Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Max Stegmann

Entwicklung eines Matlab-Programms zur Berechnung und Analyse von ebenen starren Fachwerken

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and Aeronautical Engineering

Max Stegmann

Entwicklung eines Matlab-Programms zur Berechnung und Analyse von ebenen starren Fachwerken

Bachelorarbeit eingereicht bei Prof. Dr. Sven Füser

im Studiengang Flugzeugbau
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr. Sven Füser
Zweitprüfer: Prof. Dr. Gunnar Simon Gäbel

Abgabedatum: 12.07.2021

Zusammenfassung

Max Stegmann

Thema der Studienarbeit

Programmierung eines textbasierten Systems zur Berechnung von ebenen starren und statisch bestimmt gelagerten Fachwerken in MATLAB.

Stichworte

Programm, System, Matlab, Skript, Fachwerk, Berechnung, Stabkraft, Lagerreaktion, Engine, Plausibilitätsprüfung, Textdatei, Matrizen, Knotenpunktverfahren, Kräftegleichgewicht, Variablen, Gleichungssystem

Kurzzusammenfassung

Die Bachelorarbeit umfasst die Entwicklung eins textbasierten geschlossenen Systems zur Berechnung von Fachwerken im numerischen Programm Matlab. Das Programm berechnet systematisch und eigenständig die Stabkräfte und Lagerreaktionen von ebenen, starren und statisch bestimmten Fachwerken. Dazu wird ein lineares Gleichungssystem nach dem Knotenpunktverfahren mit Matrizen erstellt und aufgelöst. Das Programm erstellt zudem eine grafische Ausgabe des Fachwerks. Die geometrischen Informationen des Fachwerks werden über Befehle eines Befehlssatzes in einer ASCII-Textdatei eingegeben und durch eine Plausibilitätsprüfung auf Fehler untersucht.

Max Stegmann

Title of the paper

Programming of a text-based system for the calculation of plane rigid and statically defined trusses in MATLAB.

Keywords

Program, system, Matlab, script, truss, calculation, rod force, bearing reaction, engine, plausibility check, text file, matrices, method of nodes, force equilibrium, variables, system of equations

Abstract

The bachelor thesis comprises the development of a text-based closed system for the calculation of trusses in the numerical programme Matlab. The programme systematically and independently calculates the rod forces and bearing reactions of plane, rigid and statically defined trusses. For this purpose, a linear system of equations is created and solved using the method of nodes with matrices. The programme also creates a graphical output of the truss. The geometric information of the truss is entered via commands of a command set in an ASCII text file and checked for faults by a plausibility check.

Inhaltsverzeichnis

Abbildungsverzeichnis V

Tabellenverzeichnis VI

Verwendete Symbole VII

1 Aufgabenstellung 1

2 Motivation 2

3 Grundlagen 3

3.1 Mechanische Grundlagen . . . 3

3.1.1 Ideale Fachwerke . . . 3

3.1.2 Knotengleichgewicht . . . 6

3.1.3 Statische Bestimmtheit . . . 8

3.1.4 Assemblierung des Gleichungssystems . . . 9

3.2 Finite-Elemente-Methode . . . 11

3.3 Matlab . . . 12

4 Programmtechnische Umsetzung 14

4.1 Programmaufbau . . . 14

4.1.1 Variablen . . . 15

4.1.2 Übersicht . . . 18

4.1.3 Eingabedatei und Befehlssatz . . . 21

4.2 Programmablauf . . . 23

4.2.1 Hauptprogramm . . . 24

4.2.2 Eingabedatei öffnen und lesen . . . 26

4.2.3 Ausgabedatei erstellen . . . 31

4.2.4 Fehleranalyse der Eingabebefehle . . . 33

4.2.5 Übergabe der Fachwerksinformationen in numerische Matrizen . . 53

4.2.6 Bildliche Ausgabe des Fachwerks 58
4.2.7 Aufstellen und lösen des Gleichungssystems 64
4.2.8 Ausgabedatei beschreiben 67

5 Beispiel **72**

6 Ausblick **82**

Literatur **84**

A Matlab Skript **85**

B Kurzspezifikation des Befehlssatzes **134**

C Selbstständigkeitserklärung **135**

Abbildungsverzeichnis

3.1 Wirkungsrichtung der Stabkraft an einem starren Stab im lokalen und globalen Koordinatensystem . . . 5
3.2 Freigeschnittenes Lager und freigeschnittener Stab mit eingezeichneten Lagerreaktionen und Stabkräften . . . 6
3.3 Logo der Software Matlab und des Unternehmens MathWorks [Mat21] . . 13

4.1 Übersicht des Programmaufbaus und Programmablaufs mit den wichtigsten Variablen . . . 20
4.2 Ausrichtung des Lagers bei Angabe mit dem Winkel zur horizontalen Achse 23
4.3 Darstellung der Knoten, Stäbe, Loslager und äußeren Kräfte im Diagramm 58
4.4 Geometrische Definition der Eckpunkt eines Dreiecks, durch das ein Lager dargestellt wird . . . 62

5.1 Darstellung des Beispiel-Fachwerks aus Abbildung 6.12 [Gro+19] 72
5.2 Element-Variablen nach dem Einlesen der Eingabebefehle . . . 74
5.3 Element-Variablen nach der Fehleranalyse . . . 75
5.4 Numerische Element-Matrizen nach der Übergabe der Eingabeargumente . 76
5.5 Bildliche Ausgabe des Beispiels . . . 77
5.6 Gleichungssystem mit assemblierter Koeffizientenmatix und äußeren Kraftvektor . . . 78
5.7 Dialogbox informiert über das Abschließen des Programms . . . 78
5.8 Seite eins der generierten Ausgabetextdatei aus dem eingegebenen Befehlsdeck durch das Programm . . . 80
5.9 Seite zwei der generierten Ausgabetextdatei aus dem eingegebenen Befehlsdeck durch das Programm . . . 81

Tabellenverzeichnis

4.1 Übersicht der Variablen im globalen Workspace 17
4.2 Befehlssatz zur Definition der Fachwerkselemente in der Eingabedatei . . . 22
4.3 Zellen der Element-Variablen N,D,E und F 26
4.4 Spalten der Element-Matrizen `Nodes`, `Rods`, `Support` und `Forces` . 53

Verwendete Symbole

$\underline{a}$	Kraftvektor mit Stabkräften und Lagerreaktionen
$\underline{e}$	Einheitsvektor
F	äußere eingeprägte Kraft
$\underline{f}$	eingeprägter Kraftvektor
j	Grad der statischen Bestimmtheit
$\underline{\underline{K}}$	Koeffizientenmatrix
M_{Res}	resultierende Moment
n	Anzahl der Knoten
R	Stabkraft
r	Anzahl der Stäbe
$\underline{Res}$	resultierende Kraftvektor
S	Lagerreaktion
s	Anzahl der Lagerreaktionen
y	y-Koordinate \y-Richtung
x	x-Koordinate \x-Richtung

1 Aufgabenstellung

Es soll ein textbasiertes geschlossenes System zur Berechnung von Fachwerken im numerisch orientierten Programm Matlab programmiert werden. Das System soll die Stabkräfte und Lagerreaktionen eines ebenen, starren und statisch bestimmten Fachwerks systematisch berechnen können.
Hierzu muss zuerst eine ASCII-Textdatei eingelesen werden, die alle geometrischen Informationen und Größen des Fachwerks in einer Kurzspezifikation enthält. Das Eingabedeck muss auf dem ANSYS-APDL-Befehlssatz beruhen und das Programm soll die Informationen zweckmäßig einlesen und in Variablen übergeben.
Das System stellt die Gleichungen nach dem Knotenpunktverfahren in Matrizendarstellung auf und fügt die Randbedingungen in Form der Auflagerkräfte und äußeren Kräfte ein. Das Gesamtsystem soll numerisch gelöst und die Daten des Eingabedecks sowie das Ergebnis der Stabkräfte und Lagerreaktionen in einer ASCII-Textdatei tabellarisch ausgeben werden.
Dabei soll das Programm funktional arbeiten. Der Programmablauf soll angelehnt an die Vorgehensweise eines Finite-Elemente-Methode-Programm (FEM-Programm) sein und möglichst wie eine eigenständige Engine ohne Steuerbefehle des Benutzers arbeiten. Das Fachwerk soll zusätzlich grafisch zur optischen Eingabekontrolle und Anschauung in einer einfachen Abbildung ausgegeben werden und alle wichtigen Elemente enthalten. Um Fehler im Programmablauf zu vermeiden und erkennen zu können, erfolgt eine Plausibilitätsprüfung der Eingabedatei, die den Benutzer auf Fehler hinweist.

2 Motivation

Das Programm soll in der Lehre der HAW Hamburg verwendet werden. Dort soll es unter anderem dazu dienen, Studierenden eine Verbindung zwischen dem Programmieren von kleinen Programmabläufen und der ingeneursmäßigen Anwendung von Statikprogrammen zu zeigen. Im Modul Datenverarbeitung fällt vielen Studierenden die Umsetzung ihres Konzepts in eine Programmiersprache noch schwer, da ihnen die Vorstellungskraft für den Programmablauf und die Programmiersprache fehlt. Die Berechnung von Fachwerken beruht auf einfacher technischer Mechanik, mit der sich alle Studierenden schon befasst haben. Unter der Verwendung von Matlab stellt das Programm eine ingenieurmäßige Umsetzung dar, wie eine Aufgabenstellung mittels effektiver Algorithmen in einer höheren Programmiersprache gelöst werden kann. Es motiviert Studierende, sich näher mit dem Programmieren zu beschäftigen und Abläufe zu automatisieren.
Außerdem zeigt es dabei anschaulich, welche einzelnen Schritte eines Mechanik-Programms notwendig sind, um mechanische statische Systeme zu berechnen. Das Programm arbeitet wie eine Engine und generiert aus einer Eingabedatei eine Darstellung des Fachwerks und die Lösung in zwei Ausgabedateien. Dem Benutzer stehen verschiedene Eingabebefehle zur Verfügung, um ein individuelles Fachwerk mit Knotenlasten und Lagern erstellen und berechnen lassen zu können.
Das System ist modular aufgebaut und stellt eine Basis zur Berechnung von Fachwerken dar. Es besteht aus einzelnen Bausteinen, die ersetzt oder verändert werden können, ohne sich mit dem gesamten Programm beschäftigen zu müssen. So kann es einfach weiterentwickelt und beispielsweise die Berechnung der Verformung der Stäbe implementiert werden.

3 Grundlagen

3.1 Mechanische Grundlagen

Ein Fachwerk (auch Stabwerk genannt) ist ein Tragwerk, welches nur aus Stäben besteht, die in sogenannten Knoten oder Gelenken miteinander verbunden sind.
Mit dem Programm sollen ebene, starre und statisch bestimmte Fachwerke automatisiert berechnet werden können. Hier werden die dazu grundlegenden mechanischen Zusammenhänge erklärt. Zudem werden damit bereits einzelne Berechnungsschritte der späteren Programmierung deutlich.

3.1.1 Ideale Fachwerke

Bei der Berechnung der Fachwerke müssen einige Annahmen und Voraussetzungen getroffen werden, damit diese allein mit Hilfe der Gleichgewichtsbedingungen und ohne Berücksichtigung der Verformung berechnet werden können. Diese Fachwerke heißen ideale Fachwerke, S.174f [Sza75], wenn sie folgende Voraussetzungen erfüllen:

- Alle Stäbe sind an ihrem Ende mit frei drehbaren und reibungsfreien Gelenken verbunden.
- Das Gewicht der Stäbe wird vernachlässigt. Die Stäbe gelten als gewichtslos.
- Alle äußere Belastungen sind Einzelkräfte und greifen nur an den Knotenpunkten an.

Durch diese Annahmen folgt, dass alle Stäbe nur auf Zug und Druck belastet werden und die Stabkräfte R entlang der Längsachse der Stäbe wirkt. Die Wirkungslinie der Stabnormalkräfte der in einem Knoten verbundenen Stäbe verläuft durch einen Punkt. Bei idealen Fachwerken sind die Knoten eckweiche Gelenke, die die Stabnormalkräfte von Stab zu Stab übertragen.

An den Lagern werden die Kräfte aus dem Fachwerk hinaus an die Umgebung übertragen. Ein Lager wird über seine Wertigkeit definiert, also die Anzahl der übertragbaren Kraftkomponenten. Hier sollen ausschließlich einwertige Loslager verwendet werden, die die Normalkraftkomponente S senkrecht zum Boden des Lagers übertragen kann. Die Kraftkomponente wird Lagerreaktion genannt. Externe eingeprägte Kräfte F greifen an den Knoten des Fachwerks an. Ein Fachwerk besteht hier aus den drei verschiedenen Tragwerkselementen und den externen eingeprägten Kräften:

- gerade Stäbe
- reibungsfreie Gelenke
- einwertige Loslager
- externe eingeprägte Kräfte

Gleiche Elemente werden mit arabischen Zahlen $1, 2, 3, ...$ nummeriert, um sie untereinander unterscheiden zu können. Für eine eindeutige Differenzierung der Lager und Stäbe im Fachwerksdiagramm werden die Lager mit römischen Zahlen $\mathrm{I}, \mathrm{II}, \mathrm{III}, ...$ nummeriert. Außerdem wird hier unterschieden zwischen dem lokalen Koordinatensystem der Stäbe und der Loslager mit dem globalen Koordinatensystem des Fachwerks:

- Im globalen benutzten Koordinatensystem des Fachwerks werden wie üblich die tiefgestellten Indizes x und y benutzt (z.B. externe Kraftkomponente in x-Richtung F_x). Vektoren und Matrizen im globalen Koordinatensystem erhalten keine Indizes (z.B. die Koeffizientenmatrix $\underline{\underline{\mathrm{K}}}$).
- Für die lokal verwendeten Koordinatensysteme der Elemente wird ein e für Element hochgestellt (z.B. Stabkraft R_2^e). Das ist ausreichend, weil die Kraftkomponenten der Stäbe und Loslager nur eine Wirkungsrichtung haben.

Alle Vektoren werden hier einfach unterstrichen und alle Matrizen werden doppelt unterstrichen, um eine einfache Unterscheidung zu den Laufindizes und anderen Variablen zu ermöglichen. Die Stabkräfte wirken nur entlang der Stabnormalrichtung und die Loslager nur senkrecht zum Lagerboden. Sie besitzen also nur eine Vektorrichtung und werden deswegen nicht unterstrichen. Die Wirkungsrichtung der Stabkraft an einem starren Stab ist in Abbildung 3.1 zu sehen. Sie können zu den Skalaren unbekannten Kraftgrößen $\boldsymbol{R}$ und $\boldsymbol{S}$ durch das hochgestellte $\boldsymbol{e}$ unterschieden werden.

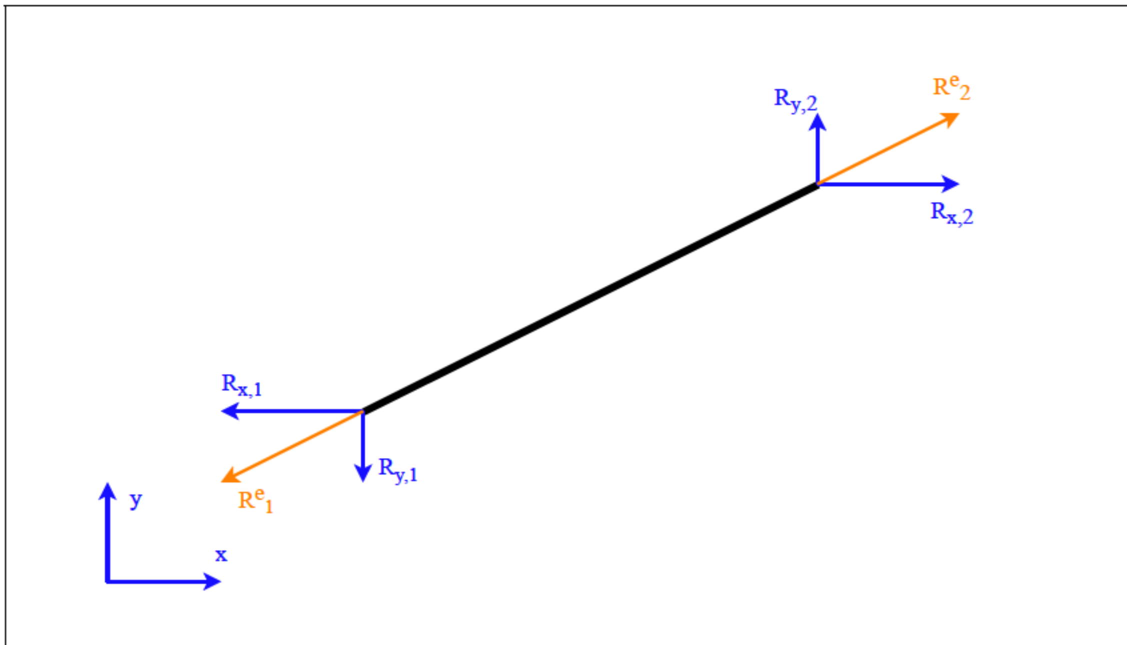


Abbildung 3.1: Wirkungsrichtung der Stabkraft an einem starren Stab im lokalen und globalen Koordinatensystem

3.1.2 Knotengleichgewicht

Für eine automatisierte Ermittlung der Stabkräfte und Lagerreaktionen wird für die Analyse des statisch bestimmten und idealen Fachwerks das Knotenpunktverfahren oder Rundschnittverfahren genannt, S.177 [Sza75], angewendet. Dazu werden die Knoten freigeschnitten, also die Stäbe und Lager von den Knoten getrennt. Nun können die von den Stäben und Lagern wirkenden Kräfte auf das Gelenk eingezeichnet werden. Die Stabkräfte werden dabei als Zugkräfte eingetragen, so dass die wirkende Stabkraftvektoren am freigeschnittenen Knoten vom Gelenk wegzeigen. Da die Kraftgrößen unbekannt sind, wird für jeden Stab $\boldsymbol{p}$ die Stabkraft mit der gehörigen skalaren Unbekannte $\boldsymbol{R_p}$, für jedes Lager $\boldsymbol{k}$ die Lagerreaktionen mit $\boldsymbol{S_k}$ und für jede eingeprägte Kraft $\boldsymbol{l}$ die Kraft mit $\boldsymbol{F_l}$ eingetragen. Ein freigeschnittenes Lager und ein freigeschnittener Stab sind in Abbildung 3.2 dargestellt.

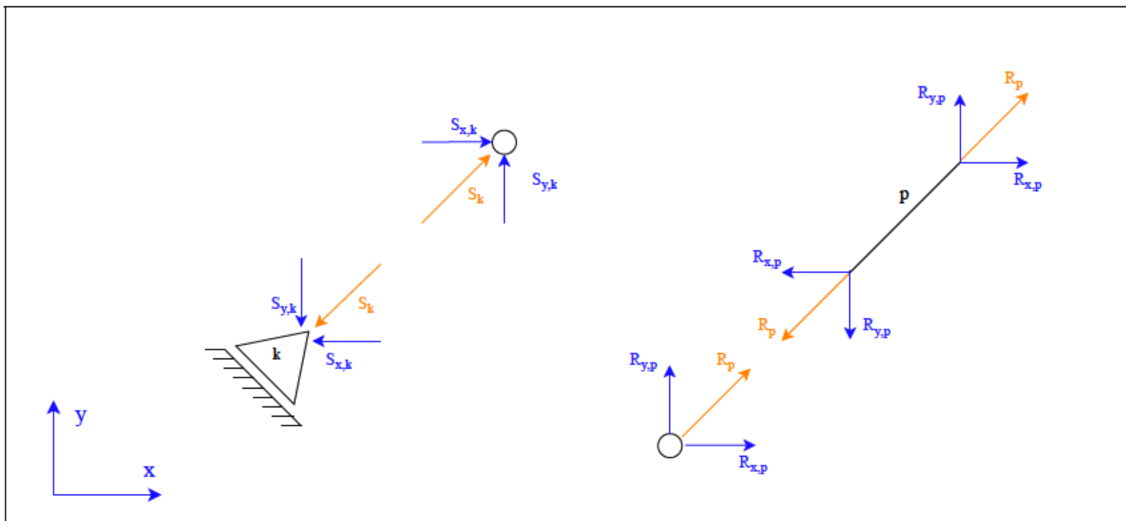


Abbildung 3.2: Freigeschnittenes Lager und freigeschnittener Stab mit eingezeichneten Lagerreaktionen und Stabkräften

Für jedes Gelenk können je zwei Komponentenbedingungen in zwei unterschiedliche Richtungen aufgestellt werden, um die skalaren Unbekannten zu ermitteln. Ein freigeschnittenes Loslager und freigeschnittener Stab

Durch die Zwangsbedingungen der Lager und der einzelnen Fachwerkselemente entstehen Zwangskräfte (Reaktionskräfte). Diese wirken den eingeprägten äußeren Kräften entgegen und halten das gesamte Fachwerkssystem und seine Elemente in Ruhe (vgl. Kap. 3.1.3).

Ein unbewegliches Element oder System befindet sich im mechanischem Gleichgewicht und erfährt keine Beschleunigung. Es herrscht das Kräftegleichgewicht, S.179 [Lor24]. Am gesamten System und folglich auch an jedem freigeschnittenen Element muss die Resultierende aller Kräfte $\underline{Res}$ und die Resultierende aller Momente M_{Res} um jeden beliebigen Punkt null sein.

$$\underline{Res} = \underline{0} \tag{3.1}$$

$$M_{Res} = 0 \tag{3.2}$$

Ist ein Stab im mechanischen Gleichgewicht, dann herrscht auch das Kräftegleichgewicht:

$$R^e_{p_1} + R^e_{p_2} = 0 \tag{3.3}$$

Unter der Annahme, dass das gesamte Fachwerk unbeweglich und statisch bestimmt ist, herrscht an jedem Knoten des Fachwerks das mechanische Gleichgewicht. Es wird hier ein Kräftegleichgewicht in x-Richtung und y-Richtung am Knoten aufgestellt. An den Knoten können Stabkräfte, Lagerreaktionen und äußere eingeprägte Kräfte angreifen. Die an dem Knoten in die x- und y-Richtung wirkenden Kräfte und Reaktionen werden aufsummiert, also folgt mit 3.1 für einen Knoten des Fachwerks:

$$\sum R_{x,p} + \sum S_{x,k} + \sum F_{x,l} = 0 \tag{3.4}$$

$$\sum R_{y,p} + \sum S_{y,k} + \sum F_{y,l} = 0 \tag{3.5}$$

Für jeden Knoten können so zwei Kräftegleichgewichte in die Koordinatenrichtungen aufgestellt werden, um die skalaren Unbekannten zu lösen. Ergibt die skalare Unbekannte einen positiven Wert, dann ist die zugehörige Stabkraft eine Zugkraft und die Kraftwirkungsrichtung ist wie oben angenommen. Bei einem negativen Wert handelt es sich um eine Druckkraft und die Wirkungsrichtung der Kraft ist umgedreht.

3.1.3 Statische Bestimmtheit

Ein Tragwerk heißt statisch bestimmt, wenn alle Stab- und Lagerkräfte aus den Gleichgewichtsbedingungen der Knoten ermittelt werden können, S.163 [Föp12]. Bei einem ebenen Fachwerk mit n Knoten (Nodes), r Stäben (Rods) und s Lagerreaktionen (Support reactions) erhält man $2 \cdot n$ Gleichungen, um die $r + s$ Unbekannten zu lösen. Der Grad der statischen (Un-)Bestimmtheit j ergibt sich damit für ein ebenes Fachwerk zu:

$$j = r + s - 2 \cdot n \tag{3.6}$$

Ist das Fachwerk statisch bestimmt, lassen sich alle Auflagerreaktionen und Stabkräfte mit den Gleichgewichtsbedingungen an den Knoten lösen. Damit ist die notwendige Bedingung geschaffen um die Stabkräfte und Lagerreaktionen berechnen zu können:

$$j = 0 \qquad \text{notwendige Bedingung für statische Bestimmtheit} \tag{3.7}$$

Ein Fachwerk ist kinematisch bestimmt, wenn alle Knoten unbeweglich sind. Kinematisch unbestimmte Fachwerke können endliche oder infinitesimal kleine Bewegungen ausführen und werden ausgeschlossen. Damit ist die Bedingung 3.7 nicht hinreichend für statische Bestimmtheit und das Fachwerk muss zusätzlich unbeweglich, also kinematisch bestimmt sein.
Ist das Fachwerk statisch unterbestimmt, so ist es auch beweglich:

$$j < 0 \qquad \text{j} - \text{fach statisch unterbestimmt} \tag{3.8}$$

Es wirkt also mindestens einer Bewegungsmöglichkeit keine Lager- oder Verbindungsreaktion entgegen. Das Fachwerk ist j-fach statisch unterbestimmt. Bei einem statisch überbestimmten Fachwerk existieren mehr Lagerreaktionen als Bewegungsmöglichkeiten:

$$j > 0 \qquad \text{j} - \text{fach statisch überbestimmt} \tag{3.9}$$

Um die Stab- und Lagerreaktionen zu bestimmen, müssen die Verformungen der Elemente berücksichtigt werden. Durch kleine Lagerabweichungen oder Wärmedehnung entstehen hier zudem Eigenspannungen.

3.1.4 Assemblierung des Gleichungssystems

Alle Kräftegleichgewichte an den Knoten können zu einem gesamten Fachwerksgleichungssystem assembliert werden. Dafür werden die Kräftegleichgewichte in x- und y-Richtung an den einzelnen Knoten mit Hilfe des zugehörigen Einheitsvektors $\underline{e}$ der Stabkräfte und der Lagerreaktionen formuliert:

$$\underline{e} = \begin{pmatrix} e_x \\ e_y \end{pmatrix} \tag{3.10}$$

Der Einheitsvektor spiegelt die Komponente der Kraft in die x- und y-Richtung wieder. Die eingeprägten Kräfte werden auf die andere Seite der Gleichung subtrahiert. Damit wird mit Gleichung 3.10 aus Gleichung 3.4 und 3.5:

$$\sum e_{x,R_p} \cdot R_p + \sum e_{x,S_k} \cdot S_k = -\sum F_{x,l} \tag{3.11}$$

$$\sum e_{y,R_p} \cdot R_p + \sum e_{y,S_k} \cdot S_k = -\sum F_{y,l} \tag{3.12}$$

Nun kann der Einheitsvektor $\underline{e}$ aus Gleichung 3.11 und 3.12 der Stabkräfte und Lagerreaktionen durch eine entsprechende Kraft-Koeffizientenkomponente ersetzt werden. Für alle Stäbe bzw. Stabkräfte und alle Loslager bzw. Lagerreaktionen im Fachwerk werden die Kräftegleichgewichte nun mit Hilfe der Koeffizientenkomponente $k_{x,R_p}, k_{y,R_p}, k_{x,S_k}, k_{y,S_k}$ an den einzelnen Knoten formuliert. Greift eine Stabkraft oder Lagerreaktion nicht am Knoten an, so ist die Koeffizientenkomponente null, ansonsten entspricht sie dem Einheitsvektor.

$$\sum_p^r k_{x,R_p} \cdot R_p + \sum_k^s k_{x,S_k} \cdot S_k = -\sum F_{x,l} \tag{3.13}$$

$$\sum_p^r k_{y,R_p} \cdot R_p + \sum_k^s k_{y,S_k} * S_k = -\sum F_{y,l} \tag{3.14}$$

Insgesamt werden also pro Knoten und Gleichgewichtsrichtung eine Koeffizientenkomponente für jede Stabkraft und Lagerreaktion benötigt. Um die einzelnen Gleichgewichtsbedingungen zu einem gesamten Gleichungssystem zusammenzufassen, werden die Koeffizientenkomponenten in eine Koeffizientenmatrix $\underline{\underline{K}}$ assembliert. Einer Zeile der Matrix entspricht einem Kräftegleichgewicht in x- oder y-Richtung an einem Knoten. Eine Spalte bezieht sich auf die skalare unbekannte Kraftgröße eines Stabs oder eines Loslagers.

