathrm{N/m}$
- Alle Massepunkte haben die gleiche Masse $m = 1\,\mathrm{kg}$

Hilfreiche Ansätze für die Implementierung

- Symmetrische Strukturen können oft nur halb modelliert werden → schneller → Randbedingungen an Symmetrielinie beachten
- Optimierer wählt anhand der Verformungsenergie Punkte zum Entfernen aus → bevor sie entfernt werden können muss überprüft werden:
  - ob die Struktur noch weiterhin aus einem Stück besteht
  - ob alle Lasten noch auf die Lager übertragen werden können
- Sinnvollerweise wird in den ersten Iterationen etwas weniger Material entfernt → Struktur kann sich "setzen"

Anforderungen, Erweiterungen & Abgabe

Minimalanforderungen

- Softwareprojekt in Python mit Objektorientierung
- Versionskontrolle über git & GitHub
- Anwendung mit Web-UI:
  - Möglichkeit zur Durchführung der Topologieoptimierung für (quasi) beliebige 2D-Strukturen
- Software-Dokumentation:
  - `requirements.txt`-Datei mit allen packages
  - `README.md` im Repository mit Anleitung zur Installation und Ausführung
- Projekt-Dokumentation - Zwei Möglichkeiten
  - `README.md` ergänzen um umgesetzten Erweiterungen, UML-Diagrammen der Softwarestruktur, Quellen zu verwendeten Inhalten etc.
  - *ODER*
  - Kurzer Bericht als pdf-Dokument mit selbem Inhalt

Minimalanforderungen 1/2

- Eine `Python`-Anwendung mit Web-UI (`streamlit`) soll entwickelt werden
- Darin kann die Topologieoptimierung beliebiger 2D-Strukturen mit unseren Einschränkungen durchgeführt werden
- Die Ausgangsstruktur soll definiert werden können:
  - Ursprünglicher Bauraum als Rechteck mit Breite & Höhe
  - Randbedingungen (Loslager, Festlager) an Massepunkten
  - Externe Kräfte an Massepunkten
- Visualisierung der Struktur vor, während & nach der Optimierung inkl. der Verformung
- Die Struktur inkl. Randbedingungen, etc. kann zu jedem Zeitpunkt gespeichert und wieder geladen werden → Fortsetzung der Optimierung möglich
- Lösung des Problems erfolgt in gedanklicher Anlehnung an die Finite Elemente Methode (FEM) → so wie oben vereinfacht beschrieben
- Es muss verifiziert werden, die Struktur nicht "auseinander fällt" durch die Optimierung

Minimalanforderungen 2/2

- Testen der Implementierung am Beispiel des Messerschmitt–Bölkow–Blohm (MBB) Balken → in Dokumentation zeigen
- Die optimierte Geometrie kann als Bild heruntergeladen werden
- Die Anwendung soll mit `streamlit` deployed werden

Mögliche Erweiterungen

- Hochladen eines Bildes (z.B. `png`, `jpg`, etc.) → Schwarze Pixel = Material, weiße Pixel = kein Material
- Zeichnen der Struktur im UI (z.B. mit `streamlit-drawable-canvas` o.ä.)
- Definition einer Auszeichnungssprache um Strukturen & Randbedingungen zu beschreiben → Kann heruntergeladen und ggf. hochgeladen werden
- Visualisierung des Optimierungskriteriums als Heatmap → welche Massenpunkte sollten (nicht) entfernt werden
- Unterschiedliche Algorithmen zur Topologieoptimierung implementieren → Vergleich der Ergebnisse
- Erweiterung auf 3D-Strukturen → quaderförmiger Bauraum
- Animation als Video/gif speichern
- Export als CAD-Datei (z.B. `stl`, etc.) für den 3D-Druck → mittels Umwandlung in Vektorgrafik & `OpenSCAD`
- Optimieren nach anderen Kriterien

Mögliche Erweiterungen

- Alternative Ansätze im UI
- Erstellen eines finalen Reports mit resultierender Masse, Steifigkeit, max. Verformung etc. der optimierten Struktur
- Visualisierung von interessanten Größen über die Iterationen
- Visualisierung von Lastpfaden
- Berücksichtigung von Fertigungsbeschränkungen → Featuregrößen, etc.
- ⚠ Komplexe Erweiterung: Dynamik berücksichtigen → Eigenfrequenzen als Optimierungskriterium einbauen

Abgabe

- Abgabetermin: 27.02.2026 um 23:55 Uhr
- Alle Gruppenmitglieder müssen den Link zum Repository auf Sakai abgeben
- Der Beitrag der einzelnen Mitglieder wird anhand ihrer Commits beurteilt → jedes Mitglied soll aktiv und inhaltvoll am Projekt mitarbeiten
- Es wird der Stand des letzten Commits im `main` bzw. `master`-Branch, der vor der Deadline stattgefunden hat, bewertet

Alternatives Abschlussprojekt

- Falls die Aufgabenstellung einer Gruppe nicht zusagt, kann ein alternatives Abschlussprojekt mit dem jeweiligen Vortragenden vereinbart werden
- In der ersten Hälfte des letzten Termins der LV muss diese Aufgabenstellung mit dem Vortragenden vereinbart werden → anschließend wird sie mit dem Vortragenden der anderen Gruppe abgestimmt

Was muss vereinbart werden?

- Idee des eigenen Projektes
- Minimalanforderungen → Programmiersprache, Versionsverwaltung, Dokumentation, etc. bleibt immer gleich
- Mögliche Erweiterungen
- Diese sind dann verbindlich für die jeweilige Gruppe → es kann aber immer auf das allgemeine Projekt zurück gewechselt werden