Die Koeffzientenmatrix ist bei n Knoten im Fachwerk also $(2n) \times (r + s)$ groß.

$$\underline{\underline{\text{K}}} = \begin{array}{c} \\ \text{I}_\text{x} \\ \text{I}_\text{y} \\ \text{II}_\text{x} \\ \text{II}_\text{y} \\ \vdots \\ k_x \\ k_y \end{array} \begin{array}{c} \begin{array}{ccccccc} S_1 & S_2 & \cdots & S_s & R_1 & R_2 & \cdots & R_r \end{array} \\ \begin{bmatrix} k_{I_x,S_1} & k_{I_x,S_2} & \cdots & k_{I_x,S_s} & k_{I_x,R_1} & k_{I_x,R_2} & \cdots & k_{I_x,R_r} \\ k_{I_y,S_1} & k_{I_y,S_2} & \cdots & k_{I_y,S_s} & k_{I_y,R_1} & k_{I_y,R_2} & \cdots & k_{I_y,R_r} \\ k_{II_x,S_1} & k_{II_x,S_2} & \cdots & k_{II_x,S_s} & k_{II_x,R_1} & k_{II_x,R_2} & \cdots & k_{II_x,R_r} \\ k_{II_y,S_1} & k_{II_y,S_2} & \cdots & k_{II_y,S_s} & k_{II_y,R_1} & k_{II_y,R_2} & \cdots & k_{II_y,R_r} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{k_x,S_1} & k_{k_x,S_2} & \cdots & k_{k_x,S_s} & k_{k_x,R_1} & k_{k_x,R_2} & \cdots & k_{k_x,R_r} \\ k_{k_y,S_1} & k_{k_y,S_2} & \cdots & k_{k_y,S_s} & k_{k_y,R_1} & k_{k_y,R_2} & \cdots & k_{k_y,R_r} \end{bmatrix} \end{array} \tag{3.15}$$

Die gesuchten skalaren Unbekannten, die Stabkräfte und Lagerreaktionen, werden im $(r + s) \times 1$ großem Kraftvektor $\underline{a}$, das für *answer* steht, aufgelistet:

$$\underline{a} = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_s \\ R_1 \\ R_2 \\ \vdots \\ R_r \end{pmatrix} \tag{3.16}$$

Die äußeren eingeprägten Kräfte stehen im eingeprägten Kraftvektor $\underline{f}$. Dieser ist $(2n)\times 1$ groß. Ein Zeile steht auch hier für eine Gleichgewichtsrichtung in x- und y-Richtung eines Knotens. Wenn an einem Knoten maximal eine externe Kraft in x- oder y-Richtung wirkt, ergibt sich der eingeprägte Kraftvektor zu:

$$\underline{f} = \begin{pmatrix} -F_{I_x} \\ -F_{I_y} \\ -F_{II_x} \\ -F_{II_y} \\ \vdots \\ -F_{k_x} \\ -F_{k_y} \end{pmatrix} \tag{3.17}$$

Das lineare Gleichungssystem des gesamten Fachwerks kann nun aufgestellt werden. Aus den Gleichgewichtsbedingungen der einzelnen Knoten aus Gleichung 3.13 und 3.14 ergibt sich mit der Koeffizientenmatrix 3.15, dem Kraftvektor 3.16 und dem eingeprägten Kraftvektor 3.17:

$$\underline{\underline{\mathrm{K}}} \cdot \underline{a} = \underline{f} \tag{3.18}$$

Gesucht werden die skalaren Unbekannten im Kraftvektor $\underline{a}$ (Gleichung 3.16). Sie beinhalten die skalaren unbekannten Stabkräfte und Lagerreaktionen im lokalen Koordinatensystem. Es wird nach dem Kraftvektor aufgelöst:

$$\underline{a} = \underline{\underline{\mathrm{K}}}^{-1} \cdot \underline{f} \tag{3.19}$$

Mit der Determinante der Koeffizientenmatrix lässt sich überprüfen, ob das Fachwerk statisch und kinematisch bestimmt ist. Das Fachwerk ist statisch und kinematisch bestimmt, wenn das Gleichungssystem eindeutig lösbar ist. Die Determinante der Koeffizientenmatrix muss ungleich null sein.

$$\det(\underline{\underline{\mathrm{K}}}) \neq 0 \tag{3.20}$$

3.2 Finite-Elemente-Methode

Das Berechnungsverfahren und der Aufbau orientieren sich an dem Verfahren eines Fenite-Elemente-Methode- Programms. Die Finite-Elemente-Methode (FEM) ist ein numerisches Verfahren, um Festigkeits- und Verformungsuntersuchungen von komplexen Systemen und Körpern durchzuführen [KW17]. Dabei wird das gegebene Bauteil oder System in einzelne Elemente unterteilt, die finiten Elemente. Diese Gebiete sind endlich klein ("finit"), die Größe bleibt also mathematisch noch relevant. Ein System oder Bauteil wird in ausreichend viele Elemente unterteilt, bis eine weitere Verfeinerung keinen Einfluss auf das Rechenergebnis hat. Bei Fachwerken ist die Unterteilung durch die Problemstellung bereits gegeben. Hier bilden die einzelnen Stäbe die Elemente des Systems und die Gelenke die Knoten, die Ecken der finiten Elemente. Damit ist eine exakte Berechnung jedes einzelnen Elementes und damit des gesamten Tragwerks möglich. Eine weitere Unterteilung bringt keinen Mehrwert, da die Stabkräfte entlang der Stäbe konstant sind.

Im Unterschied zu einem FEM-Programm, das vorrangig die Knotenverschiebungen berechnet, werden hier die Stabkräfte und Lagerreaktionen berechnet. Bei einem FEM-Programm ergeben sich die Stabkräfte später aus den Verschiebungen der Knoten. Dadurch muss hier keine lokale und globale Steifigkeitsmatrix aufgestellt werden. Hier entspricht die Steifigkeitsmatrix der Koeffizientenmatrix aus Gleichung 3.15.

Es wird für jedes Element und für jede gesuchte Unbekannte eine Ansatzfunktion gewählt, die das physikalische Verhalten des Bauteils ausreichend genau beschreibt. Dies geschieht für jedes Element im lokalen Koordinatensystem. Vom Übergang von einem Element in das benachbarte Element müssen Stetigkeitsbedingungen an den Knoten erfüllt werden. So werden die einzelnen Elementgleichungen im globalen Koordinatensystem assembliert zu einem linearen Gleichungssystem(vgl. 3.18). Hier entspricht dies dem Übergang vom lokalem Kräftegleichgewicht eines Stabes zum Kräftegleichgewicht an einem Knoten.

Nachdem ein gegebenes Problem diskretisiert ist und die Elementmatrizen aufgestellt sind, führt man vorgegebene Randbedingungen ein. Sie gelten (wirken) immer an den Knotenpunkten. Bei dem Fachwerk wird eine Neumann-Randbedingung in Form von einer zusätzlichen Kraft, der Lagerreaktion, eingefügt. Diese verhindert so die Verschiebung eines Knotens.

3.3 Matlab

Matlab [Mat21], das für *Matrix-Labor* steht, ist eine numerisch ausgelegte Multiparadigmen-Programmiersprache. Sie ist eine funktional, imperativ und prozedural aufgebaute Sprache und eignet sich besonders, um mathematische Operationen in der linearen Algebra durchzuführen, um Funktionen und Ergebnisse zu plotten und um Algorithmen zu implementieren. Da es eine effiziente Software bei der Lösung von mathematischen Problemen mittels numerischer Berechnungsmethoden und Matrizen ist, eignet sie sich besonders gut für den Bereich der Ingenieurswissenschaften und wird hier verwendet.

Die Software, dessen Logo in Abbildung 3.3 dargestellt wird, hat etwa fünf Millionen Nutzer [Mat21] und wird in der Industrie und an Hochschulen für Simulationen, Datenerfassung und Datenanalyse verwendet. Es kann durch verschiedene Toolboxen erweitert werden und bildet die Grundlage für Simulink, eine Software zur zeitgesteuerten Simulation, beispielsweise für der Regelungstechnik. Es ist eine proprietäre Software. Eine alternative frei zugängliche Software ist Octave [Eat88]. Die Sprachen sind größtenteils kompatibel.

Abbildung 3.3: Logo der Software Matlab und des Unternehmens MathWorks [Mat21]

Die Benutzeroberfläche von Matlab besteht mit den Standarteinstellungen aus fünf verschiedenen Bereichen:

- der Befehlsleiste mit den verschiedenen Reitern oben
- dem Current Directory auf der linken Seite
- dem Editor in der Mitte
- dem Command Window unten
- dem Workspace auf der rechten Seite

Im *Command Window* können direkt einzelne Anweisungen und Befehle erteilt und ausgeführt werden. Diese können im *Editor* als Quelltext zu einem Skript zusammengefasst werden. Im Command Window gibt Matlab die Antwort auf die ausgeführten Anweisungen und Befehle zurück, wenn eine Ausgabe nicht durch ein Semikolon unterdrückt wurde. Über die Befehlsleiste kann im Reiter Editor das Skript ausgeführt und weitere Einstellungen und Funktionen getätigt werden. Im *Workspace* werden die Variablen angezeigt, die im Command Window oder im Editor definiert wurden. Dies beinhaltet auch multidimensionale Matrizen oder Diagramme. Das *Current Directory* zeigt das aktuelle Verzeichnis.

4 Programmtechnische Umsetzung

4.1 Programmaufbau

Das Programm wird mit der Software Matlab erstellt. Es wird ausschließlich mit den Anweisungen und Befehlen der Basisversion von Matlab ohne zusätzliche Toolboxen gearbeitet, sodass mit dem Zusatzpaket `octave-forge` und anderen Ersatzfunktionen in Octave eine fast vollständige Kompatibilität erreicht werden kann. Da Matlab auf den englischen Programmiersprachen C++ und Fortran basiert, werden auch alle folgenden Kommentare, Funktionen und Variablen im Programm auf Englisch verfasst. Die Kommentare sollen helfen, die einzelnen Skripte, Funktionen, Schleifen und Befehle genau zu verstehen.
Matlab besitzt eine prozedurale Programmiersprache und so wird das Programm modular aufgebaut. Dazu wird es nach dem Top-Down Prinzip in einzelne Skripte aufgeteilt, die von einem Haupt-, bzw. Rootskript nacheinander aufgerufen werden. Die einzelnen Unterskripte arbeiten autonom und erfüllen jeweils eine wichtige Funktion des gesamten Programms. Die Unterskripte können wiederum Funktionen besitzen, die nacheinander ausgeführt werden. Dabei wird auch darauf geachtet, die Menge der Unterskripte und Funktionen nicht zu groß werden zu lassen, damit der Aufbau übersichtlich bleibt. Außerdem werden tieferliegende Ebenen, die durch Schleifen und Funktionen entstehen, um einen Tab-Sprung eingerückt. Dies erleichtert im Skript die Identifizierung von den Befehlen und Abschnitten, die zusammengehören. Kommentarzeilen oder Anmerkungen werden in Matlab durch ein Prozent-Zeichen `%` vor dem entsprechenden Kommentar markiert. Zudem werden in den Standarteinstellungen von Matlab Kommentarzeilen grün, Schleifenbefehle blau und Zeichenketten magenta hervorgehoben.
Durch den modularen Aufbau ist das gesamte Programm übersichtlicher. Ein Teil erfüllt eine bestimmte Funktion und so können einzelne Teile einfach ausgetauscht oder ergänzt werden. Dabei ist es dann nicht zwingend notwendig, das gesamte Programm bzw. Skript zu verstehen.

4.1.1 Variablen

Variablen, was für veränderliche Größen steht, speichern Formeln, Daten und Ausdrücke. Variablen werden durch Namen gekennzeichnet, die in Matlab fast frei vergeben werden können. Der Name muss mit einem Buchstaben anfangen und darf von weiteren Buchstaben, Zahlen und Unterstrichen gefolgt werden. Matlab unterschiedet Groß- und Kleinbuchstaben, es ist case sensitive. Es muss darauf geachtet werden kein Matlab Argument oder Befehl wie `end` oder `path` zu verwenden. Hier wird eine Sprechende Objektbezeichnung für die Benennung von Variablen verwendet. Der Name soll selbsterklärend sein und richtet sich nach ihrem Inhalt, ihrer Verwendung und zeigt die semantische Bedeutung auf. Besteht der Name aus mehreren Wörtern, dann werden die Wörter durch Unterstriche verbunden. Die gleiche Namenskonvention wird auch für die Benennung von Funktionen und Programmen verwendet.
Zugewiesen werden den Variablen Werte mittels eines Gleichheitszeichens. In Matlab müssen sie normalerweise nicht deklariert oder dimensioniert werden. Sie werden von Matlab automatisch definiert und im Workspace gespeichert. Zu beachten ist dabei, dass die einzelnen Skripte in Matlab den globalen Variablen-Workspace benutzen. Eine Variable aus dem ersten Skript ist so auch im nächsten aufgerufenen und ausgeführten Skript vorhanden. Funktionen haben dagegen einen lokalen Workspace und teilen ihre Variablen nicht. Es müssen die Ein- und Ausgabevariablen aus der Funktion definiert werden: `function [Ausgabevariable(n)]=Funktionsname(Eingabevariable(n))`.
Die Eingabevariable(n) stehen in runden Klammern hinter dem Namen der Funktion und die Ausgabevariable(n) in eckigen Klammern vor dem Gleichheitszeichen. Die Ein- und Ausgabevariablen müssen beim Aufrufen und bei der Definition der Funktion benannt werden. Hier werden die Namen der lokalen Variablen, die aus dem globalen Workspace ein- und ausgegeben werden, in den Funktionen nicht verändert. Eine Funktion steht immer am Ende eines Skriptes und wird über den Namen aufgerufen: `[Ausgabevariable(n)]=Funktionsname(Eingabevariable(n))`.
Die lokalen Workspaces sind nicht sichtbar und so kann eine Funktion zudem helfen, die Anzahl der sichtbaren Variablen zu verringern, um den globalen Workspace übersichtlicher zu gestalten und auf die wichtigsten Variablen zu beschränken.
Matlab besitzt verschiedene Arten von Variablen. Durch einige Befehle in Matlab entsteht ein bestimmter Typ von Variablen. Andere Variabletypen werden bewusst erzeugt.

Hier kommen folgende Typen vor:

- `double`
- `char-array`
- `string-array`
- `cell-array`

`Double` ist der standardmäßig numerische Datentyp und wird auch hier für Zahlen verwendet. Die Zahlen werden als Fließkommazahl mit 64-Bit gespeichert und bieten so ausreichend Genauigkeit für die Berechnungsverfahren. Zeichenfolgen werden in `char-array` und `string-array` gespeichert. Ein `char-array` besteht aus einer Folge von einzelnen Zeichen und wird mit einfachen Anführungszeichen definiert: `'Zeichenfolge'`. Der `string-array` enthält dagegen Textstücke. Er ist eine Abfolge von Zeichen und wird mit doppeltem Anführungszeichen erstellt: `SZeichenabfolge"`. Ein `cell-array` kann verschiedene Arten von Datentypen pro Zelle enthalten. Die einzelnen Zellen sind indizierte Datencontainer und enthalten einzelne, vektorielle oder in einer Matrix angeordnete Daten. Einzelne Zellen können mit geschweiften Klammern angesprochen werden, einzelne Datenbereiche aus Zellen werden mit runden Klammern ausgewählt. Die wichtigsten Variablen des Programms, die auch im globalem Workspace auftauchen, sind in der folgenden Tabelle 4.1 zu sehen. Dort ist der Typ, die Größe und der Inhalt der einzelnen Variablen aufgelistet. Darüber hinaus werden in einzelnen Funktionen weitere Variablen definiert um Informationen zwischenzuspeichern und Berechnungen durchführen zu können.

Tabelle 4.1: Übersicht der Variablen im globalen Workspace

Name	Typ	Größe	unterer Ebene	Inhalt
input_name	char array	Zeilenvektor	n/a	Name der Eingabetextdatei
output_name	char array	Zeilenvektor	n/a	Name der Ausgabetextdatei
folder_path	char array	Zeilenvektor	n/a	Ordnerpfad der Eingabedatei
Input	double	einzelner Wert	n/a	Dateikennung der Eingabedatei
Output	double	einzelner Wert	n/a	Dateikennung der Ausgabedatei
N	cell array	3 Zellen	string array Spaltenvektoren	Knotenargumente aus Eingabebefehlen
E	cell array	3 Zellen	string array Spaltenvektoren	Stabargumente aus Eingabebefehlen
D	cell array	2 Zellen	string array Spaltenvektoren	Lagerargumente aus Eingabebefehlen
F	cell array	3 Zellen	string array Spaltenvektoren	Kraftargumente aus Eingabebefehlen
input_text_lines	cell array	Spaltenvektor	string array	Text der Eingabedatei
Nodes	double	Matrix mit 3 Spalten	n/a	Knoteninformationen
Rods	double	Matrix mit 7 Spalten	n/a	Stabinformationen
Supports	double	Matrix mit 4 Spalten	n/a	Lagerinformationen
Forces	double	Matrix mit 4 Spalten	n/a	Informationen
K	double	Matrix	n/a	Koeffizientenmatrix des Fachwerks
f	double	Spaltenvektor	n/a	äußere Kraftvektor
a	double	Spaltenvektor	n/a	Lösung (Stabkräfte und Lagerreaktionen)
Truss	figure	n/a	n/a	Bildliche Ausgabe
faults_count	double	einzelner Wert	n/a	Anzahl der korregierbaren Fehler
major_faults_count	double	einzelner Wert	n/a	Anzahl der schweren Fehler
faults_line	double	einzelner Wert	n/a	Zeile eines Fehlers in der Eingabedatei
j	double	einzelner Wert	n/a	Grad der statischen Bestimmtheit
n; m; o	double	einzelner Wert	n/a	Laufindizes (for-Schleife)

Die Eingabe und Ausgabedateien werden durch den Befehl `fopen()` in den Variablen `Input` und `Output` geöffnet. Den Variablen wird von Matlab eine Dateikennung in Form einer Zahl zugeordnet. Deswegen wird sie im Workspace als `double` angezeigt. Mit den Variablen kann die geöffnete Textdatei angesprochen werden. For-Schleifen wiederholen die Befehle, die in ihnen stehen. Es gibt eine Laufvariable, die am Anfang auf einen Startwert gesetzt wird. Nach jeder Wiederholung der Schleife wird die Variable um einen Wert erhöht, bis der Zielwert erreicht wird. Die Schleifenvariablen müssen numerisch sein. Es werden ganze Zahlen verwendet und die Laufvariable erhöht sich immer um eine ganze Zahl. Hier wird immer die Laufvariable `n` verwendet. Ist es notwendig eine zweite For-Schleife in eine bereits bestehende Schleife zu durchlaufen, wird die Laufvariable `m` verwendet. Bei der dritten For-Schleife wird die Laufvariable `o` verwendet.

4.1.2 Übersicht

Aus der Aufgabenstellung folgt bereits eine gewisse Programmstruktur. Eine Textdatei mit den Fachwerksinformationen soll eingelesen werden, die Eingabebefehle und die Fachwerksinformationen sollen auf Fehler untersucht werden, das Fachwerk soll grafisch dargestellt werden, ein Fachwerksgleichungssystem soll nach dem Knotenpunktverfahren aufgestellt und dann gelöst werden und die Lösung ist in eine Textausgabedatei zu schreiben. Dadurch ergeben sich bereits fünf Abschnitte des Programms. Hinzu kommt ein Abschnitt, der die eingelesenen Fachwerksinformationen in Matrizen überträgt. Außerdem wird ein separater Abschnitt benötigt, der die Ausgabetextdatei erstellt.

Die einzelnen Abschnitte sollen in gesonderten Skripten in Unterprogrammen ablaufen und von dem Hauptprogramm `Truss_Analysis_Calculation.m` nach dem Top-Down Prinzip aufgerufen werden. Im Hauptprogramm selber befinden sich also keine Schleifen, sodass die Unterprogramme unabhängig voneinander ablaufen, sich nicht überschneiden und nur die Variablen weitergegeben werden. So erfüllt jedes Unterprogramm eine Teilfunktion des gesamten Programms. Es ergeben sich insgesamt sieben Unterprogramme und ein Hauptprogramm. Die Unterprogramme sind nach ihrer Hauptfunktion benannt:

- `Input_from_txt.m`
- `Creation_of_Output_File.m`
- `Fault_Analysis.m`
- `Transfer_to_Double_Matrices.m`
- `Draw_Truss.m`
- `Calculation.m`
- `Write_into_Ouput_File.m`

Die Benennung und Reihenfolge der Unterprogramme sowie die erstellten Variablen geben weiteren Aufschluss über die Funktion der einzelnen Unterprogramme und den Aufbau. Im ersten Schritt wird im Unterprogramm `Input_from_txt.m` die von der benutzende Person vorgegebene Eingabetextdatei geöffnet. Der Name der Datei und der Pfad wird weitergegeben an die Variable `input_text_file` und `folder_path`.

Die Argumente der Eingabebefehle werden in die Matrizen: `N` für die Knoten, `E` für die Stäbe, `D` für die Lager und `F` für die äußeren Kräfte kopiert. Außerdem wird die Textdatei Zeile für Zeile in die Variable `input_text_lines` geschrieben. Im Unterprogramm `Creation_of_Output_File.m` wird eine Ausgabetextdatei erstellt mit dem Namen der Ausgabedatei `output_text_file`, die aus dem Namen der Eingabetextdatei mit der Endung *-OUTPUT.txt* besteht. Es wird ein Titel in die Ausgabedatei geschrieben und die Eingabebefehle wiederholt.
Das Unterprogramm `Fault_Analysis` soll die Eingabebefehle, die in die Matrizen `N, E, D, F` kopiert wurden, auf Fehler untersuchen oder wenn möglich korrigieren. Die Fehler werden mit der entsprechenden Zeilenangabe aus der Eingabetextdatei in die Ausgabedatei geschrieben. Die Variable `major_faults_count` zählt die Fehler, die dazu führen, dass das Fachwerk nicht berechnet und oder dargestellt werden kann.
Das Skript `Transfer_to_Double_Matrices.m` überträgt die Eingabebefehle in die numerischen Matrizen `Joints, Rods, Supports` und `Forces`. Es berechnet außerdem die Länge und die Einheitsvektoren der Stäbe sowie der Lagerreaktionen. Im nächsten Unterprogramm `Draw_Truss.m` wird das Diagramm `Truss` erzeugt, in dem das Fachwerk grafisch dargestellt wird. Die grafische Darstellung wird als Vektorgrafik gespeichert. Im Abschnitt `Calculation.m` wird das Gleichungssystem nach Gleichung 3.18 mit der Koeffizientenmatrix $\underline{\underline{K}}$ und dem äußeren Kraftvektor $\underline{f}$ aufgestellt und nach den Stabkräften und Lagerreaktionen im Vektor $\underline{a}$ aufgelöst. Die Lösung wird im letzten Unterprogramm `Write_into_Ouput_File.m` in die Ausgabetextdatei geschrieben. Die Variablen des linearen Gleichungssystem, welche im Abschnitt `Calculation.m` aufgestellt wurden, werden zudem in dem Ausgabeordner gespeichert.

Werden im Programmablauf Fehler gefunden, die eine weitere Abwicklung verhindern, dann stoppt das Programm die weitere Ausführung der nachfolgenden Unterprogramme. Dies geschieht, wenn die benutzende Person keine Textdatei im Unterprogramm `Input_from_txt.m` als Eingabe auswählt oder kritische Fehler, die in der Variable `major_faults_count` gezählt werden, in den Eingabebefehlen in der Fehleranalyse `Fault_ Analysis` gefunden werden. Eine Übersicht des gesamten Aufbaus ist in der Abbildung 4.1 zu sehen. Die wichtigsten Variablen sind dort mit runden Klammern und die Namen der Unterprogramme mit eckigen Klammern gekennzeichnet.

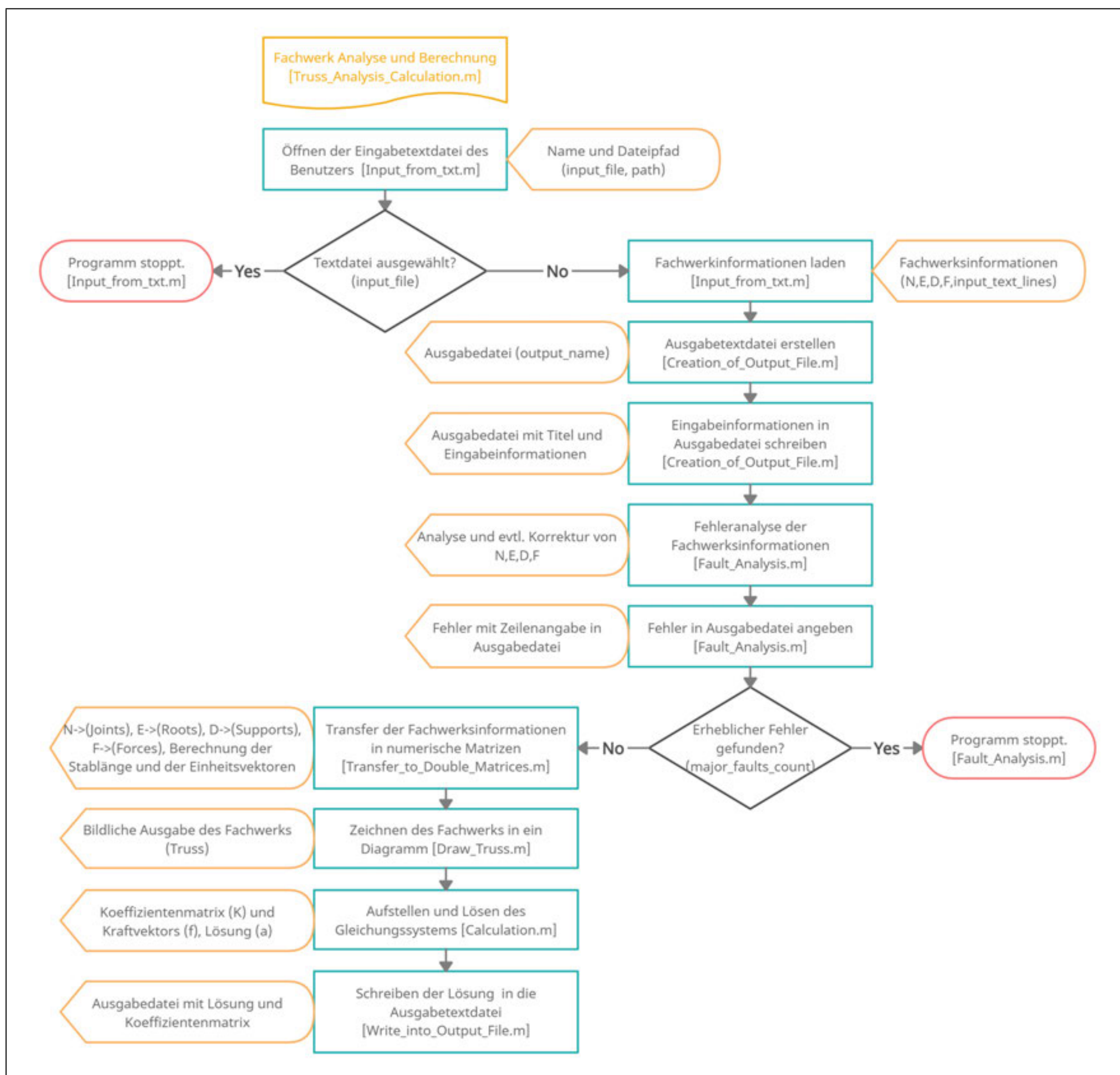


Abbildung 4.1: Übersicht des Programmaufbaus und Programmablaufs mit den wichtigsten Variablen

Die Matlabdateien sind im lokalem Ordner *(...)\Truss_ Analysis_ Calculation_ Software* so angeordnet, dass nur das Hauptprogramm `Truss_Analysis_Calculation.m` sichtbar ist. Die Unterprogramme, bzw. Skripte werden vom Hauptprogramm geöffnet und befinden sich im Unterordner *(...)\Truss_ Analysis_ Calculation_ Software\TAC_ sub_ programms*. Die Benennung der Ordner und der Unterprogramme darf auf keinen Fall geändert werden, da die Unterprogramme und der Unterordner über die jeweilige Benennung geöffnet werden. Es ist darauf zu achten, dass die Anzahl der Zeichen im Pfad des Programms und der Eingabetextdatei zusammen nicht 270 Zeichen übersteigt. Außerdem sollte der Name der Eingabedatei und des Pfads keine Umlaute und Sonderzeichen bis auf Unterstriche enthalten.
Eine zusätzliche Textdatei `Read_Me.txt` gibt Hinweise über die Nutzung, Anordnung und Benennung des Programms und der Unterprogramme. In der Datei `Input_Commands.pdf` erhalten die Benutzer Informationen, wie die Eingabetextdatei mit den Befehlen aus Kap. 4.1.3 zu verfassen ist. Diese ist im Anhang B enthalten.

4.1.3 Eingabedatei und Befehlssatz

Das zu berechnende und zu analysierende Fachwerk wird über eine Eingabedatei von der benutzenden Person individuell definiert. Hierzu können die Benutzer mit verschiedenen Eingabebefehlen in einer Text-Datei mit der Endung *-.txt* die einzelnen Elemente des Fachwerks festlegen.
Die Eingabedatei besteht aus einer sequenziellen Folge von Zeichen des ASCII (American Standard Code for Information Interchange) Zeichensatzes. Jedem ASCII Zeichen wird beim Speichern ein Binärcode aus sieben Bit zugeordnet. Da jedes Bit zwei Werte annehmen kann, ergeben sich daraus die insgesamt $2^7 = 128$ Zeichen des Zeichensatzes. Das achte Bit, das Bit welches ganz links im Binärcode steht, wird für die Fehlerkorrektur als Paritätsbit oder zur Erweiterung auf einen 8-Bit-Code ($8Bit = 1Byte$) eingesetzt. Die UTF-8 (8-Bit Universal Coded Character Set Transformation Format) Kodierung enthält die gleichen 128 Zeichen bzw. Kodierung des ASCII-Zeichensatzes auf den ersten sieben Bits. Deshalb kann die Textdatei in der (ANSIS-)ASCII oder UTF-8 Kodierung gespeichert und vom Programm in Matlab eingelesen werden.

Die Zeichenketten zur Eingabe der einzelnen Befehle sind an die APDL (Ansys Parametric Design Language) Eingabesprache von ANSYS Inc. [ANS21] angelehnt und sind in Tabelle 4.2 enthalten. Die einzelnen Parameter der Befehle werden durch Kommas getrennt und jeder Befehl steht in einer neuen Zeile. Das Ausrufezeichen signalisiert eine Kommentarzeile. Ein Knoten wird durch ein *N*, gefolgt von der Knotennummer, der x-Koordinate und der y-Koordinate im kartesischen Koordinatensystem eingegeben. Die Positionierung aller anderen Fachwerkselemente erfolgt mit der Angabe der Knotennummer und beruht so auf der Angabe der Koordinaten der Knoten. Ein Stab beginnt mit dem Zeichen *E* und enthält die Stabnummer, die Anfangsknotennummer und die Endknotennummer. Der Stab erstreckt sich so vom Anfangsknoten zum Endknoten. Lager können in Form von Loslager, gekennzeichnet durch ein *D*, direkt gefolgt von der zu sperrenden Richtung *X* oder *Y* und der Knotennummer zur Positionierung an einem Knoten eingefügt werden. Kräfte folgen dem gleichen Schema. Sie werden durch ein *F* gekennzeichnet und enthalten folgend die Richtung *X* oder *Y* , in die sie wirken sollen. Zudem setzen sie sich aus der Knotennummer zur Positionierung und aus einen Kraftwert zusammen. Der Wert legt die Größe der Kraft fest. Das Ende der Eingabebefehle zeigt *SOLVE* an.

Tabelle 4.2: Befehlssatz zur Definition der Fachwerkselemente in der Eingabedatei

Befehl	**Bedeutung**
! Kommentar	Eine Zeile Kommentar
N, Knotennummer, x-Koordinate, y-Koordinate	Erzeugt einen Knoten an den angegebenen Koordianten
E,Stabnummer,Anfangsknoten,Endknoten	Erzeugt einen Stab zwischen zwei Knoten
DX,Knotennummer	Sperrt die x-Verschiebung am Knoten auf null
DY,Knotennummer	Sperrt die y-Verschiebung am Knoten
DWinkel,Knotennummer	Sperrt die Verschiebungs-Richtung mit dem Winkel zu vertikalen Achse am Knoten
FX,Knotennummer,Wert	Erzeugt eine Kraft, die in x-Richtung wirkt, an den Knoten
FY,Knotennummer,Wert	Erzeugt eine Kraft, die in y-Richtung wirkt, an den Knoten
SOLVE	Zeigt das Ende des Datensatzes an

Zusätzlich wurde die Möglichkeit implementiert, auch schräge Lager einfügen zu können. So können Loslager eingefügt werden, die die Verschiebung nicht nur in horizontaler oder vertikaler Richtung blockieren. Dies geschieht mit der Angabe des Winkels zur vertikalen Achse und ist aus Abbildung 4.2 zu entnehmen. Der Winkel dreht im kartesischen Koordinatensystem gegen den Uhrzeigersinn - mathematisch positiver Drehsinn - und soll zwischen $-180°$ und $180°$ liegen. So wird die Richtung X oder Y durch den Winkel in Grad ersetzt. Als Dezimaltrennzeichen ist in allen numerischen Eingabeargumenten ein Punkt zu benutzen.

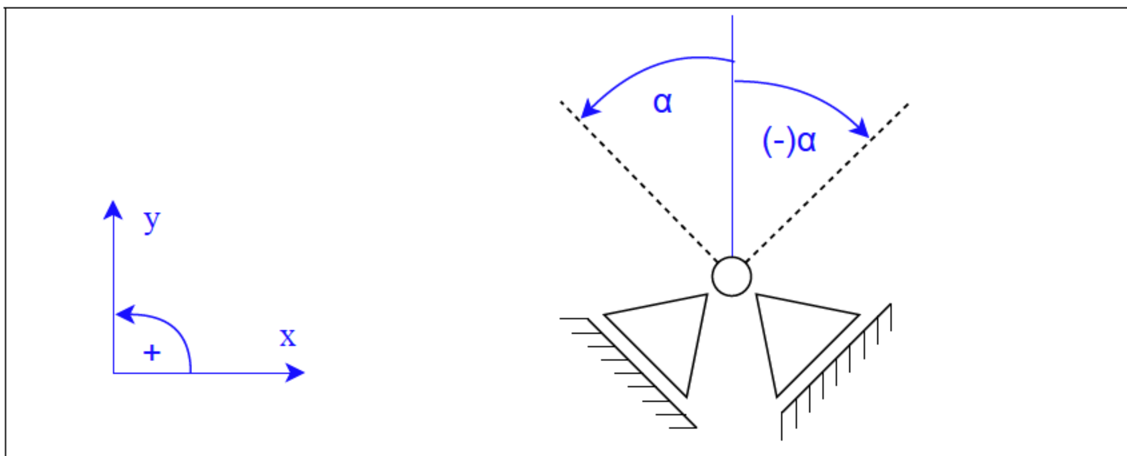


Abbildung 4.2: Ausrichtung des Lagers bei Angabe mit dem Winkel zur horizontalen Achse

Das Programm rechnet einheitenlos. Bei Angabe von äußeren Kraftwerten und den Koordinaten der Knoten in der Eingabedatei werden keine Einheiten angegeben. Es ist deshalb darauf zu achten, dass alle Längen- und Kraftwerte auf den gleiche Einheiten basieren, auch wenn sie nicht angegeben werden. Die ausgegebenen Stabkräfte und Lagerreaktionen entsprechen dann der Einheit der eingegebenen Kraftwerte.

4.2 Programmablauf

Das Programm soll eigenständig arbeiten und ähnlich einem FEM-Programm ablaufen. Dazu liest es im ersten Schritt eine Eingabedatei ein und gibt am Ende mehrere Ausgabedateien mit der bildlichen Darstellung und einer Lösungsdatei aus. Die Benutzer müssen also nur das Programm starten und die Eingabedatei auswählen.

Der restliche Vorgang läuft automatisch im Hintergrund ab und die benutzende Person wird über das Ende des Programmablaufs informiert. Dies ermöglicht eine einfache Benutzung des Programms und verhindert eine Manipulation durch die Benutzer im Command Window, wenn diese aufgefordert werden, zusätzliche Eingaben zu tätigen. Eine grobe Übersicht über den Programmablauf kann dem Kap. 4.1.2 und der Grafik 4.1 entnommen werden.
In den folgenden Unterabschnitten wird nun der Ablauf und die Funktionsweise des Programms und der einzelnen Abschnitte detailliert beschrieben und erklärt. Es wird auf die Abfolge der Matlab-Befehle und Schleifen im Skript jedes Unterprogramms und des Hauptprogramms eingegangen. Die einzelnen verwendeten Matlab-Befehle und Matlab-Anweisungen können nicht immer genau erläutert werden, um die weitere Beschreibung auf das Wesentliche des hier entwickelten Programms zu fokussieren. Eine ausführliche Erläuterung der einzelnen Matlab-Befehle kann im Hilfebereich der Homepage [Mat21] oder durch das rechts-klicken auf die jeweilige Funktion und Auswählen von *Help on...* aufgerufen werden. Die vollständigen Skripte der einzelnen Programme sind im Anhang A einsehbar.

4.2.1 Hauptprogramm

Über das Hauptskript `Truss_ Analysis_Calculation.m` startet die benutzende Person den Ablauf des gesamten Matlab-Programms. Es ist also das Kernstück und der Ablauf spiegelt dabei den Aufbau aus der Abbildung 4.1 wieder. Über das Skript werden die unterschiedlichen Unterprogramme in einer bestimmten Reihenfolge gestartet. Das Hauptprogramm dient so vielmehr als Benutzeroberfläche und ‚Steuereinheit'. Die Benutzer werden in den ersten 87 Skriptzeilen in Kommentaren über die Eigenschaften, die Benutzung, die Benennung und die wichtigsten Variablen informiert. Die folgenden Zeilen bedeuten:

Zeile 93–96: Löschen aller Variablen und aller untergeordneten Elemente der Diagramme mit `clear` und `clf`. Löschen des gesamten Textes aus dem Command Window. Schließen und löschen etwaiger offener Dateien und Diagramme mit `close all` und `clc`.

99–107: Drucken der Kopfzeile und Titel des Programms in das Command Window mit dem Befehl `disp()`.

110: Mit `addpath()` wird der Dateipfad *TAC_Sub_Programms* in das Verzeichnis von Matlab hinzugefügt. Dies wird benötigt, um im folgenden die einzelnen Unterprogramme, die sich in diesem Ordner befinden, zu öffnen. Auch die folgenden Unterprogramme werden mit dem Namen angesprochen und ausgeführt. Aus diesem Grund darf wie in Kap. 4.1.2 beschrieben der Name des Unterordners und der Unterprogramme nicht geändert werden.

115–145: Die einzelnen Unterprogramme werden mit dem Befehl `run()` und dem Namen des jeweiligen Skripts ausgeführt. Deutlich erkennbar ist die Abfolge der einzelnen Unterprogramme aus Abbildung 4.1.

148–151: Die Benutzer werden über das erfolgreiche Abschließen des Programms im Command Window und in einem Nachricht-Dialogfeld informiert.
Das Nachricht-Dialogfeld wird mit `msgbox()` erzeugt, hat den Titel Completion und zeigt mit `'help'` ein Infosymbol an. Leider ist es nicht möglich einen Link zum Ordner mit den Ausgabedateien in dem Nachricht-Dialogfeld anzuzeigen. Deswegen wird der Link zum Ordern mit dem Ausgabedateien im Command Window ausgegeben. Der Pfad liegt in der Variable `folder_path` als Zeichenkette und wird im Unterprogramm `Input_from_txt.m` aus der ausgewählten Eingabetextdatei ermittelt.

4.2.2 Eingabedatei öffnen und lesen

Das Unterprogramm `Input_from_txt.m`, welches vom Hauptprogramm als erstes aufgerufen wird, öffnet die von der benutzenden Person ausgewählte Textdatei und sucht in ihr nach Zeichenketten, die die einzelnen Fachwerkselemente definieren. Die hier definierten Variablen `N,E,D,F` enthalten die Schlüsselinformationen und sind die Grundlage für die weitere Verarbeitung. Jede Variable steht für ein Element des Fachwerks und die Benennung richtet sich nach den Anfangsbuchstaben der Eingabebefehle. Die Argumente aus den Stab-Eingabebefehlen befinden sich so zum Beispiel in der Variable `E`. Die Argumente werden getrennt in eine Zelle der Variable geschrieben. Die Argumente mehrerer Elemente werden in einer Zelle in einem Vektor untereinander geschrieben. Der Inhalt der einzelnen Zellen ist in Tabelle 4.3 aufgeführt.

Tabelle 4.3: Zellen der Element-Variablen N,D,E und F

Element	Variable	Zelle 1	Zelle 2	Zelle 3
Knoten	N	Knotennummer	x-Koordinate	y-Koordinate
Stab	E	Stabnummer	Anfangsknotennummer	Endknotennummer
Loslager	D	gesperrte Verschiebungsrichtung	Positionierungs-Knotennumer	n/a
äußere Kraft	F	Wirkungsrichtung	Positionierungs-Knotennumer	Kraftwert

Zusätzlich wird die Variable `input_text_lines` erstellt. In ihr soll die Textdatei Zeile für Zeile gespeichert werden, um später, wenn bei der Eingabe ein Fehler gemacht wurde, die genaue Position des Fehlers in der Eingabetextdatei bestimmen zu können. Außerdem wird sie genutzt, um die Eingabebefehle in der Ausgabedatei zu wiederholen. Die einzelnen Zeilen in dem Skript bedeuten:

Zeile 14: Ein Windowsdialogfenster zur Dateiauswahl wird über `uigetfile` geöffnet. Die Benutzer werden aufgefordert, die Eingabetextdatei auszuwählen. Der Pfad und der Name der Datei werden in die entsprechende Variable `input_name` und `folder_path` zurückgegeben. Es wird nicht die Variable `path` benutzt, weil dies auch ein Befehl von Matlab ist. Über `'.txt'` wird der Filter zur Auswahl der Dateien auf Textdateien mit dieser Endung gesetzt.

18–23: In der nächsten Schleife wird überprüft, ob die benutzende Person eine Textdatei ausgewählt hat. Ist die Textdatei leer, dann ist der Eingabename gleich null: `isequal(input_name,0)`. Wenn keine Textdatei ausgewählt wurde, entspricht die Endung der letzten vier Zeichen in dem Namen der Eingabedatei nicht *.txt*: `all(input_name(end-3:end) ='.txt'`. Ist eine dieser Bedingungen erfüllt, wird die benutzende Person über ein Nachrichten-Dialogfeld `msgbox()` mit einer Nachricht und einem eingebauten Fehlersymbol `'error'` über den Stopp des Programms informiert. Darauf beendet der Fehlerbefehl `error()` mit der gleichen Nachricht das Programm. Die Nachricht wird in dem Dialogfeld wiederholt, da der im Skript provozierte Fehler mit dem Fehlerbefehl schwer von einem Programmfehler zu unterscheiden ist. Leider ist es nicht möglich, das Programm auf eine andere Weise zu stoppen. Der Befehl `quit` schließt das gesamte Matlab-Fenster und `return` würde lediglich die weitere Ausführung der Funktion beenden und zum übergeordneten Skript zurückkehren. Ein Fehler in dem folgendem Unterprogramm wäre die Auswirkung.

27: Die vorher von den Benutzern ausgewählte Textdatei wird mit dem Pfad und dem Namen in der Variable `Input` geöffnet. Dabei wird die Dateizugriffsart mit `'r'` auf Leseberechtigung gesetzt.

30–31: In der Funktion `read_in_informations` werden die Befehle der Eingabedatei separat in die Art der Elemente und in eine Variable eingelesen. Die Variablen `N`, `D`, `E` und `F` sind nach den Anfangsbuchstaben des jeweiligen Elementbefehls benannt. Zudem bildet die Variable `input_text_lines` alle Zeilen der Eingabetextdatei ab. Diese fünf Variablen werden von der Funktion zurückgegeben. Eingegeben wird die Textdatei in der Variable `Input` und der Name und Pfad der Eingabedatei.

39–46: Die Textdatei soll Zeile für Zeile in die Variable `input_text_lines` eingelesen werden. Dazu wird zunächst die gesamte Textdatei mit `fileread()` in eine Zeichenkette eingelesen. Der Befehl arbeitet nicht mit der Dateikennung `Input`. Deswegen muss der Pfad und der Name der Eingabedatei angegeben werden. Durch den Befehl `'split'` wird die Zeichenkette bei Zeilenumbrüchen `\n` in einen Zeilenvektor getrennt. Es befinden sich immer noch die Zeilenumbrüche oder Wagenrückläufe der Textdatei in den Zeichenketten, die in Matlab als Pfeil ← angezeigt werden.

Diese sollen im nächsten Schritt entfernt werden, weil sonst später keine leeren Zellen erkannt werden können. Durch `[\n\r]+` werden alle Zeilenumbrüche `\n`" und Wagenrückläufe `\r` in den Zeichenketten entfernt Der Zeilenvektor wird nun um 270° gegen den Uhrzeigersinn in einen Zeilenvektor gedreht, sodass jede Zeile im Vektor der gleichen Zeile in der Eingabetextdatei entspricht. Als letztes werden Kommentarzeilen, beginnend mit einem Ausrufezeichen, genauso wie die Zeilenumbrüche entfernt. Durch `ˆ[!]` wird nur am Anfang einer Zeile nach dem Ausrufezeichen gesucht und durch `.*$` werden dann alle folgenden Zeichen bis zum Ende der Zeile erfasst. Die leeren Zeilen sollen nicht gelöscht werden, da später in der Fehleranalyse in Abschnitt 4.2.4 die Zeile des Fehlers in der Eingabetextdatei ausgegeben werden soll. Wenn die Zeilen gelöscht werden würden, würden die Zeilenindizes im Vektor `input_text_lines` mit den Zeilen der entsprechenden Eingabebefehlen in der Eingabedatei nicht mehr übereinstimmen.

48–53: Durch ein *SOLVE* wird das Ende des Befehlssatzes angegeben. Weitere Zeilen und Zeichenketten sollen nicht beachtet werden und müssen gelöscht werden, da es sonst zu Fehlern kommen kann, wenn die weiteren Zeilen zum Beispiel Ersatzeingabebefehle oder einen frei geschriebenen Text ohne Ausrufezeichen enthalten. Wenn eine Zeile die Zeichenkette *SOLVE* enthält, werden durch den Befehl `find()` alle diejenigen Zeilen im Vektor in die Variable `solve_line` zurückgegeben. Hätte die benutzende Person wiederum die Angabe von *SOLVE* vergessen, es falsch geschrieben oder nicht die Großschreibung beachtet, würde der Befehl in Zeile 50 zu einem Fehler führen, da keine Zeile gefunden werden würde. Durch die nächste Abfrage wird dies verhindert, da geprüft wird, ob eine Zeile zurückgegeben wurde. Nun werden alle Zeichenketten im Vektor `input_text_lines` ab der Zeile mit dem ersten *SOLVE* geleert. Da durch den Befehl `regexp()` in Zeile 37 der Datentyp `cell array` entstanden ist, muss die leere Zeichenkette mit `cellstr()` in dieses Format konvertiert werden.

56: Die Variablen, in die die unterschiedlichen Eingabebefehle aus der Textdatei eingelesen werden sollen, müssen vordefiniert werden, weil sonst die einzelnen Zellen nicht angesprochen werden können. Durch den Matlabbefehl `textscan()` entsteht ein multidimensionaler `cell-array` mit einer Zelle für jedes Argument des Eingabebefehls.

So hat die Variable `N` für die Knoten drei Zellen mit einem Zeilenvektor für die Argumente Knotennummer, x-Koordinate und y-Koordinate. In die Zellen werden die Argumente jedes eingegebenen Knotens in eine neue Zeile geschrieben (vgl. Tabelle 4.3).

58–75: Durch `frewind()` wird der Dateipositionsanzeiger wieder zurück an den Anfang gesetzt. Solange der Positionsanzeiger nicht am Ende der Textdatei ist (`feof(Input)==0`) sollen durch `textscan()` die nächsten Knoteneingabebefehle eingelesen werden. Der Befehl `textscan()` sucht nach Zeilen mit Knotenbefehlen, die mit `'N'` beginnen und liest die nächsten drei Zeichenketten `'%s %s %s %s'` ein. Durch `'Delimeter'` wird das Trennzeichen auf ein Komma zwischen den einzulesenden Zeichenketten gesetzt und Kommentarzeilen, die mit einem Ausrufezeichen beginnen, werden mit `'CommentStyle'` ignoriert. Der Befehl `textscan()` bricht das Einlesen ab, wenn eine Textzeile in der Eingabedatei auftaucht, die nicht dem Format `'N %s %s %s %s'` mit drei Kommas entspricht. Deswegen werden die eingelesenen Argumente zunächst in der Variable `N_scan` zwischengespeichert und dann in der Variable `N` an das Ende der jeweiligen Zelle kopiert. Es werden Zeichenketten und keine Zahlen mit `'N %f %f %f %f'` eingelesen, weil das Einlesen bei einem auftauchenden Buchstaben in einem Argument sonst abgebrochen wird. Es fehlen dann Argumente und die Zellen haben nicht mehr die gleiche Anzahl von Zeilen, was zu einem Programmfehler führen würde. Die Eingabe eines Buchstabens würde nicht der Spezifikation der Eingabebefehle entsprechen und wäre ein falsches Format. Diese falsche Eingabe wird später in der Fehleranalyse in Kap. 4.2.4 erkannt.
Da der Befehl `textscan()` in einer Zeile stoppt, die nicht dem eingegebenen Format entspricht, wird die nächste Zeile eingelesen und überprüft, ob sie die Zeichenkette `'SOLVE'` enthält. Ist dies der Fall, wird die übergeordnete Schleife `while feof(Input)==0` bereits vor dem Ende der Textdatei mit `break` verlassen, da die Zeichenkette das Ende der Eingabebefehle bedeutet und der restliche Text dahinter ignoriert werden muss. Enthält die nächste Zeile nicht den Endbefehl der Eingabedatei, springt der Dateipositionszeiger in die nächste Zeile und die Schleife beginnt von vorne, bis das Ende der Textdatei erreicht ist und alle Knotenbefehle eingelesen wurden.

78–133: Die nächsten Schleifen folgen dem gleichen Aufbau, wie dem der Schleife zum Einlesen der Knotenbefehle. Hier werden nun die Stäbe, Lager und äußeren Kräfte in jeweils eine Variable eingelesen. Die Befehle unterscheiden sich durch den Anfangsbuchstaben und die Anzahl der Eingabeargumente, sodass das Format in `textscan()` angepasst werden muss. Da mit `%s` Zeichenketten eingelesen werden, werden sowohl die Buchstaben in dem ersten Argument der Lager und äußeren Kräfte als auch die Zahlen erfasst.

140–160: Wurde bei der Eingabe der Elementbefehle in der Textdatei das letzte Argument vergessen und befindet sich zudem der Befehl am Ende der Textdatei, dann bleibt die letzte Zeile in der letzten Zelle leer. Wenn also *DX,* am Ende der Eingabetextdatei steht, wird nur das *X* in die erste Zelle kopiert. Dadurch haben die Zellen unterschiedlich viele Zeilen. Dies muss durch eine leere Zeile, die ans Ende der letzten Zelle gehängt wird, ausgeglichen werden, weil sonst im Folgendem ein Programmfehler entsteht. Es wird also überprüft, ob die Anzahl der Zeilen in der ersten Zelle größer ist als die der letzten Zelle. Dann wird eine leere Zeile manuell an das Ende der letzten Zelle gehängt, sodass die Zellen die gleiche Größe haben. Fehlt ein Argument eines Eingabebefehls nicht am Ende der Eingabedatei, erstellt der Befehl `textscan()` automatisch eine leere Zeile für das fehlende Argument in der entsprechenden Zelle. Fehlt auch das zweitletzte Argument am Ende der Textdatei muss auch an die zweite Zelle eine leere Zeile gehängt werden. Die Abfragen werden also für die zweitletzte Zelle der Element-Variablen wiederholt. Nur bei den Lagern ist dies nicht nötig, weil sie nur zwei Befehlsargumente und die Element-Variable nur zwei Zellen hat.

33: Die geöffnete Eingabedatei wird geschlossen.

4.2.3 Ausgabedatei erstellen

In dem Unterprogramm `Creation_of_Output_File.m` soll die Ausgabetextdatei erstellt werden. Die Ausgabetextdatei soll später alle Informationen über die Eingabe, die Fehler, die Berechnung und die Lösung enthalten. Sie wird in dem Unterprogramm `Fault_Analysis.m` und `Write_into_Output_File.m` geöffnet und beschrieben. Deswegen wird sie bereits jetzt im Programmverlauf erstellt. Die Ausgabetextdatei soll im gleichen Ordner wie die Eingabetextdatei liegen und mit dem Namen der Eingabedatei und *-Output.txt* benannt werden. Hier wird zudem ein Titel in die Ausgabedatei geschrieben und die Eingabebefehle werden wiederholt.

Zeile 11–13: Der Name der Ausgabedatei wird erzeugt. Hierzu wird an den Namen der Eingabedatei die Erkennung für die Ausgabedatei *-OUTPUT.txt* angehängt und in der Variable `output_name` gespeichert. Im nächsten Schritt kann dann eine neue Textdatei mit dem Dateipfad und dem vorher erstellten Namen in der Variable `Output` geöffnet werden. Die Ausgabedatei wird im gleichen Ordner der Eingabedatei, die in der Variable `folder_path` liegt, gespeichert. Wichtig ist, dass die Datei mit dem Dateizugriffstyp `'w'` für write geöffnet wird. Dadurch wird eine neue Datei zum Schreiben erstellt. Falls eine Datei mit gleichem Namen und Dateipfad vorhanden ist, wird der vorhandene Inhalt verworfen. Dies ist wichtig, falls die Benutzer nach einem Fehler Anpassungen an dem Fachwerk in der Eingabetextdatei vornehmen und das Programm erneut startet.

17–18: Die Funktion `write_title_input_into_output` druckt den Titel, weitere Informationen und die Eingabebefehle in die Ausgabedatei. Hierzu wird die Erkennung der Ausgabedatei, die Namen der Ein- und Ausgabedateien, der Dateipfad und die Eingabebefehle in die Funktion eingegeben. Die Funktion gibt keine Variablen zurück, da sie die Ausgabedatei beschreibt und alle Informationen in der Textdatei in einem lokalen Ordner gespeichert werden.

27–35: Der Titel und die wichtigsten Informationen über das Programm werden in die Textdatei geschrieben. Er enthält auch Informationen über Version, Autor, Universität und Departement. Der Titel wird oberhalb und unterhalb von einer Zeile mit Sternen hervorgehoben.

Alle weiteren Überschriften in der Ausgabetextdatei werden mit einer Zeile Sternen unterstrichen, da es sonst keine Möglichkeiten gibt, in einer ASCII oder UTF-8 Textdatei Überschriften durch Fettdruck oder einer Linie zu unterstreichen.

36–46: Dem Titel folgen Namen und Speicherort der Eingabedatei. Wenn der Speicherort in der Variable `folder_path` mehr als 76 Zeichen hat, wird der Pfad in zwei Zeilen aufgeteilt. Damit wird verhindert, dass die Zeile zu lang wird. Ermittelt wird dies über eine Abfrage der Länge der Zeichenkette. Die Hälfte der Zeichenkette muss gerundet werden, da bei einer ungeraden Zahl eine Dezimalzahl entstehen würde. Es kann mit einer Dezimalzahl kein Zeichen in einem Vektor angesprochen werden.

49–57: Nun sollen die Eingabebefehle in der Ausgabedatei wiederholt werden. Zuvor wird die entsprechende Überschrift in die Ausgabedatei gedruckt. Die Eingabebefehle liegen in der Variable `input_text_lines`. Zunächst müssen aber noch die leeren Zellen im Vektor gelöscht werden. Leere Zeilen sind durch Kommentarzeilen und durch von der benutzenden Person leer gelassenen Zeilen in der Eingabetextdatei entstanden. Kommentarzeilen wurden im Unterprogramm `Input_from_txt.m` geleert. Gelöscht werden sie nun mit dem Befehl `cellfun()`. Durch die Tilde (˜) vor dem Befehl werden die Zellen ausgewählt, die nicht leer sind und in die Variable `input_text_lines` zurückgegeben und ersetzt. Als letzter Schritt werden nun die Zeichenketten in der Variable durch den Befehl `writecell()` in die Ausgabedatei geschrieben. Durch den Schreibmodus `'append'` werden die Zeichenketten am Ende des vorhandenen Textes eingefügt. Der Befehl kann nicht mit der Dateikennung der Ausgabedatei `Output` arbeiten. Deswegen muss die Ausgabedatei mit dem Pfad und dem Namen angesprochen werden.

20: Die geöffnete Ausgabedatei wird geschlossen.

4.2.4 Fehleranalyse der Eingabebefehle

Das Unterprogramm `Fault_Analysis.m` enthält das bei weitem umfangreichste Skript, weil es möglichst viele Fehler aus der Eingabetextdatei und aus dem Zusammenbau des Fachwerks erkennen soll. Wenn die Befehle in der Eingabetextdatei richtig verwendet werden und ein ebenes, starres und statisch bestimmtes Fachwerk aufgebaut wird, dann geschieht in diesem Unterprogramm nichts und es wäre überflüssig. Die Fehler werden hier in zwei Kategorien eingeteilt:

- Erhebliche Fehler werden mit der Variable `major_faults_count` gezählt. Das Programm wird bei Entdecken dieser Fehler nach den einzelnen Funktionen gestoppt, da eine weitere Berechnung und Darstellung nicht möglich ist.
- Kleine Fehler werden korrigiert und mit der Variable `faults_count` gezählt.

Die Fehleranalyse ist aus kleineren Funktionen aufgebaut, die die Eingabeinformationen auf eine bestimmte Fehlerart untersuchen. Diese sind in einer bestimmten Reihenfolge angeordnet:

- Großschreibungs- und Leerzeichenfehler in der Funktion `correct_capitalisation_spaces_fault`
- strukturelle Fehler in der Funktion `structural_fault`
- Repetitionsfehler in der Funktion `identical_elements_fault`
- Argumentfehler in der Funktion `argument_fault`
- Nummerierungsfehler in der Funktion `numbering_fault`
- korrigierbare Fehler in der Funktion `correction`
- Assemblierungsfehler in der Funktion `assembly_fault`
- Überprüfung der statischen Bestimmtheit in der Funktion `statically_defined`

Zuerst sollen also Fehler behoben werden, bei dem die Großschreibung nicht beachtet wurde, damit alle Eingabebefehle im Folgenden richtig erkannt werden können. Besonders wichtig ist es auch, Leerzeichen zu löschen, da in der Fehleranalyse bestimmte Zeichenketten beruhend auf den Eingabebefehlen verglichen werden. Mit Leerzeichen wäre dies nicht möglich und ein Vergleich sonst identischer Zeichenketten würde keine Übereinstimmung liefern.

Danach wird nach strukturellen Fehlern gesucht, also Fehler, die darauf beruhen, dass die benutzende Person die Eingabebefehle und deren Zeichenfolge nicht richtig beachtet und verfolgt hat. Nun werden alle Befehle gelöscht, die eine exakte Kopie eines bereits vorhandenen Befehls sind. Sonst würden alle folgenden Fehler doppelt auftauchen. Im nächsten Schritt werden die Argumente der Eingabebefehle auf Vollständigkeit und das richtige Format überprüft. Nun ist sichergestellt, dass alle Befehle und Argumente vollständig vorhanden sind. Die folgende Fehleranalyse beschäftigt sich dann mit dem Zusammenbau des Fachwerks. Jede Knotennummer darf zur eindeutigen Identifizierung nur einmal verwendet werden. Außerdem muss die angegebene Knotennummer in den Elementen auch existieren. Einige Fehler können korrigiert werden. Danach wird die Position und Anordnung der Elemente im Fachwerk überprüft und zuletzt der Grad der statischen Bestimmtheit berechnet, um zu überprüfen, ob die notwendige Bedingung für statische Bestimmtheit erfüllt wird. Werden kritische Fehler entdeckt, dann stoppt das Programm in der jeweiligen Funktion. Die Fehleranalyse wird nicht fortgesetzt, da die Fehler aufeinander aufbauen. Eine fehlende Knotennummer, die in der Funktion `argument_fault` entdeckt wird, würde viele weitere Nummerierungsfehler und Assemblierungsfehler zur Folge haben. Dadurch wäre die Anzahl der ausgegebenen Fehler sehr hoch und der eigentliche Fehler wäre schwer erkennbar.
Um auch fehlerhafte Eingabebefehle und Argumente einlesen zu können, wurden im Unterprogramm `Input_from_txt.m` in Kap. 4.2.2 bereits einige Anpassungen vorgenommen. Einige Fehler können auch erst später korrigiert werden. So wird im Unterprogramm `Transfer_to_Double_Matrices.m` in Kap. 4.2.5 die Knotennummerierung korrigiert, wenn diese nicht aus einer fortlaufenden Nummerierung besteht.
Es soll darauf hingewiesen werden, dass hier nicht alle Fehler entdeckt werden können. Durch die Unmengen an Zeichenkombinationen und Eingabeformate, die möglich sind, kann nicht jede Kombination überprüft werden. Durch die oben beschriebene Anordnung der Funktionen kann der größte Teil aber erkannt werden. Zudem ist es nicht möglich an dieser Stelle zu überprüfen, ob das Fachwerk unbeweglich ist, also kinematisch bestimmt ist. Der Grad der statischen Bestimmtheit wird berechnet und so wird überprüft ob das Fachwerk die Voraussetzung für statische Bestimmtheit erfüllt. Später wird bei der Berechnung die Determinante der Koeffizientenmatrix gebildet. Bei einem Beweglichen Fachwerk ist das Gleichungssystem nicht lösbar und die Determinante ist null. So wird in der Berechnung überprüft, ob das erstellte Fachwerk nicht beweglich ist. Die einzelnen Schleifen, die das Fachwerk auf die unterschiedlichen Fehler in den unterschiedlichen Element-Variablen überprüften sind ähnlich aufgebaut.

Viele Befehle, Schleifen und Befehlskombinationen wiederholen sich und werden beim erneuten Auftreten nicht mehr im Detail beschrieben. Im Skript werden die einzelnen Funktionen, die die Eingabebefehle auf Fehler untersuchen, mit dem Namen aufgerufen:

Zeile 13–16: Die Ausgabetextdatei wird mit der Zugriffsart `'a'` für append geöffnet. Dadurch werden folgende Eingaben an den bereits vorhandenen Text angehängt und nicht überschrieben. Dann wird die Überschrift der Fehleranalyse eingefügt.

22–85: Es werden hintereinander die einzelnen Funktionen aufgerufen, um den Eingabetext auf die verschiedenen Fehler zu untersuchen. Funktionen, die die Eingabeinformationen auf kritische Fehler untersuchen haben keine Ausgabevariablen. Wird ein Fehler entdeckt stoppt das Programm über eine Schleife in der entsprechenden Funktion. Wurde kein Fehler entdeckt, werden die Variablen auch nicht verändert. Eine Ausgabe ist nicht nötig. Allgemein werden immer die Element-Variablen `N, E, D, F`, der Eingabetext Zeile für Zeile `input_text_lines`, die Dateikennung für die Ausgabedatei `Output` und der Dateipfad `folder_path` eingegeben. Nicht in allen Funktionen werden aber alle Variablen benötigt und einige können weggelassen werden.

88–93: Die Benutzer sollen in der Ausgabedatei informiert werden, wenn kein Fehler gefunden werden konnte. Da kritische Fehler zum Stopp des Programms führen, wird die Anzahl der korrigierbaren Fehler in der Variable `faults_count` abgefragt. Ist die Anzahl der Fehler null, dann wurden keine Fehler in den Eingabeinformationen gefunden und die Benutzer werden in der Ausgabedatei informiert. Daraufhin wird die Textausgabedatei geschlossen.

Großschreibungs- und Leerzeichenfehler

Großschreibungs- und Leerzeichenfehler werden in der Funktion `correct_capitalisation_spaces_fault` behoben. Ein- und ausgegeben werden die Variablen, die die Eingabebefehle und den Eingabetext enthalten: `N, E, D, F` und `input_text_lines`. Eine Korrektur findet auch in der Variable `input_text_lines` statt, weil sie mit den Argumenten aus den Element-Variablen nach der Fehlerzeile durchsucht wird. Kleingeschriebene Buchstaben müssen kapitalisiert werden, weil Matlab zwischen Groß- und Kleinschreibung unterscheidet. Auch Leerzeichen gelten in Matlab als Zeichen und müssen entfernt werden, damit später bestimmte Zeichenketten gefunden werden können.

Mit der Verbesserung kann zum Beispiel im folgendem erkannt werden, dass mit der fehlerhaften Eingabe *D x,3* die x-Verschiebung am Knoten mit der Nummer drei gesperrt werden soll. Korrigiert lautet der Befehl dann *DX,3*. Hier wird ausschließlich mit dem Befehl `regexprep()` gearbeitet, der bestimmte Zeichenketten in einem `string-array` ersetzen kann.

Zeile 103–107: Löschen aller Leerzeichen in der Variable `input_text_lines`. Wenn die Benutzer ein Ausrufezeichen für eine Kommentarzeile nicht direkt am Satzanfang hinter einem oder mehreren Leerzeichen platziert haben, wurde die Kommentarzeilen im Kap. 4.2.2 nicht erkannt. Deswegen werden diese mit dem gleichen Befehl erneut gelöscht.

109–115: Alle Leerzeichen werden aus den verschiedenen Zellen der Element-Variablen gelöscht.

118–128: Nun werden alle Kleinbuchstaben, die relevant für die Eingabebefehle sind, zu Großbuchstaben korrigiert.

Strukturelle Fehler

In der Funktion `structural_fault` sollen alle strukturellen Fehle gefunden werden. Hierzu muss nur die Variable `input_text_lines` untersucht werden, die alle Eingabebefehle Zeile für Zeile enthält. Außerdem wird die Dateikennung des Ausgabeordners `Output` und der Pfad `folder_path` benötigt, um in die Ausgabedatei den Fehler schreiben zu können. Strukturelle Fehler können nicht behoben werden, da sie auf vielen verschiedenen Ursachen beruhen können und müssen deswegen individuell von der benutzenden Person korrigiert werden.

Als strukturelle Fehler gelten Fehler bei dem die Benutzer die Struktur des Eingabesatzes nicht beachtet haben:

- Zu viele oder zu wenige Eingabeargumente in einem Befehl (z.B. *N,1,2,5,6*)
- Als Dezimalseparator wurde ein Komma anstatt einem Punkt benutzt oder umgekehrt (z.B. *FX,2,1,5*)
- Mehr als ein Eingabebefehl in einer Zeile (z.B. *DX,2 E,1,2,5*)
- Fehlendes Ausrufezeichen für eine Kommentarzeile
- Fehlende Eingabe von *N,E,D,F* am Anfang der Befehle (z.B. *1,4,6*)
- Ein Befehl wurde in mehrere Zeilen getrennt
- Das Ende des Befehlssatzes durch *SOLVE* wurde falsch geschrieben (z.B. *Solve*)

Diese Fehler können durch die Anzahl von Kommas in einer Zeile und den Buchstaben am Zeilenanfang entdeckt werden. Die Kommas sind wichtig um die einzelnen Argumente in einem Befehl abzugrenzen und in Kap. 4.2.2 die Argumente richtig einzulesen. Ohne die richtige Struktur wurden die Befehle auch nicht richtig in die Element-Variablen `N, E, D` und `F` eingelesen. Eine weitere Ausführung des Programms bei einem strukturellen Fehler ist nicht möglich. Das Fachwerk kann nicht berechnet und dargestellt werden. Es kommt zum Abbruch des Programms.

Zeile 143–145: Die Variable `major_faults_count` zählt die folgenden Fehler und wird zu Beginn auf null gesetzt. Zudem wird die Variable `input_text_-lines` in einen Zeichenvektor,`char-array`, mit dem Befehl `char()` konvertiert. Einzelne Zeichen in einem `char-array` können schneller und einfacher angesprochen werden als in einem `cell-array`.

147: Nun wird jede Zeile des Zeichenvektors, also jede Zeile der Eingabetextdatei, mit der For-Schleife durchlaufen und auf Fehler untersucht. Kommentarzeilen und Zeilen hinter dem Befehlssatzende wurden bereits geleert. Wenn die benutzende Person keinen Fehler gemacht hat, befinden sich also nur noch Eingabebefehle im Vektor.

150–159: Die erste Abfrage überprüft ob am Zeilenanfang ein *N* für Knoten oder *E* für Stab steht. Wenn zudem die Anzahl der Kommas in einer Zeile nicht genau drei beträgt, liegt eine falsche Struktur im Eingabebefehl eines Knoten oder Stabs vor.

Die Anzahl der Kommas wird mit `count()` in der Zeile der Laufvariable `n` gezählt.
Die Anzahl der kritische Fehler wird erhöht und die Benutzer werden über den Fehler in der Ausgabedatei informiert. Es wird die Zeile angegeben, die der Laufvariable der übergeordneten For-Schleife entspricht.

163–171: Wenn die erste Abfrage nicht zutrifft prüft die zweite Abfrage die eingegebenen Kraftbefehle. Beginnt die Zeile mit einem F für eine äußere Kraft, müssen sich bei einem korrekt eingegebenen Befehl zwei Kommas in der Zeile befinden. Wenn die Anzahl der Kommas nicht übereinstimmt wird die Anzahl der kritischen Fehler erhöht und die Benutzer werden genauso wie in der vorherigen Abfrage über den Fehler in der Ausgabedatei informiert.

175–183: Zuletzt prüft die dritte Abfrage die eingegebenen Lagerbefehle. Dies geschieht im gleichen Prinzip und mit den gleichen Befehlen wie bei den Knoten, Stäben und äußeren Kräften. Im Unterschied muss sich hier in einem Eingabebefehl aber genau ein Komma befinden.

186–197: Zuletzt, wenn keines der oberen Abfragen zutrifft, wird die Zeile auf andere Zeichenketten untersucht. Vielleicht hat die benutzende Person das Ausrufezeichen für eine Kommentarzeile oder die Anfangsbuchstaben der Befehle vergessen, ein Befehl in mehrere Zeilen geteilt oder sich vertippt. Die Abfrage prüft also, ob die Zeile nicht leer ist und sich nicht einer der Buchstaben *N, E, D, F* am Zeilenanfang befindet. Auch hier wird die Anzahl der kritischen Fehler erhöht und die Benutzer werden mit der Angabe der Zeile in der Ausgabedatei informiert.

204–229: Die Variable `major_faults_count` enthält nun die Anzahl der strukturellen Fehler. Ist die Anzahl größer als null, also liegt ein Fehler vor, soll das Programm gestoppt werden. Die Ursachen für diesen Fehler werden in der Ausgabedatei angegeben, damit die benutzende Person die fehlerhafte Eingabe korrigieren kann. Die Textdatei wird geschlossen und in Matlab wird das Programm mit einem `error()` gestoppt. Der Pfad zur Ausgabedatei wird in den Fehler gedruckt. Die Benutzer werden zudem über das Sichten dieses oder dieser Fehler und das Beenden des Programmes in einem Dialogfeld informiert. Leider ist es nicht möglich den Pfad zur Ausgabedatei in das Dialogfeld zu schreiben. Die Benutzer werden trotzdem über das Dialogfeld informiert, da der manuell erzeugte Fehler mit einem Programmfehler verwechselt werden könnte.

Repetitionsfehler

Nun sollen alle Befehle, die eine exakte Replikation eines anderen Befehls sind, in der Funktion `identical_elements_fault` gelöscht werden. Dazu werden die Element-Variablen `N, E, D, F` und die Variable `input_text_lines` eingegeben. Da die exakte Replikation entfernt und korrigiert wird, werden die Variablen auch wieder ausgegeben. Ausgegeben wird zusätzlich `faults_count`, das die Anzahl der korrigierbaren Fehler zählt. Würde dieser Fehler nicht korrigiert werden, würden weitere Fehler immer doppelt auftauchen. Außerdem könnte eine Berechnung mit zwei identischen Knoten, Stäbe oder Lager nicht durchgeführt werden.

Zeile 238–240: Die Variable `faults_count` zählt die folgenden Fehler und wird zu Beginn auf null gesetzt. Zudem wird die Variable `input_text_lines` in einen Zeichenvektor konvertiert. Einzelne Zeichen in einem `char-array` können schneller und einfacher angesprochen werden.

244–245: Es sollen im folgendem immer zwei Zeilen der Eingabedatei in der Variable `input_text_lines` verglichen werden. Deshalb werden zwei For-Schleifen mit den Laufvariablen `n` und `o` benötigt.

249–261: Als nächstes wird abgefragt, ob alle Zeichen der einen Zeile der anderen Zeile entsprechen: `all(input_text_lines_char(n,:) == input_text_lines_char(m,:))`. Damit nicht gleiche Zeilen verglichen werden, dürfen die Zahlen in der Laufvariable nicht gleich sein. Außerdem dürfen keine leere Zeilen verglichen werden, weil sich auch hier Übereinstimmungen ergeben und Fehler zurückgegeben werden würden. Das Hätte keine weitere Auswirkung für den Verlauf des Programms. Allerdings werden die Benutzer über alle Fehler in der Ausgabedatei informiert. Die Ausgabedatei wäre sehr unübersichtlich.

Trifft die Abfrage zu, wird die Anzahl der Fehler erhöht und die Benutzer werden über den Fehler in der Ausgabedatei informiert. Dort werden auch die Zeilen des Fehlers ausgegeben, die der Laufvariablen der For-Schleifen entsprechen.

264–323: Um den Fehler zu korrigieren muss einer der beiden gleichen Eingabebefehle entfernt werden. Es wurden bereits die Zeilen des Fehlers in der Eingabedatei und der Variable `input_text_lines` gefunden. Besonders wichtiger ist es aber den Fehler in den Element-Variablen `N, E, D` und `F` zu finden und zu korrigieren, da auf den enthaltenen Argumente die Berechnung und Darstellung beruhen. Die folgenden Schleifen dienen dazu die Zeile mit der Replikation in den Element-Variablen zu finden. Es wird zunächst mit einer weiteren For-Schleife, Laufvariable `o`, die Knotenvariable `N` durchlaufen. Aus den einzelnen Argumenten in den drei Zellen wird der Knotenbefehl wieder aufgebaut: `['N,', char(N{1}(o,1)), ',', char(N{2}(o,1)), ',', char(N{3}(o,1))]`. Wenn sich eine Übereinstimmung mit der Zeile der der Eingabedatei ergibt, ist dies einer der identisch vorhandenen Eingabebefehle. Die Argumente in der entsprechenden Zeile der drei Zellen werden gelöscht. Damit treten die Argumente des doppelten Eingabebefehls nur noch einmal in der Element-Variable auf. Damit nicht die Argumente des zweiten doppelten Befehls auch gelöscht werden, wird die übergeordnete For-Schleife mit `break` verlassen.

Ist der doppelt vorhandene Befehl kein Knotenbefehl, wird in der Schleife keine Übereinstimmung gefunden. Im Folgenden wird nun überprüft, ob eine Zeile aus den anderen Element-Variablen dem doppelten Eingabebefehl entspricht. Genauso wie bei den Knoten wird auch bei den anderen Elementen aus den Argumenten wieder ein Eingabebefehl aufgebaut und dann mit der Zeile in der Variable `input_text_lines_char` verglichen. Bei einer Übereinstimmung werden die Argumente in den Zeilen der Element-Variable gelöscht und die For-Schleife verlassen. Bei den äußeren Kräften und den Losalgern muss zudem ein Fehler beachtet werden, der erst in der nächsten Funktion bearbeitet wird. Wird eine Kraft ohne erstes Argument eingegeben mit *F,2,30*, dann wird das zweite Argument in die erste Zelle `F{1}` eingelesen und das dritte Argument in die zweite Zelle `F{2}` kopiert. Das gleiche passiert bei den Lagern auch, wenn das erste Argument fehlt. Deshalb muss der Eingabebefehl aus den Argumenten in den äußeren Kräften in der Element-Variable zudem anders aufgebaut werden: `['F,',char(F{1}(o,1)),',',char(F{2}(o,1))]`. Es finden also bei den äußeren Kräften und Lagern zwei Abfragen statt um diesen Argumentfehler berücksichtigen zu können.

325: Wurde der entsprechende doppelte Befehl in den Element-Variablen gefunden und gelöscht, kann nun auch die entsprechende Zeile in der Variable `input_text_lines` geleert werden. Es wird ein leerer `string-array` in der entsprechenden Zelle erzeugt. Die Zelle wird nicht gelöscht, weil sich sonst die restlichen Zeichenketten nicht mehr in der gleichen Zeile wie in der Eingabedatei befinden.

Argumentfehler

Argumentfehler entstehen, wenn die Benutzer einzelne Argumente im Eingabebefehl vergessen haben, zum Beispiel *N,2,,3* , oder wenn die Benutzer das falsche Format eingegeben haben, zum Beispiel *DX,a* . Diese Fehler sollen in dieser Funktion `argument_fault` korrigiert werden. Eingegeben werden dafür die Element-Variablen, die Dateikennung für die Ausgabedatei, `input_text_lines` und der Dateipfad. Dieser Fehler ist nicht korrigierbar und das Programm stoppt, wenn solch ein Fehler gefunden wird. Es soll die Zeile des Fehlers angegeben werden indem in der Variable `input_text_lines` nach der Zeile gesucht wird, die der Zeile in der Eingabedatei entspricht: `find(contains(input_text_lines,Eingabebefehl));`. Hierzu wird aus den Eingabeargumenten erneut der Eingabebefehl erstellt, zum Beispiel der Knotenbefehl aus Zeile n der Knoten-Element-Variable: `['N,', char(N{1}(n,1)), ',', char(N{2}(n,1)), ',', char(N{3}(n,1))]`.
Durch den Befehl `contains()` entsteht ein logischer Spaltenvektor, indem die Zeile, die diese Zeichenkette enthält, mit eins angegeben wird. Durch `find()` wird dann die Nummer der Zeile mit einer eins zurückgegeben. Diese Befehlskombination wird im folgendem immer benutzt um die Zeile eines Fehlers in der Eingabedatei zu finden.

Zeile 343–482: Jedes Argument in der Knoten-Element-Variable soll auf Vorhandensein und das Format überprüft werden. Deshalb soll jede Zeile in der Variable `N` mit der For-Schleife durchlaufen werden.
Mit `isnan(str2double(N{1}(n,1)))` wird überprüft ob das eingegebene Argument in der ersten Zelle eine Zahl und auch vorhanden ist. Bei der Transformation in eine `double` Zahl mit dem Befehl `str2double` entsteht `NaN`, wenn der `string-array` keine Zahl enthält oder leer ist. `NaN` steht für Not a number und ist ein undefinierbares numerisches Ergebnis. Das Ergebnis kann auf `NaN` mit `isnan()` überprüft werden.

Nun wird die entsprechende Fehlerzeile mit der oben erklärten Befehlskombination in die Variable `faults_line` zurückgegeben. Die Benutzer werden mit dem Fehler und der Zeilenangabe in der Ausgabedatei informiert. Das gleiche Vorgehen wird in den Zellen zwei und drei der Knoten-Element-Variable wiederholt. Auch hier muss die Eingabe eine Zahl sein.
In der letzten Zelle muss zudem überprüft werden, ob mehrere Fehlerzeilen in der Variable `fault_line` zurückgegeben wurden. Wenn zum Beispiel nach dem Knoten *N,1,2,* gesucht werden soll, wird auch die Zeile des Knotens *N,1,2,0* zurückgegeben, weil es die gleiche Zeichenfolge enthält. Gibt es in der Eingabedatei einen zweiten Knotenbefehl mit denselben ersten zwei Argumenten, dann werden beide Zeilen als Fehlerzeile ausgegeben. Es muss also in Zeile 387 überprüft werden, ob die Variable `fault_line` nur eine Zeile enthält. Dann ist die enthaltende Zeile in der Variable die Fehlerzeile und die Benutzer können über den Fehler in der Ausgabedatei informiert werden. Ist mehr als eine Fehlerzeilen vorhanden, dann muss untersucht werden, was die richtige Fehlerzeile ist. Für jede Fehlerzeile wird eine Zeichenkette der jeweiligen Zeile erstellt und in Zeile 399 wird überprüft, ob das letzte Zeichen ein Komma ist. Ist dies der Fall, dann fehlt das letzte Argument und die richtige Fehlerzeile ist gefunden. Die Benutzer können über den Fehler und die Fehlerzeile in der Ausgabedatei informiert werden. Dann wird das gleiche Verfahren auch auf die Stab-Element-Variable angewendet. Auch hier werden alle Zellen nach dem gleichen Schema der Knoten auf Zahlen untersucht.

485–631: Für die Lager und äußeren Kräfte in den Element-Variablen `D` und `F` muss ein etwas anderes Vorgehen angewendet werden. Wenn das erste Argument im Eingabebefehl fehlt, dann werden die Argumente in die falschen Zellen eingelesen, wie bereits im Repetitionsfehler beschrieben. Wird zum Beispiel eine Kraft ohne erstes Argument eingegeben, dann wird das zweite Argument in die erste Zelle `F{1}` eingelesen und das dritte Argument in die zweite Zelle `F{2}` kopiert. Bei einem gleichen Vorgehen wie bei den Knoten würde nicht die richtige Fehlerzeile gefunden werden können. Außerdem würde ein Fehler in dem dritten Eingabeargument angegeben werden, weil die dritte Zelle dann leer ist. Eigentlich fehlt aber das erste Argument. Das Gleiche passiert bei den Lagern auch, wenn das erste Argument fehlt. Deswegen wird bei den Lagern und äußeren Kräften zuerst die Zeile in der Eingabedatei gesucht nach der oben erklärten Befehlskombination.

Die nächste Abfrage prüft ob eine Zeile gefunden wurde. Wenn die Variable `faults_line` noch leer ist, dann hat die benutzende Person wie oben erklärt das erste Argument vergessen. Die Zeichenkette muss umgestellt werden, damit die richtige Zeile gefunden wird.

Es wird nun in Zeile 500–502 überprüft ob kein X, kein Y und auch keine Zahl als Winkel in der ersten Zelle enthalten ist und die zweite Zelle mit einem Argument gefüllt wurde. Ist dies nicht der Fall, dann wurde ein falsches Format eingegeben. Wurde das Argument leer gelassen, dann ist das zweite Argument in die erste Zelle verschoben wurden. Das zweite Zeichen in der Zeichenkette ist dann ein Komma. Wird die Abfrage erfüllt, dann wird die Anzahl der Fehler erhöht und die Benutzer werden in der Ausgabedatei mit dem Fehler und der entsprechenden Fehlerzeile informiert. Die letzte Zelle des Lagers muss eine numerische Eingabe enthalten. Wenn das erste Argument leer gelassen wurde, dann wird das zweite Argument in die erste Zelle eingelesen und die zweite Zelle ist leer. Deshalb darf das zweite Zeichen des Befehls kein Komma sein, wenn die zweite Zelle leer ist oder keinen numerischen Eintrag enthält. Ist das zweite Zeichen ein Komma, muss die erste Zelle auf einen numerischen Eintrag untersucht werden. Eine dieser kombinierten Abfragen in Zeile 517–520 muss stimmen. Auch bei den Lagern kann es vorkommen, dass wenn das letzte Argument fehlt, eine zweites Lager gefunden wird. Dann werden mehr als zwei Fehlerzeilen zurückgegeben. Wie bei den Knoten muss dann auch die richtige Fehlerzeile gefunden werden.

Nach dem gleichen Schema werden auch die Zellen der Element-Variable der äußeren Kräfte untersucht. Im Unterschied zu den Kräften muss das erste Argument ein X oder Y sein. Die Eingabe über einen Winkel ist hier nicht möglich. Außerdem enthält die Element-Variable drei Zellen die untersucht werden müssen. Zelle zwei und drei müssen numerische Einträge besitzen.

636–657: Wurde ein kritischer Fehler gefunden und die Variable `major_faults_count` ist nicht mehr null, dann soll das Programm beendet werden. Eine Berechnung und Darstellung von Elementen, deren Befehl eine fehlendes Argument oder falsches Format besitzt, ist nicht möglich. Die Ursachen für diesen Fehler werden in der Ausgabedatei angegeben, damit die Benutzer die fehlerhafte Eingabe korrigieren können. Die Textdatei wird geschlossen und in Matlab wird das Programm mit einem `error()` gestoppt.

Der Pfad zur Ausgabedatei wird in den Fehler gedruckt. Die Benutzer werden zudem über das Sichten dieses oder dieser Fehler und das Beenden des Programmes in einem Dialogfeld informiert.

Nummerierungsfehler

Nachdem in der letzten Funktion alle Argumente überprüft wurden, ist nun sichergestellt, dass alle Befehle vollständig sind und korrekte Argumente enthalten. In der Funktion `numbering_fault` muss nun die Nummerierung der Knoten überprüft werden. Die Knoten enthalten die Koordinaten zur Positionierung im Koordinatensystem. Über die Knotennummer werden alle weiteren Elemente positioniert. Die eindeutige Nummerierung der Knoten ist sehr wichtig. Keine Knotennummer darf doppelt auftauchen. Außerdem müssen die angegebenen Knotennummern in den anderen Elementen auch existieren. Auch hier wird das Programm gestoppt, da dieser Fehler nicht automatisch korrigiert werden kann.

Zeile 668: Die Variable `major_faults_count` zählt auch hier die Anzahl der kritischen Fehler und wird zunächst auf null gesetzt.

672–700: In der Nächsten Schleife sollen Knoten gefunden werden, die die gleiche Knotennummer besitzen. Gibt es eine Knotennummer zweimal und wird diese Knotennummer in einem anderen Element zur Positionierung angegeben, dann kann sie nicht mehr einem Knoten eindeutig zugeordnet werden. Eine eindeutige Positionierung des Elements ist nicht möglich. Deswegen darf jede Zahl nur einmal an einen Knoten vergeben werden. Dafür müssen nun zwei Knotennummern verglichen werden. Zwei For-Schleifen sind dafür notwendig mit den Laufvariablen `n` und `m`. Diese durchlaufen die Element-Variable `N`. Mit der Abfrage in Zeile 677 wird die Knotennummer der zwei Konten verglichen. Außerdem muss die erste Laufvariable kleiner sein als die zweite. Damit wird sichergestellt, dass keine gleichen Zeilen verglichen werden und jedes Zeilenpaar nur einmal verglichen wird. Wenn sie die gleiche Nummer besitzen liegt ein Nummerierungsfehler vor. Die Anzahl der Fehler wird erhöht und mit der im Argumentfehler erklärten Befehlskombination wird die Fehlerzeile des ersten und des zweiten Knotens ermittelt. Der Fehler und die Fehlerzeile werden in die Ausgabedatei geschrieben.

704–724: Die angegebene Knotennummer in der Element-Variable `E` muss auch bei einem Knoten vorkommen. In der zweiten Zelle befinden sich die Knotennummern, die den Stabbeginnpunkt definieren. Die Knotennummern in der dritten Zelle bestimmen den Stabendpunkt. Jede Zeile enthält zwei Knotennummern, die mit einer For-Schleife durchlaufen werden sollen. Durch eine Abfrage wird überprüft ob die Zahl in der Zelle zwei und drei auch eine Knotennummer aus der Zelle `N1` ist: `any(str2num(char(N{1})) == str2double(E{3} (n,1)))`.
Die Anfangsknotennummer `E{2}(n,1)` und Endknotennummer `E3(n,1)` wird mit jeder Knotennummer aus der Zelle `N{1}` verglichen. Stimmen keine der Knotennummern überein, dann existiert die angegebene Knotennummer im Stab nicht. Die Fehlerzeile wird ermittelt und der Fehler in die Ausgabedatei geschrieben.

727–768: Mit dem gleichen Befehlen und Schleifen werden nun die Knotennummern in den restlichen Elementen untersucht. In der Zelle `F{2}` befinden sich die Knotennummern an denen die äußeren Kräfte wirken. In Zelle `D{2}` befinden sich die Knotennummern zur Positionierung der Lager.

772–791: Auch hier wird das Programm in der gleichen Weise ,wie in den vorherigen Funktionen, mit einem Fehler gestoppt, wenn ein Nummerierungsfehler gefunden wird.

Korrigierbare Fehler

Als nächstes sollen in der Funktion `correction` alle restlichen korrigierbaren Fehler behoben werden. Dazu gehören:

- Zwei Stäbe haben die gleiche Stabnummer.
- Der Winkel im Lager liegt nicht zwischen $-180°$ und $180°$.
- Zwei Elemente liegen übereinander.

Wenn zwei Stäbe die gleiche Stabnummer besitzen, ist dies eigentlich kein Fehler der korrigiert werden muss. Die Benutzer sollen aber informiert werden, weil die Stabnummer zur Identifikation des Stabs im Diagramm und der Ausgabedatei angegeben wird. Auch wenn in der Funktion `identical_elements_fault` identische Elemente gelöscht wurden, können immer noch zwei Elemente übereinanderliegen.

So können zwei Stäbe mit unterschiedlicher Stabnummer gleiche Start- und Endknotennummern enthalten. Hier wird dieser Fehler korrigiert indem ein Element gelöscht wird. Greifen zwei äußere Kräfte am gleichen Knoten an und wirken in die gleiche Richtung, werden die Kraftwerte addiert, bevor eine Kraft gelöscht wird.
In die Funktion wird hier neben den üblichen Variablen auch `faults_count` eingegeben. Diese soll die korrigierbaren Fehler weiter zählen. Ausgegeben werden die Element-Variablen, die Fehlerzahl und `input_text_lines`.

Zeile 805-835: Es wird nach zwei Stäbe gesucht, die die gleiche Stabnummer besitzen. Dafür sind zwei For-Schleifen mit den Laufvariablen `n` und `m` notwendig, um zwei unterschiedliche Stäbe vergleichen zu können. Mit der nächsten Abfrage in Zeile 810 wird überprüft, ob die beiden Stabnummern gleich sind. Dabei muss zudem `n<m` sein. Damit wird sichergestellt, dass nicht zwei gleiche Knoten verglichen werden und ein Knotenpaar nur einmal geprüft wird. Sonst würde ein Fehler zweimal auftauchen, da die Laufvariablen `n` und `m` den Wert des jeweils anderen annehmen können. Die Anzahl der Fehler wird erhöht und die Zeile der beiden Fehler wird ermittelt. Der Fehler und die Fehlerzeile werden daraufhin in die Ausgabedatei geschrieben.

839–884: In der nächsten Schleife sollen zwei Stäbe gefunden werden, die übereinander liegen. Zwei Stäbe liegen übereinander, wenn sie die gleichen Anfangs- und Endknotennummer haben. Damit die Knotennummern in zwei Stäben verglichen werden können werden zwei For-Schleifen benötigt. Wenn die Anfangsknotennummer der beiden Stäben in der Zelle `E{2}` und die Endknotennummern in der Zelle `E{3}` übereinstimmen, dann liegen die Knoten übereinander. Die Stäbe liegen auch übereinander, wenn die Anfangsknotennummer des einen Stabes die Endknotennummer im anderen Stab ist. Der Stab *E,2,4,1* und der Stab *E,4,1,4* liegen also auch übereinander. Deswegen müssen auch die Anfangsknotennummer und die Endknotennummer der Stäbe verglichen werden. Stimmen die Knotennummern überein, dann werden die Zeilen der Stäbe in der Eingabedatei ermittelt und der Fehler in die Ausgabedatei geschrieben. Danach wird der zweite Knoten der übereinanderliegenden Knoten gelöscht. Dafür wird zunächst die entsprechende Zeile in den Zellen geleert. Die Zeile entspricht der Laufvariable `m`. Nach beenden der Schleife werden dann die leeren Zeilen in den Zellen gelöscht, indem die nicht leeren Zeilen mit dem Befehl `E( cellfun('isempty',E))` erneut in die Element-Variable kopiert werden.

Die Zeilen dürfen erst nach den For-Schleifen gelöscht werden, weil sonst die Anzahl der Zeilen nicht mehr mit dem Zielwert der Laufvariable übereinstimmt. Nimmt die Laufvariable den Zielwert an, dann wird eine nicht existierende Zeile in der Element-Variable angesprochen. Das Programm würde einen Fehler melden.

890–941: Im nächsten Schritt sollen Kräfte, die am gleichen Knoten wirken und in die gleiche Richtung zeigen zusammengefasst werden. Später bei der Berechnung wird angenommen, dass an jedem Knoten maximal eine Kraft in eine Richtung wirkt. Deswegen werden die Werte der beiden Kräfte addiert. Die Kräfte werden wie zuvor die Stäbe miteinander verglichen. Bei den Kräften muss die Knotennummer aus der Zelle `F{2}` und die Wirkungsrichtung in der Zelle `F{1}` der beiden Kräfte übereinstimmen. Auch hier werden die Zeilen der Kräfte in der Eingabedatei ermittelt und die Benutzer werden in der Ausgabedatei informiert. Zudem wird der Wert der beiden Kräfte in der Zeile 920 addiert und zurück in die erste Kraft geschrieben. Damit der neue Wert das gleiche Format hat, muss die berechnete Zahl in ein `string-array` und dann in ein `cell` umgewandelt werden: `cellstr(string())`. Die neue Kraft mit den addierten Kraftwerten wird auch in die Ausgabedatei geschrieben. Nun wird die Zeile der zweiten Kraft wie zuvor bei den Stäben geleert und dann gelöscht.

945–977: Die Orientierung der Lager und die blockierte Bewegungsrichtung an einem Knoten kann auch durch einen Winkel angegeben werden (vgl. Abbildung 4.2). Dieser soll zwischen $-180°$ und $180°$ liegen. Für die spätere Berechnung und Darstellung ist dies unerheblich, da dort mit sin() und cos() Funktionen gearbeitet wird. In der nächsten Schleife wird aber nach übereinanderliegenden Lagern gesucht. Deswegen ist es wichtig, dass der Winkel in diesem Intervall liegt. Es wird zunächst geprüft, ob eine numerische Eingabe vorliegt, also ob die blockierte Verschiebung am Knoten durch einen Winkel angegeben wurde. Liegt der Winkel zudem über $180°$ oder unter $-180°$ soll dieser korrigiert werden um in dem angegebenen Bereich zu liegen. Hierzu soll dem Winkel ein Vielfaches von $360°$ abgezogen oder zu dem Winkel ein Vielfaches von $360°$ addiert werden. So bleibt die Ausrichtung des Lager gleich, aber der Winkel liegt im vorgegebenen Intervall.

Wird der Winkel durch 360° geteilt und auf eine ganze Zahl gerundet, weiß man in welchem Intervall der Winkel liegt und wie oft die 360° subtrahiert oder addiert werden müssen:`str2double(D{1}(n,1)) - round(str2double(D{1}(n,1)) / 360) * 360`. Bei einem Winkel von $-830°$ ergibt sich so: $-830°/360° = -2,30\overline{5}$. Wird das Ergebnis gerundet erhält man -2. Der Winkel 360° muss also zweimal abgezogen werden, um ein Winkel von $-180°$ bis 180° zu erhalten: $-830° - (-2) \cdot 360° = 110°$. Die Benutzer werden über die Korrektur in der Ausgabedatei informiert. Der neue berechnete Winkel wird in der Ausgabedatei auch angegeben.

983–1038: In dieser Schleife sollen Lager, die übereinander liegen identifiziert werden. Die Lager müssen also am gleichem Knoten liegen und die gleiche Verschiebungsrichtung sperren. In der Funktion `identical_elements_fault` wurden bereits komplett identische Befehle gelöscht. Die zu sperrende Richtung des Knotens kann aber auch über einen Winkel zur vertikalen Achse angegeben werden (vgl. Kap. 4.1.3). So kann zum Beispiel die zu sperrende Richtung des Lagers in einem Lager mit X und in einem zweiten Lager mit $-90°$ angegeben werden. Die definierten Lager sind die selben, der Befehl ist aber unterschiedlich. Bei allen Lagern muss aber die Knotennummer übereinstimmen. Dies prüft die erste Abfrage. Die folgenden Winkel- und Richtungskombinationen ergeben die gleiche zu sperrende Richtung, obwohl sich die Befehle unterscheiden:

- gleicher Winkel, mind. ein Winkel wurde in der letzten Schleife korrigiert (z.B. *D45,2* und *D405,2*)
- die Winkel liegen im 180° Winkel zueinander (z.B *D30,3* und *D-150,3*)
- ein Winkel liegt bei 180° und ein Winkel bei $-180°$
- eine Richtung wird über X gesperrt, ein Winkel liegt bei 90° oder $-90°$ (z.B. *DX,1* und *D-90,1*)
- eine Richtung wird über `Y` gesperrt, ein Winkel liegt bei 180°, $-180°$ oder 0° (z.B. *D0,5* und *DY,5*)

Liegen zwei Lager in 180° zueinander, dann sperren sie die gleiche Richtung, die Lagerreaktion ist nur umgedreht. Zwei Lager liegen direkt übereinander, wenn sie sich im 360° Winkel zueinander befinden. Dies ist möglich wenn ein Lager mit 180° und das andere Lager mit $-180°$ angegeben wird.

Die restlichen Kombinationen aus Richtung und Winkel sperren auch die gleichen Verschiebungen. Um alle genannten Möglichkeiten abzudecken erfordert dies eine große kombinierte Abfrage. In der ersten Abfrage wird kontrolliert ob unterschiedliche Lager miteinander verglichen werden und die Knotennummern in Zelle zwei übereinstimmen. Die nächste Abfrage in Zeile 998–1004 prüft dann die oben angegebenen Kombinationen. Da durch die zwei For-Schleifen die Laufvariablen sich tauschen, reicht es aus, wenn geprüft wird, ob der Winkel am zweiten Knoten dem ersten Winkel plus 180° oder plus 360° entspricht. Eine Prüfung anders herum, der zweite Knoten ist der Winkel des ersten Knotens plus 180°, ist nicht nötig. Das gleiche gilt für die Kombination aus einer Richtung und einem Winkel. Hier wird geprüft ob ein Winkel 180°, −180° oder 0° ist, indem er durch 180° geteilt wird. Ergibt die Division das gleiche wie wenn die Division auf die nächste ganze Zahl gerundet wird, dann ist der Winkel ein vielfaches von 180° und die gleiche Richtung wird gesperrt: `round(str2double(D{1}(n,1)) / 180) == str2double(D{1}(n,1)) / 180`. Das gleiche Vorgehen ist anwendbar, um zu prüfen ob eine Winkel bei 90° oder −90° liegt. Wie bei den Stäben und äußeren Kräften wird ein Lager entfernt und die benutzende Person in der Ausgabedatei informiert.

1042–1045: Zuletzt werden die Benutzer über die Anzahl der korrigierten Fehler informiert. Durch die Funktion `identical_elements_fault` wurden vielleicht einige Fehler schon korrigiert. Die gesamte Anzahl der korrigierten Fehler aus den beiden Funktionen liegt in der Variable `faults_count`. Ist die Anzahl größer als null werden die Benutzer über die Anzahl der korrigierten Fehler informiert.

Assemblierungsfehler

In der Funktion `assembly_fault` wird der Zusammenbau des Fachwerks überprüft. Wurden die Elemente falsch zueinander positioniert, kann sich zum Beispiel ein bewegliches Fachwerk ergeben, was nicht berechenbar ist. Das Programm wird also gestoppt, wenn ein Assemblierungsfehler gefunden wird. Auf folgende Fehler wird das Fachwerk untersucht:

- Der Start- und Endpunkt eines Stabs ist gleich (z.B *E,2,3,3*).
- Zwei Knoten liegen übereinander (z.B. *N,1,0,2* und *N,3,0,2*).
- An einem Knoten liegen mehr als zwei Lager.
- An einen Knoten greifen weniger als zwei Stäbe oder ein Stab und ein Lager an.

Der Stab hat den selben Start- und Endpunkt, wenn die Knotennummer in Zelle zwei die der in Zelle drei entspricht. Der Stab hat also die Länge null. Einen Stab mit der Länge null kann es nicht geben. Außerdem ist es nicht möglich die Richtung der wirkenden Stabkraft zu definieren. In der Berechnung würde ein Fehler entstehen. Wenn zwei Knoten übereinanderliegen, kann kein richtiges Kräftegleichgewicht in x- oder y-Richtung an diesem Punkt erstellt werden. Eine eindeutige Zuordnung der Stäbe und Lager zu den Knoten ist nicht möglich. Ein Knoten muss entfernt werden. Dies muss aber manuell durch die benutzende Person durchgeführt werden, da die Positionierung anderer Elemente über die Knotennummer definiert wird. Wenn an einem Knoten mehr als zwei Lager angreifen, kann das Fachwerk nicht statisch bestimmt sein. Die Lagerreaktionen können nicht eindeutig bestimmt werden aus den Kräftegleichgewichten. Wenn an einem Knoten weniger als zwei Stäbe oder weniger als ein Stab und ein Lager liegen, dann kann dieser nicht mehr in Ruhe gehalten werden.
Insgesamt kann hier aber nicht ausgeschlossen werden, dass das Fachwerk beweglich ist. Eine Überprüfung aller Möglichkeiten von beweglichen Elementanordnungen wäre zu komplex und ist hier nicht möglich. Bei der Berechnung wird die Koeffizientenmatrix des Gleichungssystems gebildet. Das Gleichungssystem eines beweglichen Fachwerks ist nicht lösbar. Die Koeffizientenmatrix ist singulär. Die Unbeweglichkeit kann also erst nach Bildung der Determinante in dem Unterprogramm `Calculation` ausgeschlossen werden. Durch die bereits durchgeführte Fehleranalyse in den vorangegangenen Funktionen wurden alle Fehler ausgeschlossen oder behoben, die es verhindern das Fachwerk bildlich auszugeben.

Deswegen wird hier, auch wenn ein Assemblierungsfehler gefunden wird, das Fachwerk bildlich ausgegeben bevor es gestoppt wird. Hierzu wird neben den Element-Variablen, der Ausgabedatei, dem Pfad und `input_text_lines` auch der Name der Eingabedatei `input_name` in die Funktion gegeben.

1067–1083: Zunächst wird nach Stäben gesucht, die als Start- und Endpunkt die gleiche Zahl, also die gleiche Knotennummer haben. Ist die Zahl in der Zelle zwei `E{2}` gleich der Zahl in der Zelle drei `E{3}`, dann hat der Stab die Länge null. Die Anzahl der Fehler wird erhöht und die Benutzer werden über den Fehler und die ermittelte Fehlerzeile informiert.

1088–1119: Zwei Knoten liegen übereinander, wenn die x- und y-Koordinaten übereinstimmen. Um zwei Knoten zu vergleichen, werden zwei For-Schleifen mit zwei Laufvariablen `n` und `m` benötigt. Damit nicht ein Knoten mit sich selber verglichen wird und jedes Knotenpaar nur einmal, muss `n<m` sein. Stimmt die x-Koordinate in Zelle `N{2}` und die y-Koordinate in Zelle `N{3}` überein, dann liegen die Knoten übereinander. Die Zeilen der Knoten werden ermittelt und der Fehler und die Fehlerzeile in die Ausgabedatei geschrieben.

1123–1156: Um zu ermitteln, ob mehr als zwei Lager an einem Knoten liegen, werden drei For-Schleifen benötigt mit den Laufvariablen `n, m` und `o`. Wenn die Knotennummer des ersten Lagers gleich der Knotennummer des zweiten Lagers ist und die Knotennummer des ersten Lagers die des dritten Lagers ist, dann haben alle die gleiche Knotennummer und es liegen drei Lager an einem Knoten. Zudem wird bei der Abfrage kontrolliert, dass die Knotennummern unterschiedlicher Lager miteinander verglichen werden. Durch die Voraussetzung `n<m` und `m<o` werden keine gleichen Lager verglichen und jedes Lagertrio wird nur einmal verglichen. Nun werden alle drei Zeilen der Lager in der Eingabedatei ermittelt und der Fehler wird in die Ausgabedatei geschrieben.

1161–1181: Die nächste Schleife und Abfrage soll die Anzahl der Stäbe und Lager an einem Knoten zählen. Hierzu wird ein großer Vektor aus den Knotennummern gebildet, die sich in den Stäben und Lagern befinden: `[str2num(char(E{2})); str2num(char(E{3})); str2num(char(D{2}))]`. Nun wird jede Zahl des Vektors mit der Knotennummer des jeweiligen Knotens verglichen. Die Anzahl der Übereinstimmungen wird mit dem Befehl `sum()` ermittelt.

Ist die Anzahl der angreifenden Lager und Stäbe kleiner als zwei, dann ist der Knoten beweglich. Das Fachwerk kann also nicht kinematisch bestimmt sein.

1186–1210: Wurde ein kritischer Assemblierungsfehler gefunden, dann soll das Programm gestoppt werden. Zunächst wird aber das Unterprogramm `Transfer_to_Double_Matrices.m` und das Programm `Draw_Truss.m` aufgerufen. Diese erstellen eine bildliche Ausgabe des Fachwerks. Diese wird auch im Ausgabeordner gespeichert. Auf die Unterprogramme wird in den folgenden Abschnitten näher eingegangen. Da eine Berechnung mit so einem Fehler aber nicht möglich ist, wird das Programm vorzeitig beendet. Die Benutzer werden in einem Dialogfeld informiert. Daraufhin wird das Programm mit einem `error()` gestoppt.

Statische Bestimmtheit

Zuletzt soll der Grad der statischen Bestimmtheit mit der Funktion `statically_defined` ermittelt werden. Nach Gleichung 3.7 muss der Grad der statischen Bestimmtheit null sein, damit das Fachwerk die Voraussetzung erfüllt, statisch bestimmt zu sein. Das Fachwerk muss dazu noch kinematisch bestimmt sein. Wie oben erwähnt, kann dies aber nicht hier in diesem Unterprogramm vollständig überprüft werden. Ist das Fachwerk überbestimmt (vgl. Gleichung 3.9) oder statisch unterbestimmt (vgl. Gleichung 3.8), dann kann das Fachwerk nicht berechnet werden. Bei einem überbestimmten Fachwerk können die Lagerreaktionen und Stabkräfte nicht alleine aus den Kräftegleichgewichten an den Knoten ermittelt werden. Ist das Fachwerk unterbestimmt, dann ist das Fachwerk beweglich. Das Programm soll gestoppt werden. Allerdings ist auch hier wie bei einem Assemblierungsfehler in der Funktion `assembly_fault` die bildliche Ausgabe des Fachwerks möglich. Deswegen stimmen die Eingabevariablen mit denen der Funktion `assembly_fault` überein.

Zeile 1223: Zunächst wird der Grad der statischen Bestimmtheit nach Gleichung 3.6 ermittelt. Die Höhe der einzelnen Zellen ist die Anzahl des jeweiligen Elements. Der Grad der statischen Bestimmtheit wird in der Variable `j` gespeichert.

1229–1252: Ist der Grad der statischen Bestimmtheit `j` größer als null, dann ist das Fachwerk statisch überbestimmt. Eine Berechnung ist nicht möglich und das Programm wird gestoppt.

Zunächst wird eine bildliche Ausgabe erzeugt. Hierfür werden die Unterprogramme `Transfer_to_Double_Matrices.m` und das Programm `Draw_-Truss.m` ausgeführt. Die Benutzer werden über den Grad der statischen Bestimmtheit in der Ausgabedatei und einem Dialogfeld informiert. Daraufhin wird das Programm mit einem `error()` gestoppt.

1257–1281: Wenn `j` kleiner als null ist, dann ist das Fachwerk statisch unterbestimmt. Auch hier wird das Fachwerk zunächst bildlich ausgegeben, bevor die Benutzer über das unterbestimmte Fachwerk in der Ausgabedatei und einem Dialogfeld informiert werden. Das Programm wird gestoppt.

4.2.5 Übergabe der Fachwerksinformationen in numerische Matrizen

Das Unterprogramm `Transfer_to_Double_Matrices.m` soll die vorhandenen Informationen aus der Eingabedatei in numerische Matrizen überführen. Dazu werden die Argumente der Eingabebefehle, die in den Zellen der Elment-Variablen `N, E, D` und `F` liegen, bearbeitet und in Matrizen kopiert. Die Argumente liegen als `strings` in den `cell-arrays` und sollen in `double` Matrizen kopiert werden. Es soll eine Matrix für jedes Element angelegt werden. Die Namen richten sich nach den englischen Namen des jeweiligen Elements: `Nodes, Rods, Support` und `Forces`. Liegen die Fachwerksinformationen in numerischen Matrizen vor, dann können die einzelnen Zellen einfacher angesprochen werden und besser verarbeitet werden, da die folgenden Befehle und Rechenoperationen mit numerischen Eingabedaten arbeiten. In dem Unterprogramm `Input_from_txt` ist es nicht möglich mit der Funktion `textscan()` die Eingabeargumente in einem anderen Format einzulesen. Dies liegt vor allem an möglichen Fehlern in den Eingabebefehlen. So musste die Befehlskombination so ausgelegt werden, auch fehlerhafte Eingabebefehle einlesen zu können (vgl. Kap. 4.2.2).
In den Element-Matrizen enthält eine Spalte eine bestimmte Information. Jede Zeile enthält ein Element. Die Element-Variablen und die enthaltenen Informationen in den Spalten ist in der Tabelle 4.4 aufgeführt.

Tabelle 4.4: Spalten der Element-Matrizen `Nodes, Rods, Support` und `Forces`

Element	Variable	Zelle 1	Zelle 2	Zelle 3
Knoten	N	Knotennummer	x-Koordinate	y-Koordinate
Stab	E	Stabnummer	Anfangsknotennummer	Endknotennummer
Loslager	D	gesperrte Verschiebungsrichtung	Positionierungs-Knotennumer	n/a
äußere Kraft	F	Wirkungsrichtung	Positionierungs-Knotennumer	Kraftwert

Die enthaltenen Informationen in der Matrix der Knoten entspricht den eingegebenen Argumenten in den Befehlen. Die Informationen werden also aus der Element-Variable `N` in die Matrix `Nodes` kopiert. Jede Zeile in der Matrix enthält also die Informationen eines Knotens. Die restlichen Elemente sollen mit der Angabe des Einheitsvektors in den Matrizen gespeichert werden. Die Komponenten der Einheitsvektoren werden vor allem zur Berechnung benötigt. Zunächst wird aber die Länge der Stäbe berechnet und in Spalte vier geschrieben. Diese wird benötigt um die Einheitsvektoren zu berechnen. Der Einheitsvektor ist der normierte Vektor von Anfangsknoten zu Endknoten. Die x-Komponente und die y-Komponente wird in die Spalte fünf und sechs geschrieben. Die restlichen Informationen werden aus der Element-Variable `E` kopiert. Die Lagerposition in der Matrix `Supports` wird durch die Knotennummer in der ersten Spalte definiert. Die Bewegungsrichtung, die das Lager blockiert, wird durch die Angabe des Einheitsvektors angegeben. Sie entspricht zudem der wirkenden Lagerreaktion am freigeschnittenen Knoten. In die Spalte vier der Lager-Matrix und die Spalte sieben der Stab-Matrix soll später die Lösung aus der Berechnung kopiert werden. Sie sollen also die Stabkräfte und Lagerreaktionen enthalten. Diese bleiben hier noch leer. Die Informationen der äußeren Kräfte sollen in der Matrix `Forces` gespeichert werden. Die erste Spalte enthält die Knotennummer, an der die Kraft wirkt. Die Wirkungsrichtung in den nächsten Spalten wird über den Einheitsvektor definiert. Die letzte Spalte enthält den Kraftwert.
Die Positionierung der einzelnen Elemente erfolgt über die Knoten, die die Koordinaten enthalten. In den Elementen wird ein Knoten über die Knotennummer ausgewählt. Wenn die Nummerierung der Knoten aufeinanderfolgend ist und bei eins beginnt, zum Beispiel*1,2,3,4,...*, dann ist die Knotennummer in den Elementen gleichzeitig die Zeile des Knotens in der Knoten-Matrix. Fehlt eine Knotennummer oder beginnt die Nummerierung nicht bei eins, dann würde die Zeile nicht mehr mit der Knotennummer übereinstimmen, zum Beispiel *2,4,5,...*. Um den richtigen Knoten mit der Knotennummer aus einem Element zu finden, wäre eine Befehlskombination oder Schleife notwendig. Es ist deshalb einfacher einen Knoten über die Zeile anzusprechen und anzugeben. Die Knotennummern in der Stab-Matrix, Lager-Matrix und Kraft-Matrix sollen also über die Zeile der Knoten in der Knoten-Matrix angegeben werden.

Zeile 17–19: Zunächst werden die Knoteninformationen aus der Element-Variable `N` in die Knoten-Matrix kopiert. Jede Zelle in der Element-Variable soll in eine neue Spalte kopiert werden. Die Argumente liegen in einer Zelle als Vektor im Format `string-array`. Mit `str2num` kann der komplette Vektor aus Zeichenketten in einen numerischen Vektor transformiert werden.

Der Befehl `str2double` ist schneller, kann aber keine ganzen Vektoren umformatieren. Die Knoten werden nach der Knotennummer in der ersten Zeile aufsteigend mit dem Befehl `sortrows()` geordnet.

24–26: Auch die Stabinformationen aus den Zellen der Element-Variable `E` werden zunächst auf die gleiche Weise in die Stab-Matrix kopiert. Die Stäbe werden auch aufsteigend nach der Stabnummer geordnet.

31–36: Wie oben beschrieben, soll die angegebene Knotennummer in der Spalte zwei und drei in die Zeile des Knotens mit der Stabnummer in Stab-Matrix geändert werden. Hierzu wird für jeden Stab die Zeile in der Knotenmatrix gesucht, die die Knotennummer enthält. Die Knotennummer eines Knotens in Spalte zwei des Stabes muss mit einer Knotennummer in der ersten Spalte der Knoten übereinstimmen: `Nodes(:,1)==Rods(n,2)`. Es entsteht ein logischer Vektor. Bei einer Übereinstimmung der Knotennummern wird eine eins in die Zeile des Vektors geschrieben. Durch den Befehl `find()` wird dann diese Zeile ausgegeben und zurück in den Knoten geschrieben. Die Nummer in der Spalte für den Anfangsknoten verweist nun auf die Zeile des Knotens in der Knotenmatrix. Dies wird nun auch für den Endknoten wiederholt. Ist die Knotennummerierung aufeinanderfolgend und beginnt mit eins, dann ändern sich die Nummern nicht.

41–56: Als nächstes soll die Stablänge berechnet und der Einheitsvektor gebildet werden. Dieser wird für die Berechnung benötigt (vgl. Kap. 3.1.4). Für jeden Stab wird zunächst die Länge berechnet. Die Stablänge l ist die Länge des Vektors vom Anfangsknoten mit den Koordinaten $[x_1\ y_1]$ zum Endknoten mit den Koordinaten $[x_2\ y_2]$ und wird wie folgt gebildet:

$$l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Da die Nummern in den Spalten zwei und drei in der Stab-Matrix auf die Zeile der Knoten verweisen, können die Koordinaten des Knotens in der Spalte zwei und drei der Knoten-Matrix einfacher ausgewählt werden. Die Zelle der x-Koordinate des Anfangsknoten wird dann mit `Nodes(Rods(n,2),2)` angesprochen. Nun sollen die Einheitsvektoren der Stäbe berechnet werden. Der Einheitsvektor ist der normierte Vektor des Stabes von Anfangsknoten zu Endknoten. Normiert wird der Vektor, indem er durch die Länge geteilt wird. Die Länge eines normierten Vektors beträgt eins.

Die x-Komponente x^e und die y-Komponente y^e wird berechnet:

$$x^e = (x_2 - x_1)/l$$

$$y^e = (y_2 - y_1)/l$$

60: Die Knotennummern zur Positionierung der Lager in der Zelle `D{1}` der Element-Variable wird in die erste Spalte der Lager-Matrix kopiert.

64–87: Die For-Schleife dient dazu für jeden Knoten den Einheitsvektor der Richtung zu Bestimmen, die das Loslager am Knoten sperrt. Zunächst wird in der gleichen Weise, wie bei den Stäben, die Knotennummer geändert, sodass die Nummer in der ersten Spalte die Zeile des Knotens in der Knoten-Matrix angibt. Da die zu sperrende Richtung im Eingabebefehl durch ein X für die x-Richtung , ein Y für die y-Richtung, oder durch den Winkel zur vertikalen Achse angegeben werden kann ist eine Abfrage nötig. Ist die Angabe ein X, dann ist die x-Komponente des Einheitsvektors eins. Die y-Komponente ist null, muss aber nicht in die Zelle kopiert werden. Wenn ein Wert in eine Zelle in einer neuen Spalte geschrieben wird, dann legt Matlab automatisch die restlichen Zellen in der Spalte mit einer null an. Ist das Argument ein Y, dann ist dagegen die y-Komponente eins und wird in die dritte Spalte geschrieben. Wenn die Angabe der Richtung nicht über ein X oder Y definiert wurde, dann enthält die Element-Variable einen Winkel. Die Komponenten des Einheitsvektors können über den Sinus und Kosinus bestimmt werden. Dies ist möglich, weil die Kosinus- und Sinus-Funktion im Einheitskreis mit dem Radius eins definiert sind. Die Hypotenuse im Einheitskreis entspricht dem Einheitsvektor. Hier wird der Winkel zur vertikalen Achse gegen den Uhrzeigersinn angegeben. Die x-Komponente ist so der negative Sinus des Winkels zwischen der vertikalen Achse und dem Einheitsvektor. Die y-Komponente ist der Kosinus. Die Zeichenkette wird durch den Befehl `str2double()` in ein numerisches Format umgewandelt. Die Winkelangabe sollte in Grad erfolgen. Mit `sind()` und `cosd()` wird der Sinus und Kosinus eines Winkels, der in Grad angegeben wurde, bestimmt.

91–92: Zuletzt soll die Matrix der äußeren Kräfte aufgestellt werden. Hierzu wird wie bei den Lagern die Knotennummern, die den Knoten angibt an dem die Kraft wirkt, zunächst in die erst Spalte kopiert. In die vierte Spalte wird der Kraftwert geschrieben. Er kann einfach kopiert werden, nachdem er in das numerische Standartformat, `double`, umgewandelt wurde.

96–111: Die For-Schleife ändert genauso wie bei den Stäben und Lagern die angegebene Knotennummer in die Zeile des Knotens in der Knoten-Matrix um. In Spalte zwei und drei soll der Einheitsvektor der Richtung stehen, in die die Kraft wirkt. Wirkt die Kraft in x-Richtung, angegeben durch ein X in dem Eingabebefehl, dann ist die x-Komponente eins. Wirkt sie dagegen in y-Richtung, angegeben durch ein Y, dann ist die y-Komponente eins. Eine Angabe über den Winkel ist zur Definition einer äußeren Kraft in der Eingabedatei nicht möglich.

4.2.6 Bildliche Ausgabe des Fachwerks

Das nächste Unterprogramm soll das gesamte Fachwerk bildlich ausgeben. Dazu wird in Matlab ein Diagramm erstellt, in das die einzelnen Elemente durch Formen und Linien dargestellt werden. Das Diagramm und dessen Funktionen in Matlab ist vor allem dazu ausgelegt Funktionen und Datenpunkte darzustellen. Hier müssen bestimmte Anweisungen und Befehle benutzt werden, damit bestimmte Formen dargestellt werden, aus denen sich dann das Fachwerk zusammensetzt. Knoten sollen als blaue Punkte, Stäbe als blaue Linien, Kräfte als rote Pfeile und Lager als grüne Dreiecke dargestellt werden. Dazu sollen die Stabnummer, die Knotennummer und der Kraftwert neben das jeweilige Element geschrieben werden. Die einzelnen Elemente sind in der Abbildung 4.3 dargestellt.

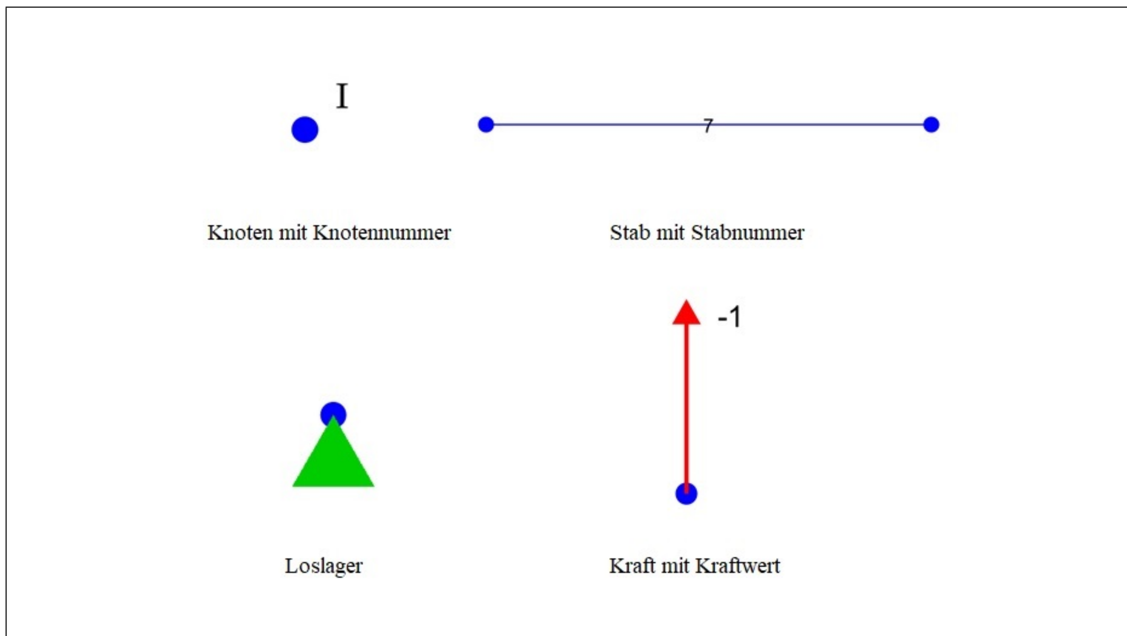


Abbildung 4.3: Darstellung der Knoten, Stäbe, Loslager und äußeren Kräfte im Diagramm

Das Diagramm soll in Matlab selber nicht angezeigt werden. Das Programm soll eigenständig arbeiten und im Hintergrund arbeiten wie eine ‚Engine‘. Deshalb soll die Abbildung genauso wie die Ausgabedatei im Ausgabeordner gespeichert werden. Dort soll die Datei als Vektorgrafik im Format *.svg* abgelegt werde. Durch dieses Format können Ausschnitte beliebig oft vergrößert werden, ohne dass sich die Qualität verschlechtert. Durch die folgenden Befehle im Skript wird die Abbildung erzeugt:

Zeile 10–13: Mit dem ersten Befehl wird eine neue leere Diagrammfläche erzeugt und in der Variable `Truss` gespeichert. Durch die Argumente `'visible','off'` wird das Diagramm für die benutzende Person nicht sichtbar gemacht und liegt damit nur im Hintergrund vor. Das Diagramm soll den Titel des Fachwerks erhalten. Hierzu wird der Befehl `title()` benutzt. Dieser Druckt den angegebenen Text, *Truss of* und dann den Namen der Eingabedatei, über das Fachwerk. Sonderzeichen werden normalerweise als Befehle interpretiert um die Textdarstellung anzupassen. Durch `'Interpreter','none'` wird die Interpretation ausgestellt und zum Beispiel Unterstriche im Namen der Eingabedatei stellen das folgende Zeichen nicht tief.

13–21: In den nächsten Zeilen werden nun alle Einstellungen des Diagramms vorgenommen. Zunächst wird mit `hold on` der Titel des Fachwerks beibehalten wenn neue Elemente hinzugefügt werden. Die weitere Befehle sollen dafür sorgen, dass die Hauptgitterliniern im Diagramm angezeigt werden, der Hintergrund des Diagramms weiß ist und das Diagramm als Vollbild dargestellt und gespeichert wird. Dafür wird die Fenstergröße dem Bildschirm der benutzenden Person angepasst. Beginnend an der unteren Ecke des Bildschirms `[0 0]` soll sich das Fenster bis zur oberen rechten Ecke `[1 1]` erstrecken. Durch `'Units','normalized'` ist die Angabe in Prozent anstatt in Pixel möglich.

25: Die Funktion `elements_into_figure` soll die Knoten, Stäbe, Lager und Kräfte in das Diagramm einfügen. Hierzu werden die Element-Matrizen und das Diagramm in die Funktion eingegeben. Herausgegeben wird das Diagramm mit den enthaltenden Elementen.

42–43: Die Knoten sollen als ausgefüllte Kreise in das Diagramm geschrieben werden. Hierzu wird durch `plot(Nodes(:,2),Nodes(:,3),'o')` an den x- und y-Koordinaten in der zweiten und dritten Spalte der Knoten-Matrix ein Kreis eingefügt. Durch die Marker-Argumente wird die Darstellung der Kreise angepasst. Die Farbe wird auf blau und die Größe auf zehn gesetzt.

47–60: Die Stäbe sollen als Linie gedruckt werden. Dies ist nur mit einer For-Schleife möglich, da jeweils eine Linie pro Stab ausgegeben werden soll. Hierzu werden die x-Koordinaten und dann die y-Koordinaten der Anfangs- und Endknotenpunkte im Befehl `plot()` angegeben. Die Farbe wird auf blau gestellt und die Linienbreite auf eins.

Die Knotennummer soll in die Mitte des Stabes geschrieben werden. Die x- und y-Koordinaten der Mitte des Stabes $[x_m\, y_m]$ wird über den Einheitsvektor $[x^e\ y^e]$ und die Länge l des Stabes berechnet:

$$x_m = x_1 + 0,5 \cdot x^e \cdot l$$

$$y_m = y_1 + 0,5 \cdot y^e \cdot l$$

Die Anfangskoordinaten $[x_1\ y_1]$ befinden sich in der Knoten-Matrix. Der Einheitsvektor und die Länge der Stäbe ist in der Stab-Matrix gespeichert. Der Text wird mittig ausgerichtet und die Größe auf 13 gesetzt.

64–67: Die Kräfte sollen als Pfeil dargestellt werden. Werden mehrere Kräfte eingefügt, dann soll die Länge des Pfeils abhängig von dem Wert der Kraft sein. Der Pfeil der größten Kraft soll halb so lang sein wie die durchschnittliche Stablänge. Der Pfeil jeder weiteren Kraft soll dann mit dem Verhältnis der Kraftwerte angepasst werden. Die maximale Kraft wird in der Variable `maxforce` gespeichert. Es wird der Betrag jedes Kraftwerts ermittelt und durch `max()` wird die größte Kraft ausgegeben. Die Stablänge steht in Spalte vier der Stab-Matrix und durch `mean()` wird der Durchschnittswert berechnet.

72–97: Für jede Kraft soll ein Pfeil in die Richtung, in die die Kraft wirkt, in das Diagramm gedruckt werden. Die Länge des Kraftpfeils wird wie oben beschrieben für die aktuelle Kraft der For-Schleife berechnet und in der Variable `forcel` gespeichert. Steht in der zweiten Spalte der Kraft eine eins für die x-Komponente des Einheitsvektors, dann wirkt die Kraft in x-Richtung. Der Pfeil soll also ausgehend von dem Knoten, an dem sie wirkt, in die x-Richtung zeigen. Hierzu wird zu der zweiten x-Komponente im Befehl `plot()` die Pfeillänge addiert. Durch die Argumente `'r->','MarkerIndices',2,` wird ein rotes Dreieck, dass nach rechts zeigt, am zweiten Datenpunkt erstellt. Eine Kraft zeigt in y-Richtung, wenn die y-Komponente des Einheitsvektors eins ist. Auch hier soll ein Pfeil erstellt werden. Dieser soll in die positive y-Richtung zeigen. Zu der y-Komponente des zweiten Datenpunktes wird die berechnete Pfeillänge addiert. Das Dreieck muss hier nach oben zeigen.

100–108: Durch die Befehle `axis padded` und `axis equal` wird ein Achsenrahmen um das Fachwerk erzeugt. Die gleiche Länge für die Dateneinheiten wird entlang jeder Achse benutzt. Es ist wichtig die Befehle jetzt auszuführen, da im nächsten Schritt die maximale und minimale x-Koordinate ermittelt wird. Durch die Befehle wird verhindert, dass das Fachwerk verzerrt wird, da Matlab die Achsenausschnitte anpasst, sodass das Fachwerk das gesamte Bild ausfüllt.
Durch den Befehl `xlim` wird die minimale und maximale x-Koordinate berechnet. In die Variable `xl` wird dann die Differenz gespeichert. Mit der Variable `xl` soll der Offset der Knotennummerierung zu den Knoten und der Offset der Kraftwerte zu den Pfeilen angepasst werden. Außerdem wird damit die Größe der Dreiecke, mit denen die Lager dargestellt werden ermittelt. Eine Anpassung ist notwendig, da der dargestellte Achsenausschnitt bei jedem Fachwerk anders ist und von den x- und y-Koordinaten, die frei gewählt werden können, abhängt. Wird zum Beispiel nur ein kleiner Achsenausschnitt gezeigt, dann muss die Knotennummer nur um einen kleinen Wert in x- und y-Richtung verschoben werden, damit die Knotennummer neben dem Kreis angezeigt wird.

112–129: Nun sollen die Lager als gleichseitige Dreiecke dargestellt werden. Hierzu wird der Winkel des Einheitsvektors des Lagers zur vertikalen Achse ermittelt und in der Variable `alpha` gespeichert. Dies geschieht mittels des Arkustangens und der Komponenten des Einheitsvektors. Beim Arkustangens muss aber beachtet werden, dass dieser nur Werte im Intervall von $-90°$ bis $90°$ ausgibt. Der Befehl `atan2d()` beachtet die Quadranten des Tangens und gibt den richtigen Winkel zurück. Das Dreieck wird mit dem Befehl `fill()` dargestellt. In dem Befehl werden die drei Koordinaten der Eckpunkte des Dreiecks angegeben. Der erste Punkt liegt auf dem Knoten, die anderen Knoten werden gegen den Uhrzeigersinn angegeben. Die restlichen Punkte werden je nach dem Winkel des Lagers mit dem Radius `xl/30` um den Knoten gedreht. Dies wird in Abbildung 4.4 verdeutlicht. Der Wert `xl/30` hat sich als geeignet erwiesen, sodass die Lager in einer angemessenen Größe dargestellt werden.

So ergeben sich die Koordinate des zweiten Eckpunkts $[x_{2D}\, y_{2D}]$ mit den Koordinaten des ersten Eckpunkts $[x_{1D}\, y_{1D}]$, der auf dem Knoten liegt zu:

$$x_{2D} = x_{1D} - \sin(\alpha + 150°) \cdot xl/30$$

$$y_{2D} = y_{1D} + \cos(\alpha + 150°) \cdot xl/30$$

Der dritte Eckpunkt liegt bei einem gleichseitigen Dreieck $60°$ weiter gegen den Uhrzeigersinn. So ergeben sich die Koordinaten zu:

$$x_{3D} = x_{1D} - \sin(\alpha + 210°) \cdot xl/30$$

$$y_{3D} = y_{1D} + \cos(\alpha + 210°) \cdot xl/30$$

Die Farbe des Dreiecks wird nach dem Schema des RGB-Tripletts in Prozent angegeben. Mit `[0 0.8 0]` wird ein grün erzeugt. Die schwarze Umrandung des Dreiecks wird ausgeblendet.

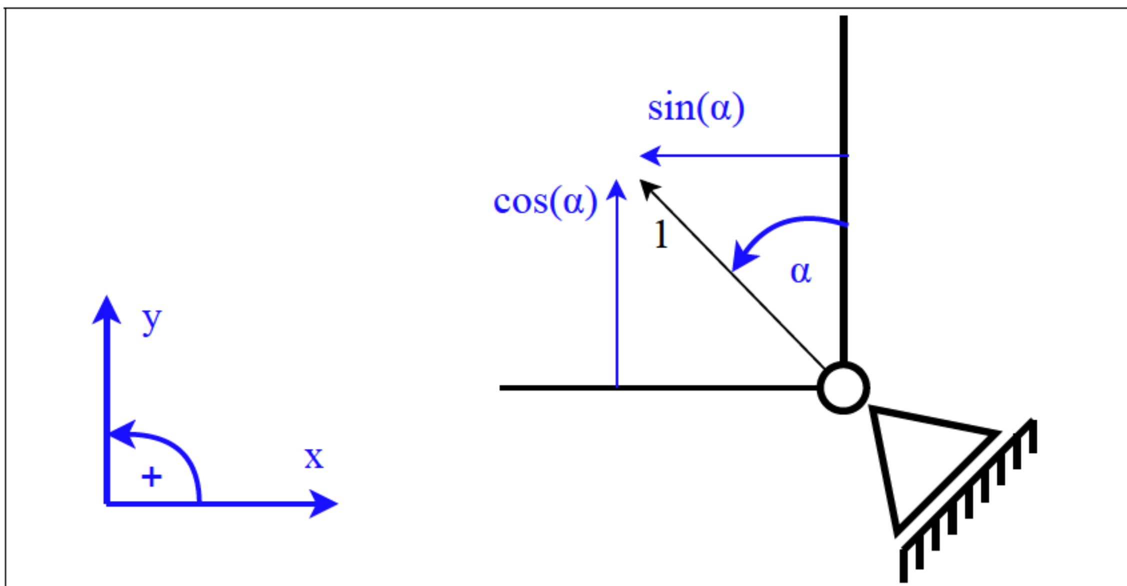


Abbildung 4.4: Geometrische Definition der Eckpunkt eines Dreiecks, durch das ein Lager dargestellt wird

135–141: Im nächsten Schritt wird nun die Knotennummer als römische Zahl rechts über den Knoten geschrieben. Um eine römische Zahl in Matlab darstellen zu können, wird die Latex-Interpretation verwendet. Durch den Latex-Befehl `'\MakeUppercase {\romannumeral', num2str(Nodes(n,1)),'}'` wird eine römische Zahl aus der Knotennummer erzeugt. Die Zahl wird um `xl*0.012` in positive x- und y-Richtung von den Koordinaten des Knotens verschoben. Der Wert hat sich als passend erwiesen, um die Zahl bei unterschiedlichen Bildschirmgrößen nahe neben den Knoten zu schreiben, ohne dass sich diese überschneiden. Es kann nicht vermieden werden, dass ein Stab oder ein Lager durch die Knotennummerierung läuft. Ein Stab kann in jede Richtung laufen. Lager werden aber meistens unter dem Knoten positioniert mit einem Winkel zwischen $90°$ und $-90°$. Die Knotennummern und Kraftwerte werden zum Schluss erzeugt, weil die erzeugten Elemente durch Matlab immer in den Vordergrund am oberen Ende eingefügt werden. So sind diese immer zu sehen. Die Textgröße wird auf 14 gesetzt und der Text wird links ausgerichtet.

146–165: Zuletzt soll der Kraftwert der Kräfte neben die Pfeile geschrieben werden. Bei Kräften, die in x-Richtung wirken, soll der Wert über den Pfeil geschrieben werden und bei Kräften, die in y-Richtung wirken, soll der Wert rechts neben den Pfeil dargestellt werden. Die Schleife hat den gleichen Aufbau, wie die der Kraftpfeile. Als Koordinaten wird die Pfeilspitze der Kräfte angegeben. Von der Pfeilspitze wird der Text bei einem horizontalen Pfeil, Kraft wirkt in x-Richtung, um $0.020 * xl$ in y-Richtung verschoben. Bei einem vertikalen Pfeil wird der Kraftwert um $0.015{*}xl$ in x-Richtung verschoben. Die beiden Werten werden zu der entsprechenden Koordinate der Pfeilspitze addiert. Angezeigt wird der Kraftwert in Spalte vier als Zeichenkette mit der Schriftgröße 14.

28–30: Die Befehle `axis padded` und `axis equal` werden wiederholt, da die Lager und Beschriftungen hinzugefügt wurden. Diese könnten sich jetzt außerhalb des Diagrammbereichs befinden. Durch die Befehle wird ein kleiner Achsenrahmen um das Fachwerk erzeugt und die gleiche Länge für die Dateneinheiten entlang jeder Achse benutzt.

Das ist wichtig, da Matlab sonst die Länge der Dateneinheiten anpasst, sodass das Fachwerk möglichst den gesamten Bildausschnitt ausfüllt. Dadurch werden die Dreiecke verzerrt.

33–35: Das Fachwerk wird als Vektorgrafik gespeichert. Der Speicherort und der Name setzt sich aus dem Pfad, den Namen der Eingabedatei ohne die Textendung und den Anhang *-FIGURE.svg* zusammen. Das Fachwerk wird geschlossen.

4.2.7 Aufstellen und lösen des Gleichungssystems

Im Unterprogramm `Calculation.m` soll das Gleichungssystem des Fachwerks aufgestellt werden. Dafür wird die Koeffizientenmatrix nach Gleichung 3.15 und der äußere Kraftvektor nach Gleichung 3.17 erstellt und assembliert. Dann wird die Determinante der Koeffizientenmatrix ermittelt. In der Fehleranalyse kann nicht ausgeschlossen werden, dass das Fachwerk unbeweglich ist. Ist die Determinante singulär, dann ist das Gleichungssystem nicht lösbar. So ist dies die Prüfung, ob das Fachwerk starr ist und die Lagerreaktionen und Stabkräfte berechnet werden können. Das Gleichungssystem wird dann wie in Gleichung 3.19 aufgelöst. Der Lösungsvektor enthält die Stabkräfte und Lagerreaktionen als skalare Unbekannten. Die Lösung wird außerdem in eine neue Spalte der Lager-Matrix und der Kraft-Matrix gespeichert. So werden die Element-Matrizen aus Tabelle 4.4 vervollständigt. Durch folgende Zeilen im Skript wird das Gleichungssystem aufgestellt:

Zeile 14: Die Funktion `assembly_of_K_f` soll die Koeffizientenmatrix und den externen Kraftvektor erstellen und assemblieren. Hierzu werden die Element-Matrizen eingegeben. Die Funktion gibt dann die Koeffizientenmatrix `K` und den äußeren Kraftvektor `f` zurück.

37–41: Die Größe der Koeffizientenmatrix und die Größe des äußeren Kraftvektors werden vordefiniert. Das beschleunigt die Berechnung, weil sich sonst die Größe der Matrix und des Vektors nach jeder Wiederholung in der folgenden For-Schleife vergrößert. Bestimmt wird die Größe der Matrix und des Vektors durch den Befehl `zeros()`. So enthält jede Zelle zunächst eine Null.

Eine Zeile der Koeffizientenmatrix und im äußeren Kraftvektor entspricht dem Knotengleichgewicht in x- oder y-Richtung an einem Knoten. So ist die Anzahl von Zeilen in `K` und `f` die doppelte Anzahl der Knoten. Eine Spalte bezieht sich auf die skalare unbekannte Kraftgröße eines Stabs oder eines Loslagers. Die Anzahl der Spalten ist die Anzahl der Lager plus die Anzahl der Stäbe (vgl. Kap. 3.1.4).

43–50: Für jeden Knoten sollen zwei Zeilen in der Koeffizientenmatrix für das Kräftegleichgewicht in x- und y-Richtung mit den Informationen der angreifenden Stabkräfte und Lagerreaktionen nach Formel 3.15 assembliert werden. Genauso sollen in den äußeren Kraftvektor die wirkenden Kräfte geschrieben werden. Um für jeden Knoten eine Zeile anzusprechen werden die Variablen `kx` und `ky` definiert. Für den ersten Knoten soll die erste Zeile in der Matrix das Kräftegleichgewicht in x-Richtung und die zweite Zeile das Kräftegleichgewicht in y-Richtung enthalten. Genauso soll die erste Zeile in dem äußeren Kraftvektor die äußere Kraft, die in x-Richtung wirkt, und die zweite Zeile die äußere Kraft, die in y-Richtung wirkt, enthalten. Für den zweiten Knoten ist dann die dritte und vierte Zeile vorbehalten.
Durch $kx = (n-1) \cdot 2 + 1$ wird mit dem aktuellen Knoten n in der For-Schleife die Zeile des Kräftegleichgewichts in x-Richtung angesprochen. Durch $ky = (n-1) \cdot 2 + 2$ erhält man die Zeile in y-Richtung.

52–62: Die ersten Spalten der Koeffizientenmatrix enthalten die Informationen über die Lagerreaktionen. Die Richtung der Lagerreaktion wird so angenommen, dass sie am freigeschnittenen Knoten in Knotenrichtung wirkt. Es soll durch die For-Schleife und die Abfrage geprüft werden, ob das Lager an dem Knoten liegt und eine Lagerreaktion auf den Knoten wirkt. Ist die Zeile des Knotens des aktuellen Lagers in Spalte eins gleich der Laufvariable `n`, dann wirkt die Lagerreaktion an diesem Knoten. Die Komponenten des Einheitsvektors in Spalte zwei und drei werden in die Zeile `kx`, Anteil in x-Richtung, oder `ky`, Anteil in y-Richtung, geschrieben. Die Werte werden in die Spalte `m` geschrieben. Die Laufvariable `m` enthält die aktuelle Zeile des Lagers in der For-Schleife. So werden die Informationen eines Lagers in eine neue Spalte der Koeffizientenmatrix geschrieben.

64–80: Ähnlich werden die Komponenten der Einheitsvektoren der Stäbe assembliert. Die Stabkraft wird als Zugkraft angenommen und wirkt am freigeschnittenen Knoten vom Knoten weg. Hier muss unterschieden werden, ob der Stab an dem Knoten anfängt, dann enthälter die Knotenzeile `n` in der Spalte zwei, oder ob der Stab an dem Knoten aufhört, dann enthält er die Knotenzeile in Spalte drei.
Ist die Knotenzeile in Spalte zwei enthalten, dann ist der Knoten der Anfangsknoten und der Einheitsvektor wird wie bei den Lagern in die entsprechende Zeile der Koeffizientenmatrix geschrieben. Die Einheitsvektoren der Stäbe werden jeweils in eine neue Spalte hinter den Lagern geschrieben. So ist die Spalte in der Koeffizientenmatrix die Anzahl der Lager plus die Zeile des Stabes in der For-Schleife (Laufvariable `m`). Ist der Knoten der übergeordneten For-Schleife der Endknoten des Stabes, dann muss der negative Wert in die Zelle der Matrix geschrieben werden, da der Einheitsvektor von Anfangsknoten zum Endknoten definiert wurde. Die Stabkraft am Endknoten zeigt in die entgegengesetzte Richtung.

83–96: Zum Schluss soll der äußere Kraftvektor nach Gleichung 3.17 assembliert werden mit den Werten der Kräfte. In der Spalte eins der Kraft-Matrix liegt die Zeile des Knotens, an der die Kraft wirkt. Diese muss mit der Laufvariable `n` übereinstimmen. Dann wird überprüft ob die Kraft in x-Richtung wirkt, die x-Komponente des Einheitsvektors in Spalte zwei ist eins, oder in y-Richtung wirkt, die y-Komponente in Spalte drei ist eins. Der negative Kraftwert in Spalte vier wird dann in die entsprechende Zeile des Vektors `f` geschrieben.

18–28: Ist die Koeffizientenmatrix und der äußere Kraftvektor assembliert, dann kann das Gleichungssystem aufgelöst werden. Es wird der Lösungsvektor `a` nach Formel 3.19 berechnet. Dieser enthält nach Gleichung 3.16 die Lagerreaktionen und Stabkräfte, wie die Spalten der Koeffizientenmatrix mit den Informationen gefüllt wurden. Zunächst wird geprüft, ob das Gleichungssystem lösbar ist. Der Lösungsvektor kann berechnet werden, wenn die Determinante nicht singulär ist, also nicht null ist. Dann wird der Kraftvektor `a` nach Formel 3.19 berechnet. Die berechneten Lagerreaktionen in den ersten Zeilen von `a` werden in die vierte Spalte der Lager-Matrix geschrieben. Die Stabkräfte werden in die siebte Spalte der Stab-Matrix kopiert.

4.2.8 Ausgabedatei beschreiben

Im letzten aufgerufenen Unterprogramm `Write_into_Output_File.m` sollen die Elemente und die Lösung in die Ausgabedatei geschrieben werden. Die Elemente werden, wie sie in den einzelnen Element-Matrizen liegen, tabellarisch dargestellt. So wird mit dem Einheitsvektor erkannt, in welche Richtung die ermittelten Stabkräfte und Lagerreaktionen wirken. Um eine weitere Analyse des Fachwerks möglich zu machen und die Berechnung nachvollziehen zu können, wird die Koeffizientenmatrix und der äußere Kraftvektor ausgegeben und in einer weiteren Datei gespeichert. Ist die Determinante der Koeffizientenmatrix null, dann ist das Gleichungssystem nicht lösbar. Die Stabkräfte und Lagerreaktionen konnten im Unterprogramm in Kap. 4.2.7 nicht berechnet werden und können nicht in die Ausgabedatei geschrieben werden. Die Benutzer werden über die singuläre Determinante informiert. Außerdem sollen die Benutzer informiert werden, wenn die Determinante sehr klein ist. Dann wurde das Fachwerk nicht optimal aufgebaut und es ergeben sich hohe Stab- und Lagerkräfte. In Matlab ist leider kein alternativer Befehl oder alternative Variante vorhanden, ein Gleichungssystem auf die ‚lösbarkeit' und ‚Qualität' des Fachwerks zu untersuchen. Die Determinante ist kein genauer Indikator für die Qualität des Aufbaus und gibt nur eine Tendenz an.

Zeile 13: Die Ausgabedatei wird mit dem Zugriffsart append geöffnet. Dadurch werden weitere Textzeilen hinter die Fehleranalyse eingefügt.

17–43: In der nächsten Abfrage wird geprüft, ob die Determinante null oder nahe null ist. Wenn der Betrag der Determinante kleiner als 10^{-10} ist, soll sie auf null gerundet werde und es wird angenommen, dass die Determinante singulär ist. Das Gleichungssystem ist nicht lösbar. Die Benutzer werden unter der zusätzlichen Überschrift Attention in der Ausgabedatei darüber informiert. Das Fachwerk muss angepasst werden, damit es im nächsten Durchlauf des Programms berechnet werden kann. Die Benutzer werden auch informiert, wenn die Determinante kleiner als $0,1$ ist. Das Fachwerk kann möglicherweise angepasst werden, sodass die resultierenden Kräfte im Fachwerk und in den Lagern kleiner sind. Wie oben erwähnt, gibt die Determinante keine genaue Indikation über die Qualität des Fachwerks, sondern nur eine Tendenz an.

46–48: Nun wird die Überschrift Output in die Ausgabedatei geschrieben. Der folgende Inhalt in der Ausgabedatei soll die Informationen der einzelnen Elemente in tabellarischer Form enthalten.

54: Zunächst sollen die Knoten und äußere Kräfte in die Ausgabedatei mit der Funktion `nodes_forces_output` geschrieben werden. Hierzu wird die Knoten- und die Kraft-Element-Matrix sowie die Dateikennung der Ausgabedatei in die Funktion eingegeben. Die Matrizen werden in tabellarischer Form in die Ausgabedatei geschrieben. Alle Informationen über die Knoten und die äußeren Kräfte sind bereits vollständig in der Eingabedatei vorhanden. Die einzelnen Informationen und Daten werden durch das Programm in Matrizen angeordnet und weiter bearbeitet. So wurde die Kraftwirkungsrichtung mit einem Einheitsvektor dargestellt.

96–97: Zunächst werden die Knoten in tabellarischer Form in die Ausgabedatei geschrieben. Die vorhandenen Knoteninformationen sind die Knotennummern, die x-Koordinaten und die y-Koordinaten. Um einzelne Spalten in der Textdatei zu erstellen wird mit dem Tab gearbeitet, der in Matlab durch `\t` eingegeben wird. Durch ihn wird der Abstand zwischen den einzelnen Spalten erzeugt. Zudem muss die Anzahl der Zeichen in den Zeichenfolgen und die Anzahl der Dezimalzahlen definiert werden. Dadurch besitzen die Überschriften und die Zellen darunter die gleiche Breite und werden untereinander angeordnet. Das Format der Überschrift wird mit `%11s\t%11s\t%11s` definiert. Die Spalten haben also eine Breite von elf Zeichen. Die Breite einer Spalte wird durch die Überschrift definiert, da sie die meisten Zeichen enthält.

101–104: Als nächstes werden die einzelnen Spalten mit den Informationen der Knoten durch die For-Schleife gefüllt. Die Informationen eines Knotens aus der Element-Matrix wird in eine neue Zeile in die Tabelle geschrieben. Das Format einer Zeile wird in Matlab mit `%-11.9g\t%-11.9g\t%-11.9g` festgelegt. Eine Zahl wird durch `%-11.9g` definiert. Dadurch werden nur signifikante Stellen angezeigt, ohne nach- oder vorgestellte Nullen. Dadurch ist die Tabelle übersichtlicher. Die maximale Größe der Zahl ist elf Zeichen und die maximale Anzahl an Dezimalstellen beträgt neun Zeichen, weil hier zusätzlich ein Zeichen für das Dezimaltrennzeichen und ein Zeichen für den ganzzahligen Teil abgezogen werden muss. Durch das Minus am Anfang wird die Zahl linksbündig ausgerichtet. Dieses Format und Vorgehen wird im Folgendem immer benutzt um die Zahlen in den Element-Matrizen in Tabellen auszugegeben.

Die Anzahl der Zeichen mit der die Zahlen dargestellt werden muss in den Tabellen angepasst werden, sodass sie zu der Länge der Überschrift und dem Tab passen.

108–116: Als nächstes sollen die Informationen der äußeren Kräfte in eine zweite Tabelle geschrieben werden. Es soll die Knotennummer zur Positionierung, der Einheitsvektor und der Kraftwert in die Tabelle angegeben werden. Die breite der Spalten wird der Länge der Spaltenüberschriften angepasst. Der Einheitsvektor soll in eckigen Klammern geschrieben werden. Also wird um Spalte zwei und drei einer Zeile eine eckige Klammer eingefügt. Die x-Koordinate und die y-Koordinate wird hier durch ein Komma separiert, damit die Zahlen deutlicher getrennt werden. Auch hier werden die Spalten durch einen Tab separiert. Der Tab verschiebt den Positionszeiger in der Ausgabedatei auf den Anfang der nächsten acht Zeichen. Deswegen kann die Länge der Zahlen, also die Anzahl der Zeichen, in den Zeilen größer sein, als in der Überschrift. Der Tab verschiebt den Positionszeiger automatisch wieder an den Anfang der nächsten Spalte.

61: Durch die nächste Funktion `rods_supports_output` sollen dann die Stäbe und Lager in die Ausgabedatei geschrieben werden. Im Unterschied zu den Knoten und äußeren Kräften enthalten die Element-Matrizen die Lösungen der Stabkräfte und Lagerreaktionen. Die Richtungen der Stäbe und der Lagerreaktionen an den Knoten wurde wie bei den äußeren Kräften mit Einheitsvektoren definiert. Eingegeben wird neben der Stab- und Lager-Element-Matrix und der Dateikennung auch die Koeffizientenmatrix.

128–151: Wie oben wird zunächst überprüft, ob der Betrag der Determinante kleiner als 10^{-10} ist. Dann ist die Determinante singulär und das Gleichungssystem konnte im Unterprogramm `Calculation.m` nicht gelöst werden. In die Spalte sieben der Stab-Element-Matrix und in die Spalte vier der Lager-Element-Matrix wurde nicht die Lösung geschrieben. Die Stäbe und Lager sollen also ohne die Stabkräfte und Lagerreaktionen angegeben werden. Die Tabellen werden wie die Tabellen der Knoten und äußeren Kräfte aufgebaut. Die Größe der einzelnen Spalten wird den Überschriften angepasst.

154–176: Wenn das Gleichungssystem gelöst werden konnte und die Determinante nicht null ist, dann enthalten die Element-Matrizen auch die Stabkräfte und Lagerreaktionen. Die zwei Tabellen in der Ausgabedatei werden also um eine Spalte erweitert und die Lösung kann in die Ausgabedatei geschrieben werden.

64–69: Zuletzt sollen die Benutzer in der Ausgabedatei über die Determinante der Koeffizientenmatrix aufgeklärt werden. Die Determinante wird berechnet und ausgegeben. Außerdem werden die Benutzer darüber informiert, dass die Koeffizientenmatrix `K`, der äußere Kraftvektor `f` und der Lösungsvektor `a` im Ausgabeordner gespeichert werden. Dies ermöglicht den Benutzern, die Berechnung und Assemblierung der Matrix und des Vektors nachvollziehen zu können.

75: Wird gewünscht das Gleichungssystem mit der Koeffizientenmatrix `K`, dem äußere Kraftvektor `f` und dem Lösungsvektor `a` in die Ausgabedatei zuschreiben, muss nur die Funktion `equotation_system_output` in dieser Zeile entkommentiert werden. Diese Schreibt das komplette Gleichungssystem $K \cdot a = f$ in die Ausgabedatei. Es stellt alle Zellen gerundet auf drei Nachkommastellen genau dar und gibt Überschriften neben den Spalten und Zeilen der Koeffizientenmatrix an. So ist genau erkennbar welche Spalte sich auf welche Lagerreaktion oder Stabkraft bezieht oder welche Zeile zu welchem Knoten-Kräftegleichgewicht gehört.

77–86: Die Matrix und die Vektoren sollen nun im Ausgabeordner gespeichert werden. Dies erfolgt mit dem Befehl `save()`. Der Dateiname setzt sich aus dem Pfad der Ausgabedatei, dem Namen der Eingabedatei ohne die Textendung und -VAR zusammen. Die Endung steht für Variable. Gespeichert werden die Variablen in einem Matlabformat. Dort werden die Zellen der Matrizen und der Vektoren wie in einer Excel-Tabelle dargestellt. Das Format wird mit der Endung `.mat` definiert. Unterschieden werden muss hier auch, ob das Gleichungssystem lösbar ist. Wenn die Determinante null ist, kann nur die Koeffizientenmatrix und der äußere Kraftvektor gespeichert werden. Ansonsten kann auch der Lösungsvektor ausgegeben werden.

Das Speichern der Variablen nimmt einen großen Teil der Programmlaufzeit ein. Um die Laufzeit des Programms zu verkürzen kann die Schleife auskommentiert werden und stattdessen die Funktion `equotation_system_output` in Zeile 75 entkommentiert werden. Dadurch wird das Gleichungssystem in die Textdatei geschrieben und das Programm arbeitet deutlich schneller. Zuletzt wird die Ausgabedatei wieder geschlossen.

5 Beispiel

Um den Programmablauf und die Entstehung der Variablen und der Ausgabedateien besser nachvollziehen zu können, soll das Programm in Matlab gestartet werden. Ein Beispiel-Fachwerk, dessen Eingabedatei definiert wird, soll berechnet und grafisch ausgegeben werden. Als Beispiel-Fachwerk wird das Fachwerk aus Abbildung 6.12 [Gro+19] verwendet. Dieses ist in Abbildung 5.1 dargestellt.

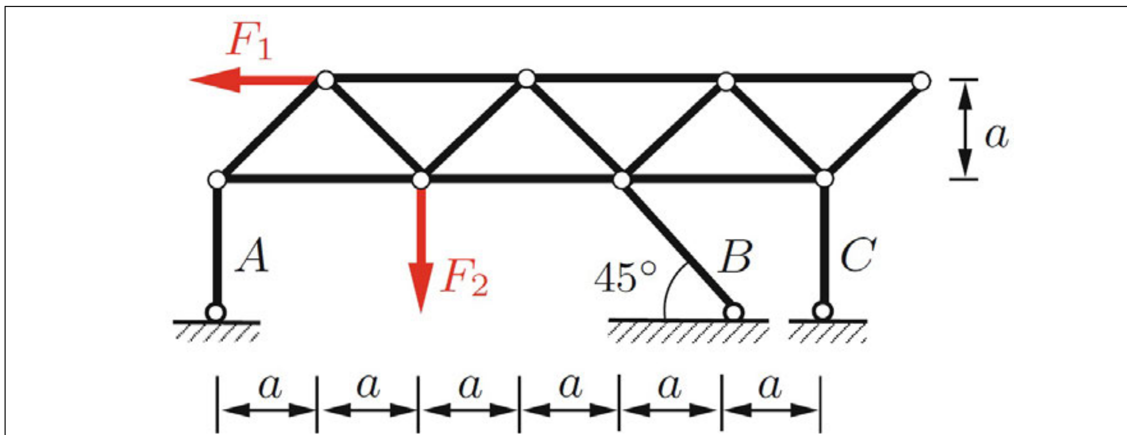


Abbildung 5.1: Darstellung des Beispiel-Fachwerks aus Abbildung 6.12 [Gro+19]

Aus dem dargestellten Fachwerk soll nun die Eingabedatei mit dem Eingabebefehlen verfasst werden. Der Abstand der Knoten ist in dem Fachwerk mit der Variable a angegeben. In der Eingabedatei kann keine Variable definiert werden und so wird diese durch den Wert eins ersetzt. Der Wert dieser Variable ist unerheblich, da sie sich später bei der Berechnung rauskürzen würde. Wichtig ist die Positionierung der Elemente zueinander. Den beiden Kräften wird $F_1 = 2F$ und $F_2 = F$ zugeordnet [Gro+19]. Die Variable muss auch durch den Wert eins ersetzt werden. Die berechneten Stabkräfte und Lagerreaktionen in der Ausgabedatei können dann wieder mit der Variable F multipliziert werden. Das Fachwerk ist durch Stäbe gelagert. Diese sperren genauso wie Loslager die Verschiebung des Knotens in eine Richtung.

Der Koordinatenursprung soll in dem linken unteren Knoten liegen. Die Knoten und Stäbe werden von links nach rechts nummeriert. Die Eingabetextdatei wird dann wie folgt verfasst:

```
! Beispiel Fachwerk aus Abb.6.12 in Gross et al. 2019
N,1,0,0
N,3,2,0
N,5,4,0
N,7,6,0
N,2,1,1
N,4,3,1
N,6,5,1
N,8,7,1
E,1,1,2
E,2,1,3
E,3,2,3
E,4,2,4
E,5,3,4
E,6,3,5
E,7,4,5
E,8,4,6
E,9,5,6
E,10,5,7
E,11,6,7
E,12,6,8
E,13,7,8
E,14,4,3
DY,1
D45,5
D0,7
FX,2,-2
FY,3,-1
SOLVE
!Ersatzelemente
E,5,3,6
N,4,3,2
```

Das Lager B sperrt die Verschiebung am Knoten schräg zur vertikalen und horizontalen Achse. Eingegeben werden muss die Richtung mit dem Winkel zur vertikalen Achse gegen den Uhrzeigersinn. Der Winkel ist dann 45°. Die Ersatzelemente nach dem SOLVE dürfen vom Programm nicht eingelesen werden, da sie hinter dem definierten Ende des Befehlssatzes stehen. Um zu demonstrieren, dass das Programm auch Fehler erkennt wird ein Stab doppelt eingegeben. Der Stab E,5,3,4 und der Stab E,14,4,3 ist gleich. Sie liegen zwischen den gleichen Knoten. Eine Berechnung wäre nicht möglich, da die Stabkräfte am freigeschnittenen Knoten in die gleiche Richtung wirken. Eine eindeutige Zuordnung der Kraftgröße zu einem Stab ist dann nicht mehr möglich.
In der Fehleranalyse können deutlich mehr Fehlerarten erkannt werden. Eine Demonstration aller Fehler, die erkannt werden, ist zu umfangreich und übersteigt den Rahmen dieser Arbeit. Der hier eingebaute Fehler wird vom Programm automatisch korrigiert, sodass das Programm fortgesetzt wird.
Nach dem Einlesen der Informationen im Unterprogramm `Input_from_txt.m` enthalten die Element-Variablen die Argumente der Eingabebefehle. Diese werden in der Abbildung 5.2 dargestellt. Man erkennt, dass alle Argumente korrekt eingelesen wurden. Durch den Befehl `textscan()` wurden die Argumente in Vektoren in `cell-arrays` geschrieben. Dort liegen sie als `string`, also als Zeichenketten, vor. Nach dem Befehl SOLVE wurde das weitere Einlesen der folgenden Zeilen gestoppt, da die zwei Elemente hinter dem Befehl in den Element-Variablen fehlen.

```
N =                    E =                           D =                   F =

  8×3 cell array        14×3 cell array                3×2 cell array         2×3 cell array

    {'1'}    {'0'}        {'1' }    {'1'}    {'2'}       {'Y' }    {'1' }       {'X'}    {'2'}    {'-2'}
    {'3'}    {'2'}        {'2' }    {'1'}    {'3'}       {'45'}    {'3' }       {'Y'}    {'3'}    {'-1'}
    {'5'}    {'4'}        {'3' }    {'2'}    {'3'}       {'0' }    {'4 '}
    {'7'}    {'6'}        {'4' }    {'2'}    {'4'}
    {'2'}    {'1'}        {'5' }    {'3'}    {'4'}
    {'4'}    {'3'}        {'6' }    {'3'}    {'5'}
    {'6'}    {'5'}        {'7' }    {'4'}    {'5'}
    {'8'}    {'7'}        {'8' }    {'4'}    {'6'}
                          {'9' }    {'5'}    {'6'}
                          {'10'}    {'5'}    {'7'}
                          {'11'}    {'6'}    {'7'}
                          {'12'}    {'6'}    {'8'}
                          {'13'}    {'7'}    {'8'}
                          {'14'}    {'4'}    {'3'}
```

Abbildung 5.2: Element-Variablen nach dem Einlesen der Eingabebefehle

Im Unterprogramm `Fault_Analysis.m` werden die identischen Stäbe E,5,3,4 und E,14,4,3 in der Funktion `correction` erkannt. Daraufhin wird der zweite Stab gelöscht. Nach der Fehleranalyse ist dieser nicht mehr in den Element-Variablen, die in Abbildung 5.3 zu sehen sind, enthalten. Die Variable `faults_count` enthält nach der Fehleranalyse den Wert eins, da ein Fehler korrigiert wurde. Die Benutzer werden über den Fehler in der Ausgabedatei informiert.

```
N =

  8×3 cell array

    {'1'}    {'0'}    {'0'}
    {'3'}    {'2'}    {'0'}
    {'5'}    {'4'}    {'0'}
    {'7'}    {'6'}    {'0'}
    {'2'}    {'1'}    {'1'}
    {'4'}    {'3'}    {'1'}
    {'6'}    {'5'}    {'1'}
    {'8'}    {'7'}    {'1'}

E =

  13×3 cell array

    {'1' }    {'1'}    {'2'}
    {'2' }    {'1'}    {'3'}
    {'3' }    {'2'}    {'3'}
    {'4' }    {'2'}    {'4'}
    {'5' }    {'3'}    {'4'}
    {'6' }    {'3'}    {'5'}
    {'7' }    {'4'}    {'5'}
    {'8' }    {'4'}    {'6'}
    {'9' }    {'5'}    {'6'}
    {'10'}    {'5'}    {'7'}
    {'11'}    {'6'}    {'7'}
    {'12'}    {'6'}    {'8'}
    {'13'}    {'7'}    {'8'}

D =

  3×2 cell array

    {'Y' }    {'1'}
    {'45'}    {'3'}
    {'0' }    {'4'}

F =

  2×3 cell array

    {'X'}    {'2'}    {'-2'}
    {'Y'}    {'3'}    {'-1'}
```

Abbildung 5.3: Element-Variablen nach der Fehleranalyse

Nun werden die Element-Matrizen gebildet. Dabei wird die Richtung der Stäbe, der äußeren Kräfte und der Lagerreaktionen mit dem Einheitsvektor angegeben. Dieser wird für die spätere Berechnung und bildliche Ausgabe benötigt. Zudem wird die Länge der Stäbe ermittelt. Die Argumente werden neu angeordnet, sodass die einzelnen Spalten die Information nach Abbildung 4.4 enthalten. Die Matrizen enthalten nur noch numerische Einträge und sind in Matlab im Datentyp `double` gespeichert. Die Element-Matrizen sind in Abbildung 5.4 dargestellt.

```
Nodes =

     1     0     0
     2     1     1
     3     2     0
     4     3     1
     5     4     0
     6     5     1
     7     6     0
     8     7     1

Rods =

    1.0000    1.0000    2.0000    1.4142    0.7071    0.7071
    2.0000    1.0000    3.0000    2.0000    1.0000         0
    3.0000    2.0000    3.0000    1.4142    0.7071   -0.7071
    4.0000    2.0000    4.0000    2.0000    1.0000         0
    5.0000    3.0000    4.0000    1.4142    0.7071    0.7071
    6.0000    3.0000    5.0000    2.0000    1.0000         0
    7.0000    4.0000    5.0000    1.4142    0.7071   -0.7071
    8.0000    4.0000    6.0000    2.0000    1.0000         0
    9.0000    5.0000    6.0000    1.4142    0.7071    0.7071
   10.0000    5.0000    7.0000    2.0000    1.0000         0
   11.0000    6.0000    7.0000    1.4142    0.7071   -0.7071
   12.0000    6.0000    8.0000    2.0000    1.0000         0
   13.0000    7.0000    8.0000    1.4142    0.7071    0.7071

Supports =

    1.0000         0    1.0000
    3.0000   -0.7071    0.7071
    4.0000         0    1.0000

Forces =

     2     1     0    -2
     3     0     1    -1
```

Abbildung 5.4: Numerische Element-Matrizen nach der Übergabe der Eingabeargumente

Alle nötigen Informationen der Elemente sind aufbereitet und vorhanden, um die Zeichnung des Fachwerks zu erstellen und die Berechnung durchzuführen. Die bildliche Ausgabe wird im Unterprogramm `Draw_Truss.m` erstellt. Dort werden Symbole in einem Diagramm angeordnet, sodass sie das Fachwerk darstellen. Das Diagramm wird im Ausgabeordner als Vektorgrafik gespeichert und ist in Abbildung 5.5 zu erkennen. Das Fachwerk wurde in der Ausgabe nicht verzerrt und die Dateneinheiten auf der x- und y-Achse haben die gleiche Länge. Die Knotennummern werden mit römischen Zahlen im Vordergrund angezeigt. Die Kraftpfeile zeigen in die entgegengesetzte Richtung, aber stellen einen negativen Kraftwert dar, der neben den Pfeilen steht. Die Länge des Kraftpfeils mit dem Wert -1 ist halb so lang wie der Kraftpfeil mit dem Wert -2. Das Lager am Knoten $\mathbf{V}$ wird, wie in der Eingabedatei angegeben, als Dreieck, schräg im $45°$ Winkel zur vertikalen Achse dargestellt.

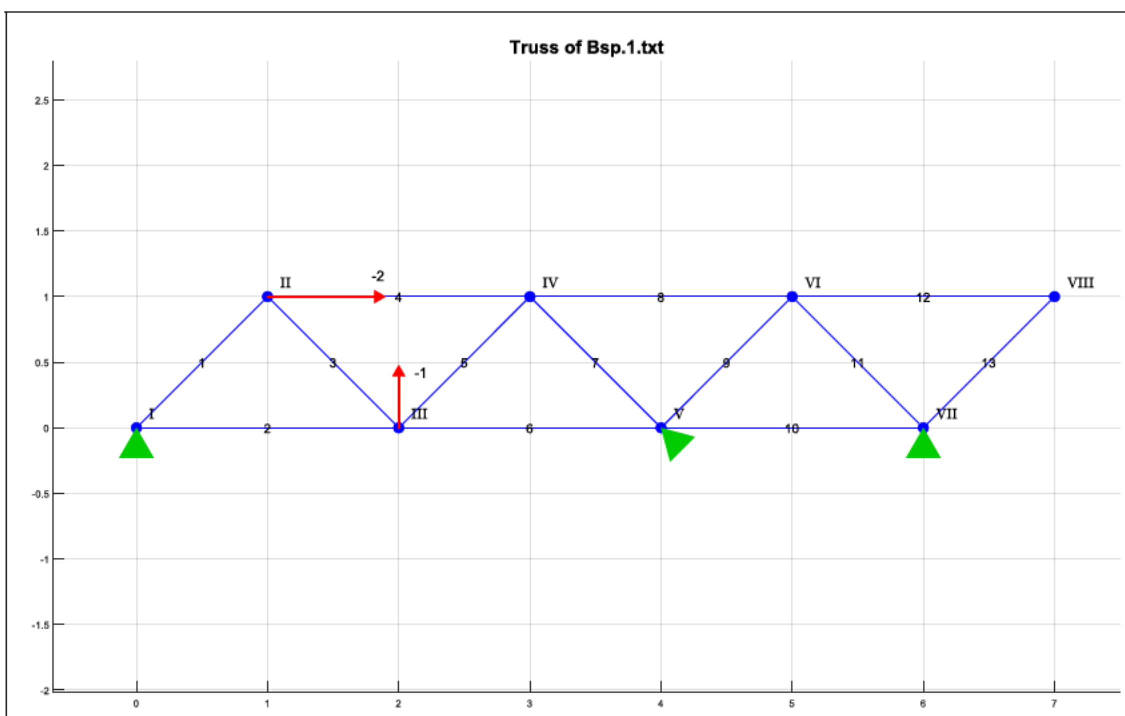


Abbildung 5.5: Bildliche Ausgabe des Beispiels

Die Koeffizientenmatrix wird im Unterprogramm `Calculation.m` erstellt. Dort werden die Zellen der Koeffizientenmatrix nach Gleichung 3.15 und die Zeilen des äußeren Kraftvektors nach Gleichung 3.17 assembliert. Die ersten drei Spalten beziehen sich auf die Lager und die resultierenden Lagerreaktionen an den freigeschnittenen Knoten. Das erste Lager sperrt die y-Verschiebung des ersten Knotens.

Das Kräftegleichgewicht des ersten Knotens in y-Richtung wird in der zweiten Zeile erstellt. Da die Lagerreaktion in y-Richtung wirkt, wird in die Zelle eine eins geschrieben. Dies ist die Komponente des Einheitsvektors in die y-Richtung. Am Knoten eins greifen zudem die Stäbe eins und zwei an. Stab zwei verläuft in x-Richtung und so wird eine eins, die Komponente des Einheitsvektors in x-Richtung, in die erste Zeile und die fünfte Spalte geschrieben. Die Stabkraft des ersten Stabs wirkt dagegen schräg im 45° Winkel. Die Komponenten in die x- und y-Richtung sind deswegen $\sqrt{2}/2$. Die Stabkraft muss im Kräftegleichgewicht an Knoten eins in x- und y-Richtung berücksichtigt werden. Die Komponenten des Einheitsvektors werden in die erste und zweite Zeile der vierten Spalte geschrieben. Die gesamte Koeffizientenmatrix wird mit den anderen Ausgabedateien im Ordner gespeichert und ist in Abbildung 5.6 zusammen mit dem äußeren Kraftvektor zu sehen.

Node	Supp.1	Supp.2	Supp.3	Rod1	Rod2	Rod3	Rod4	Rod5	Rod6	Rod7	Rod8	Rod9	Rod10	Rod11	Rod12	Rod13	Solution	Force
1X:	0	0	0	0.707	1	0	0	0	0	0	0	0	0	0	0	0		0
1Y:	1	0	0	0.707	0	0	0	0	0	0	0	0	0	0	0	0		0
2X:	0	0	0	-0.707	0	0.707	1	0	0	0	0	0	0	0	0	0		2
2Y:	0	0	0	-0.707	0	-0.707	0	0	0	0	0	0	0	0	0	0		0
3X:	0	0	0	0	-1	-0.707	0	0.707	1	0	0	0	0	0	0	0		0
3Y:	0	0	0	0	-0	0.707	0	0.707	0	0	0	0	0	0	0	0		1
4X:	0	0	0	0	0	0	-1	-0.707	0	0.707	1	0	0	0	0	0		0
4Y:	0	0	0	0	0	0	-0	-0.707	0	-0.707	0	0	0	0	0	0	* a =	0
5X:	0	-0.707	0	0	0	0	0	0	-1	-0.707	0	0.707	1	0	0	0		0
5Y:	0	0.707	0	0	0	0	0	0	-0	0.707	0	0.707	0	0	0	0		0
6X:	0	0	0	0	0	0	0	0	0	0	-1	-0.707	0	0.707	1	0		0
6Y:	0	0	0	0	0	0	0	0	0	0	-0	-0.707	0	-0.707	0	0		0
7X:	0	0	-0	0	0	0	0	0	0	0	0	0	-1	-0.707	0	0.707		0
7Y:	0	0	1	0	0	0	0	0	0	0	0	0	-0	0.707	0	0.707		0
8X:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-0.707		0
8Y:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0	-0.707		0

Abbildung 5.6: Gleichungssystem mit assemblierter Koeffizientenmatix und äußeren Kraftvektor

Das Gleichungssystem wird aufgelöst und ergibt den Lösungsvektor. Dieser enthält in den ersten drei Zeilen die Lagerreaktionen und darunter die Stabkräfte. Die Anordnung ist die gleiche wie die Spalten der Koeffizientenmatrix in Abbildung 5.6. Nach dem erfolgreichen Durchlaufen des Programms erscheint eine Dialogbox (Abbildung 5.7). Diese informiert über den erfolgreichen Durchlauf des Programms und den Abschluss der Berechnungen. Der Link zum Ordner, der alle Ausgabedateien enthält, wird in das Command-Window geschrieben.

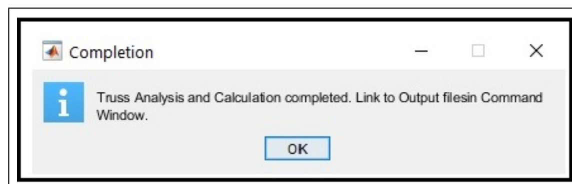


Abbildung 5.7: Dialogbox informiert über das Abschließen des Programms

Die berechnete Lösung wird zusammen mit den restlichen Informationen, enthalten in den Element-Matrizen, in die Ausgabetextdatei geschrieben. Dabei werden die Matrizen als Tabellen ausgegeben. Zuvor wurde die Ausgabetextdatei immer wieder in den Unterprogrammen geöffnet und mit den Eingabebefehlen und den gefundenen Fehlern beschrieben. Die gesamte Ausgabedatei ist in Abbildung 5.8 und Abbildung 5.9 dargestellt und enthält die Lösung der Stabkräfte und der Lagerreaktionen in der letzten Spalte der Elemente. Die Lösungen stimmen mit den Angaben zu den Stabkräften und Lagerreaktionen auf Seite 161 überein [Gro+19]. Dort werden aber nur die gerundeten Werte für die Stabkräfte angegeben.

```
******************************************************************
                    2D-Truss Analysis Software
           Calculation of Forces with the Node Methode
                   Version: 1.0 Date: 25.06.2021
                   Bachelor Thesis, Max Stegmann
HAW Hamburg, Department of Automotive andAeronautical Engineering
******************************************************************
Truss (Input file): Bsp.1.txt
path: C:\Users\maxjo\OneDrive\Desktop\Studium\Bachelora
      arbeit\Matlab\Truss_Analysis_Calculation_Software\

                         Input Commands:
******************************************************************
N,1,0,0
N,3,2,0
N,5,4,0
N,7,6,0
N,2,1,1
N,4,3,1
N,6,5,1
N,8,7,1
E,1,1,2
E,2,1,3
E,3,2,3
E,4,2,4
E,5,3,4
E,6,3,5
E,7,4,5
E,8,4,6
E,9,5,6
E,10,5,7
E,11,6,7
E,12,6,8
E,13,7,8
E,14,4,3
DY,1
D45,5
D0,7
FX,2,-2
FY,3,-1

                          Fault Analysis
******************************************************************
Line 14 and 23 in input text file: Truss assembly fault detected!
Two Rods with the same start and end node numbers (on top of each other):
       E,5,3,4  E,14,4,3
Rod E,14,4,3 is deleted.

1 minor Fault(s) were corrected.

                              OUTPUT
*********************************************************************
Node Coordinates
Node Number x-Coordinate      y-Coordinate
1           0           0
2           1           1
3           2           0
4           3           1
5           4           0
```

Abbildung 5.8: Seite eins der generierten Ausgabetextdatei aus dem eingegebenen Befehlsdeck durch das Programm

```
6           5           1
7           6           0
8           7           1

External Forces
At Node          Direction[x,y]          Value
2           [      1,       0]      -2
3           [      0,       1]      -1

Support Reactions
At Node          Direction[x,y]          Reaction
1           [      0,       1]      +1.66667
5           [-0.7071,  0.7071]      -2.82843
7           [     -0,       1]      +1.33333

Rod-Forces
Rod Number  Start-Node  End-Node    Direction[x,y]          Force
1           1           2           [ 0.7071,  0.7071]      -2.35702
2           1           3           [      1,       0]      +1.66667
3           2           3           [ 0.7071, -0.7071]      +2.35702
4           2           4           [      1,       0]      -1.33333
5           3           4           [ 0.7071,  0.7071]      -0.942809
6           3           5           [      1,       0]      +4
7           4           5           [ 0.7071, -0.7071]      +0.942809
8           4           6           [      1,       0]      -2.66667
9           5           6           [ 0.7071,  0.7071]      +1.88562
10          5           7           [      1,       0]      +1.33333
11          6           7           [ 0.7071, -0.7071]      -1.88562
12          6           8           [      1,       0]      -0
13          7           8           [ 0.7071,  0.7071]      -0

The system of equations K*a=f can be solved unambiguously if the
determinant of the coefficient matrix is not equal to zero:
det(K)=-3.750000e-01

The coefficient matrix K, the external force vector f and the
solution a (when system of equations is solveable) was saved
 as a -.mat file to the folder.
```

Abbildung 5.9: Seite zwei der generierten Ausgabetextdatei aus dem eingegebenen Befehlsdeck durch das Programm

6 Ausblick

Mit diesem Programm wird die automatisierte Berechnung und Darstellung von Fachwerken beliebiger Größe ermöglicht. Über die Eingabedatei können individuell erstellte Fachwerke definiert werden. Viel mehr zeigt das Programm aber, wie viel Vorbereitung notwendig ist, um mit manuellen Eingabebefehlen als Grundlage zu arbeiten. Die ersten vier Unterprogramme beschäftigen sich ausschließlich mit dem Einlesen, der Analyse und der Aufarbeitung der Eingabeinformationen. Erst dann erfolgt die Darstellung, die Berechnung und die Ausgabe der Lösung. Ein Computerprogramm kann immer nur Informationen in einer fest definierten Form einlesen und ist nicht variabel. So ist es wichtig, dass das Format der Eingabedatei und der Eingabebefehle immer eingehalten wird. Besonders wenn davon ausgegangen werden muss, dass die Eingabedatei fehlerhafte Eingabebefehle enthalten kann, ist eine umfangreiche Fehleranalyse notwendig um verschiedene Arten von Fehlern erkennen zu können. Die Fehleranalyse ist hier bereits mehr als 1200 Zeilen lang, trotzdem entdeckt sie noch nicht alle möglichen auftretenden Fehler. Es können noch Schleifen ergänzt werden um die Beweglichkeit des Fachwerks zu prüfen. Wenn zum Beispiel an einen Knoten nur zwei Stäben angrenzen, dessen Einheitsvektor in die gleiche oder entgegengesetzte Richtung zeigen, dann kann der Knoten infinitesimal kleine Bewegung senkrecht zu den Einheitsvektoren ausführen. Der Knoten wäre beweglich und das Gleichungssystem nicht lösbar. Um den Fokus aber nicht zu stark auf die Fehleranalyse zu legen wurde an dieser Stelle darauf verzichtet eine weitere gezielte Untersuchung auf die kinematische Bestimmtheit des Fachwerks durchzuführen.

Das Programm stellt die Basis zur Berechnungen von Fachwerken dar. Ein ebenes, starres und statisch bestimmtes Fachwerk ist das am einfachsten zu berechnende Fachwerk. Durch den modularen Aufbau können aber nun weitere Funktionen und Unterprogramme hinzugefügt oder bereits vorhandene ergänzt werden, um zum Beispiel auch dreidimensionale oder überbestimmte Fachwerke berechnen zu können. Zudem ist ein modularer Aufbau sehr nützlich, wenn bestimmte Funktionen oder Teile eines Programmes mehrfach oder an verschiedenen Stellen ausgeführt werden sollen. Dies zeigt die Fehleranalyse.

Dort werden in den Funktionen `statically_defined` und `assembly_fault` am Schluss die Unterprogramme `Transfer_to_Double_Matrices.m` und `Draw_Truss.m` aufgerufen. Dadurch wird der Rest des Programms ausgelassen und nur diese Unterprogramme werden ausgeführt. So kann das Fachwerk trotzdem noch dargestellt werden und es muss dafür nicht die gesamte Befehlskette wiederholt werden. Es müssen nur die zwei ‚Module' gestartet werden.

Mit der Übergabe des Programms in den Besitz der HAW-Hamburg, soll es den Studierenden in technischen Studiengängen helfen und motivieren, sich mit der Datenverarbeitung zu beschäftigen und sich für diese zu begeistern. Die Arbeit eines\einer Ingenieurs:in findet zum größtem Teil am Computer statt und oft müssen eine große Menge an Daten verarbeitet werden. Abläufe eines Vorgehens werden häufig automatisiert. Bei anderen ist dies sogar notwendig, da die Menge der Daten zu groß ist. Um zu verstehen, wie Mechanik-Programme arbeiten, und um selber einzelne Abfolgen automatisieren zu können, ist es wichtig, dass sich Studierende mit der Datenverarbeitung beschäftigen. Dieses Programm soll den Studierenden die Verknüpfung der zwei Disziplinen aufzeigen und eine Vorgehensweise in den verschiedenen Unterprogrammen darlegen. Ein Unterprogramm erfüllt eine bestimmte Funktion und so kann das Verfahren auch auf andere mechanische Probleme angewendet werden.

Literatur

[ANS21] Inc. ANSYS. *ANSYS*. Version 6.2.0. 2021. URL: https://www.ansys.com. Aufruf: 30.06.2021.

[Eat88] J. Eaton. *GNU Octave*. Version 6.2.0. 1988. URL: https://www.gnu.org/software/octave/. Aufruf: 30.06.2021.

[Föp12] A. Föppl. *Einführung in die Mechanik*. 2. Aufl. Vorlesung über Technische Mechanik. Technische Hochschule München: B. G. Teubner Verlag, 1912. ISBN: 978-3-642-98431-0.

[Gro+19] D. Gross et al. *Technische Mechanik 1*. 14. Aufl. Statik. Darmstadt und München: Springer Vieweg, 2019. ISBN: 978-3-662-59156-7.

[KW17] K. Knothe und H. Wessels. *Finite Elemente*. 5. Aufl. Eine Einführung für Ingenieure. Berlin: Springer Vieweg, 2017. ISBN: 978-3-662-49351-9.

[Lor24] H. Lorenz. *Technische Mechanik starrer Gebilde*. 3. Aufl. Lehrbuch der Technischen Physik. Technische Hochschule Danzig: Springer-Verlag Berlin Heidelberg, 1924. ISBN: 978-3-642-98431-0.

[Mat21] MathWorks. *MATLAB*. Version R2021a. 2021. URL: https://de.mathworks.com/. Aufruf: 30.06.2021.

[Sza75] I. Szabo. *Einführung in die Technische Mechanik*. 8. Aufl. Technische Universität Berlin: Springer-Verlag Berlin Heidelberg, 1975. ISBN: 3540442480.

A Matlab Skript

D:\Truss Analysis Cal...\Truss Analysis Calculation.m 1 of 3

```
1  %% %2D-Truss Analysis and Calclulation Software
2  %Bachelor Thesis, Max Stegmann, Version: 1.0 Date: 25.06.2021
3  %HAW Hamburg, Department of Automotive and Aeronautical Engineering
4  %property of HAW Hamburg
5
6  %% %Instructions
7      % 1. To run the Programm store the files on your local drive.
8      % 2. Unzip the folder TAC_Sub_Programms.zip to TAC_Sub_Programms
9      % 3. Open the root programme Truss_Analysis_Calculation.m
10     % 4. On macOS (apple) you may need to grant matlab permission to
11     %   access your local folders
12     % 5. Run the programme
13     % 6. You need to accept to change or add the current folder
14     %   in matlab
15     % 7. Select the input text file containing the truss informations
16     %   from the windows dialog
17     % 8. The solution/output file is stored in the same folder as your
18     %   input file
19
20 %% %Attention
21     % 1. Make sure the sub folder TAC_Sub_Programms is stored in the same
22     %   folder as the root programme Truss_Analysis_Calculation.m
23     % 2. Make sure there is not another sub folder in the sub folder
24     %   TAC_Sub_Programms, sub programms need to be located one
25     %   level behind the root programme
26     % 3. Don't change the name of the folder TAC_Sub_Programms or any
27     %   sub programme in this folder!
28     % 4. Path name of input file + name of input file must not exceed
29     %   270 characters
30     % 5. The name of the path and the input file should not contain umlauts
31     %    and special characters except for underscores.
32     % 6. Detailed informations on the commands to create the truss elements
33     %    can be found in the file: Input_Commands.pdf
34
35 %% %Important Variables
36     % Input: fileID of opened input text file
37     % Output: fileID of opened output text file
38     % input_name: name of the input text file
39     % output_name: name of the output text file
40     % folder_path: path of the folder of the input text file
41     %
42     % N: cell array containing the read in node informations from input
43     %       file 1.cell: node number; 2.cell: x-coordinate; 3.cell:
44     %       y-coordinate
45     % E: cell array containing the read in rod informations from input file
46     %       1.cell: rod number; 2.cell: beginning node; 3.cell: end node
47     % D: cell array containing the read in support informations from input
48     %       file 1.cell: locked direction; 2.cell: node location
49     % F: cell array containing the read in external force informations from
50     %       inputfile 1.cell: locked direction; 2.cell: node location,
51     %       3.cell: value
52     % input_text_lines: cell vector containing the text from the input file
53     %       every line of the text file is stored in a new row
```

```
 54      %
 55      % Nodes: numeric (double) matrix containing the node information
 56      %       1.column: node number (from N); 2.column: x-coordinate(from N);
 57      %       3.column: y-coordinate (from N)
 58      % Rods: numeric (double) matrix containing the rod information
 59      %       1.column: rod number(from E); 2.column: beginning node(from E);
 60      %       3.column: end node(from E); 4.column: calculated length of the
 61      %       rod 5.column: calculated x component of the unit vector
 62      %       6.column: calculated y component of the unit vector 7.column:
 63      %       copied solution for the rod force from a
 64      % Supports: numeric (double) matrix containing the support information
 65      %       1.column: node location (from D); 2.column: calculated x
 66      %       component of the unit vector 3.column: calculated y component
 67      %       of the unit vector 4.column: copied solution for the support
 68      %       reaction from a
 69      % Forces: numeric (double) matrix containing the external force
 70      %       information 1.column: node location (from F) 2.column:
 71      %       calculated x component of the unit vector 3.column: calculated
 72      %       y component of the unit vector 4.column: force value
 73      %
 74      % K: koefficient matrix one line corresponds to the balance of forces
 75      %       of one node in x- or in y-direction, on row corresponds to one
 76      %       rod force and to one support reaction
 77      % f: external forces vector
 78      % a: vector a containing the solution(rod forces and support reactions)
 79      %       of the linear system of equations K*a=f
 80      %
 81      % Truss: figure containing the visualisation of the Truss with the
 82      %       elements: nodes, rods, supports and external forces
 83      %
 84      % n;m;o: for-loop indices fault_line(-_2/-_3): line of the fault in the
 85      % input text file
 86      %       that was found
 87      % faults_count: counts number of the minor faults that are corrected
 88      % major_faults_count: counts number of the major faults -> further
 89      %       processing stops
 90      % j: degree of static (in-)determinacy of the truss
 91
 92  %%
 93  clear
 94  clf
 95  clc
 96  close all
 97
 98  %titel
 99  disp('*******************************************************************');
100  disp('                    <strong>2D-Truss Analysis Software</strong>');
101  disp(['           ',...
102      '<strong>Calculation of Forces with the Node Methode</strong>']);
103  disp('                   Version: 1.0 Date: 25.06.2021');
104  disp('                   Bachelor Thesis, Max Stegmann');
105  disp('HAW Hamburg, Department of Automotive and Aeronautical Engineering');
106  disp('*******************************************************************');
```

```
107 fprintf('\n');
108
109 %add the path with the subscripts
110 addpath('TAC_Sub_Programms')
111
112 %sub programme to return the name and path of the input text file
113     %reads in the input informations and every line
114     %returned: input_name, folder_path, N, E, D, F, input_text_lines, Input
115 run('Input_from_txt.m')
116
117 %sub programme to create a output text file and print the title and input
118     %informations into the output text files
119     %returned: output_name, Output
120 run('Creation_of_Output_File.m')
121
122 %sub programme to check the truss informations for faults
123     %faults are written to the output file with the fault line of the input
124     %text file, further processing is stopped if a major fault is found
125     %faults are corrected if possible
126     %returned: corrected N, E, D and F; other variables are for internal
127     %use
128 run('Fault_Analysis.m')
129
130 %sub programme to transfer the input informations from
131     %cell arrays into two dimensional double matrices
132 run('Transfer_to_Double_Matrices.m')
133
134 %sub programme to draw the Truss with all Nodes, Rods, Support and
135     %Forces into an diagramme and saves it as a svg file
136 run('Draw_Truss.m')
137
138 %sub programme to set up the Truss Matrix with one equotation for every
139     %Node and then solves the equotation system, returning the solution for
140     %Rod Forces and Support Reactions
141 run('Calculation.m')
142
143 %sub programme to print the solution and the coefficient matrix into
144     %the output text file, Variables K,a,f are saved to the folder
145 run('Write_into_Ouput_File.m');
146
147 %print completion and path of output files into command window
148 msgbox(['Truss Analysis and Calculation completed. Link to Output files'...
149     'in Command Window.'],'Completion','help');
150 disp('<strong>Truss Analysis and Calculation completed.</strong>');
151 disp(['Link to Output files: ','<a href="',folder_path,'">path</a>']);
```

```
1 %% %2D-Truss Analysis and Calclulation Software
2 %Bachelor Thesis, Max Stegmann
3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
4
5 %% %sub programme: Input_from_txt
6     %the sub programme provides the input text file name and the
7     %path from the user input and checks whether one text file was selected
8     %the textfile with the truss informations is opened and the
9     %informations are transfered into cell arrays
10    %sub programme to read in the Truss informations
11
12 %uigetfile opens a windows dialogue, the user can select one text file
13    %the name and the path is returned into input_name & folder_path
14 [input_name,folder_path]=uigetfile('*.txt','Open text input file!');
15
16 %checks if the user cancelled the operation or did not select a text file
17    %->input_name is empty or folder_path doesn't end on ".txt"
18 if isequal(input_name,0)||all(input_name(end-3:end)~='.txt')
19    %stop the programme and inform the user in a message box and
20    msgbox('No input text file selected. Programme stopped!',...
21        'Error','error');
22    error('No input text file selected. Programme stopped!');
23 end
24
25 %open textfile
26    %the complete path file name is the folder path + the file name
27 Input=fopen([folder_path,input_name],'r');
28
29 %function reads in the text informations from "Input"
30 [N,E,D,F,input_text_lines]=read_in_informations(Input,input_name,...
31    folder_path);
32
33 fclose(Input);      %close textfile
34 %%
35 function [N,E,D,F,input_text_lines]=read_in_informations(Input,...
36    input_name,folder_path)
37    %read in the input text into a cell array, every line in a new column
38        %primary used to search for faults
39  input_text_lines=regexp(fileread([folder_path,input_name]),'\n','split');
40    %delete the newline character
41    input_text_lines=regexprep(input_text_lines,'[\n\r]+','');
42    %rotate the cell array 270° -> every row in the array is a line in the
43        %text file
44    input_text_lines=rot90(input_text_lines,3);
45    %clear comment lines staring with "!"
46    input_text_lines=regexprep(input_text_lines, '^[!].*$','');
47    %find the line with solve (end of input commands)
48    solve_line=find(contains(input_text_lines,'SOLVE'));
49    %if a SOLVE was inserted solve_line is not empty
50    if ~isempty(solve_line)
51        %delete further lines after first "SOLVE"->solve_line(1,1)
52        input_text_lines(solve_line(1,1):end,1)=cellstr('');
53    end
```

```
54
55   %predefining the cell arrays to read in the truss informations(textscan)
56    N=cell(1,3,1);E=cell(1,3,1);D=cell(1,2,1);F=cell(1,3,1);
57    %set textmarker in Input file to beginning
58    frewind(Input);
59    while feof(Input)==0
60        %searching for all Nodes, starting with N and reading in
61            %the next 3 characters, separation symbol (delimiter) is a ","
62            %ignoring commentlines (commentstyle) starting with '!'
63            %copy informations into N
64        N_scan=textscan(Input, 'N %s %s %s','Delimiter', ',',...
65            'CommentStyle','!');
66        N{1}(height(N{1})+1:height(N{1})+height(N_scan{1}),1)=N_scan{1};
67        N{2}(height(N{2})+1:height(N{2})+height(N_scan{2}),1)=N_scan{2};
68        N{3}(height(N{3})+1:height(N{3})+height(N_scan{3}),1)=N_scan{3};
69        %textscan stops if a line doesnt match search structure
70            %check if the next line contains SOLVE
71        if contains(char(fgetl(Input)),'SOLVE')
72            %stop textscan and exit the while loop
73            break
74        end
75    end
76
77    %set textmarker in Input file to beginning
78    frewind(Input);
79    while feof(Input)==0
80        %searching for all Rods, starting with E and reading
81            %the next 3 characters, separation symbol (delimiter) is a ","
82            %ignoring commentlines (commentstyle) starting with '!'
83            %copy informations into E
84        E_scan=textscan(Input, 'E %s %s %s','Delimiter', ',',...
85            'CommentStyle','!');
86        E{1}(height(E{1})+1:height(E{1})+height(E_scan{1}),1)=E_scan{1};
87        E{2}(height(E{2})+1:height(E{2})+height(E_scan{2}),1)=E_scan{2};
88        E{3}(height(E{3})+1:height(E{3})+height(E_scan{3}),1)=E_scan{3};
89        %textscan stops if a line doesnt match search structure
90            %check if the next line contains SOLVE
91        if contains(char(fgetl(Input)),'SOLVE')
92            %stop textscan and exit the while loop
93            break
94        end
95    end
96
97    %set textmarker in Input file to beginning
98    frewind(Input);
99    while feof(Input)==0
100       %searching for all Supports starting with D and reading in
101           %the next 2 characters, separation symbol (delimiter) is a ","
102           %ignoring commentlines (commentstyle) starting with '!'
103           %copy informations into D
104      D_scan=textscan(Input,'D%s %s','Delimiter', ',','CommentStyle','!');
105       D{1}(height(D{1})+1:height(D{1})+height(D_scan{1}),1)=D_scan{1};
106       D{2}(height(D{2})+1:height(D{2})+height(D_scan{2}),1)=D_scan{2};
```

```
107         %textscan stops if a line doesnt match search structure
108             %check if the next line contains SOLVE
109         if contains(char(fgetl(Input)),'SOLVE')
110             %stop textscan and exit the while loop
111             break
112         end
113     end
114 
115     %set textmarker in Input file to beginning
116     frewind(Input);
117     while feof(Input)==0
118         %searching for all Forces starting with F and reading in
119             %the next 3 characters, separation symbol (delimiter) is a ","
120             %ignoring commentlines (commentstyle) starting with '!'
121             %copy informations into F
122         F_scan=textscan(Input, 'F%s %s %s',...
123             'Delimiter', ',','CommentStyle','!');
124         F{1}(height(F{1})+1:height(F{1})+height(F_scan{1}),1)=F_scan{1};
125         F{2}(height(F{2})+1:height(F{2})+height(F_scan{2}),1)=F_scan{2};
126         F{3}(height(F{3})+1:height(F{3})+height(F_scan{3}),1)=F_scan{3};
127         %textscan stops if a line doesnt match search structure
128             %check if the next line contains SOLVE
129         if contains(char(fgetl(Input)),'SOLVE')
130             %stop textscan and exit the while loop
131             break
132         end
133     end
134 
135     %if file ends with a incomplete entry of a element (e.g. N,1,0,) the
136         %last cell has one row less than others -> add a empty string,
137         %otherwise an error will occur later
138         %for everly element test if last cell has not the same amount of
139         %rows as the first row and if so add a empty string
140     if height(N{1})>height(N{3})
141         N{3}(height(N{1}),:)=cellstr('');
142     end
143     if height(E{1})>height(E{3})
144         E{3}(height(E{1}),:)=cellstr('');
145     end
146     if height(D{1})>height(D{2})
147         D{2}(height(D{1}),:)=cellstr('');
148     end
149     if height(F{1})>height(F{3})
150         F{3}(height(F{1}),:)=cellstr('');
151     end
152     if height(N{1})>height(N{2})
153         N{2}(height(N{1}),:)=cellstr('');
154     end
155     if height(E{1})>height(E{2})
156         E{2}(height(E{1}),:)=cellstr('');
157     end
158     if height(F{1})>height(F{2})
159         F{2}(height(F{1}),:)=cellstr('');
```

```
160     end
161 end
```

C:\Users\maxjo\OneDrive\...\Creation of Output File.m

```
 1 %% %2D-Truss Analysis and Calclulation Software
 2 %Bachelor Thesis, Max Stegmann
 3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
 4
 5 %% %sub programme: Creation_of_Output_File
 6     %create the ouput text file in the same folder as the input file and
 7     %print the title of the text file and repeat the input text file
 8     %informations
 9
10 %create the text output file name with the input_name + "-SOLUTION"
11 output_name=[input_name(1:end-4),'-OUTPUT.txt'];
12 %open the new text-document with write access
13 Output=fopen([folder_path,output_name],'w');
14
15 %function prints the title and repeats the input informations
16     %in the output text file, nothing is returned
17 write_titel_input_into_output(Output, input_text_lines,...
18     input_name,output_name,folder_path);
19 %close the output file
20 fclose(Output);
21 %%
22 %function to prints the title and and repeats the input informations
23     %in the output text file, nothing is returned
24 function write_titel_input_into_output(Output, input_text_lines,...
25     input_name, output_name,folder_path)
26     %print titel to the textfile
27     fprintf(Output,'%s\n%s\n%s\n%s\n%s\n%s\n%s\n',['*******************'...
28         '************************************************'],...
29         '                    2D-Truss Analysis Software',...
30         '           Calculation of Forces with the Node Methode',...
31         '                   Version: 1.0 Date: 25.06.2021',...
32         '                   Bachelor Thesis, Max Stegmann',...
33         ['HAW Hamburg, Department of Automotive and'...
34         'Aeronautical Engineering'],['*********************************'...
35         '***********************************']);
36     fprintf(Output,'%s\n%s',['Truss (Input file): ',input_name],...
37         'path: ');
38     %if the path name is longer than 76 character, split the path name into
39     %2 lines (round needs to be used a odd number cannot be divided by two
40     if length(folder_path)>76
41         fprintf(Output,'%s\n\t%s\n\n',...
42             folder_path(1,1:round(length(folder_path)/2)),...
43             folder_path(1,round(length(folder_path)/2):end));
44     else
45         fprintf(Output,'%s\n\n',folder_path);
46     end
47
48     %print the header of Input
49     fprintf(Output,'%s\n','                         Input Commands:');
50     fprintf(Output,'%s',['******************************************'...
51         '**************************']);
52
53    %delete empty lines in input_text_lines
```

C:\Users\maxjo\OneDrive\...\Creation of Output File.m

```
54     input_text_lines=input_text_lines(~cellfun('isempty',input_text_lines));
55     %print the input text file with removed empty lines
56     writecell(input_text_lines,[folder_path,output_name],'WriteMode',...
57         'append','Delimiter','tab')
58      end
```

```
1 %% %2D-Truss Analysis and Calclulation Software
2 %Bachelor Thesis, Max Stegmann
3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
4 
5 %% %sub programme: Fault_Analysis
6     %the sub programme checks the truss informations provided by the
7     %user in the cell arrays N,E,D,F for faults and writes
8     %the faults with the corresponding text line of the input text file
9     %into the output file, minor faults are corrected
10    %further processing is stopped if a major fault occurs
11 
12 %open the output file in mode append -> write behind the existing text
13 Output=fopen([folder_path,output_name],'a');
14 %print the header
15 fprintf(Output,'\n%s\n','                          Fault Analysis');
16 fprintf(Output,'%s\n',['*******************************************'...
17         '**************************']);
18  %function corrects capitalisation (lower case letter)fault
19      %e.g. n->N Dx->DX
20      %deletes spaces in the arguments and in the input_text_lines
21      %returned: corrected N,E,D,F,input_text_lines
22 [input_text_lines,N,E,D,F]=correct_capitalisation_spaces_fault...
23     (input_text_lines,N,E,D,F);
24 
25 %function checks input text in input_text_lines for structural faults:
26     %-check for missing or too many comma / wrong input command structure
27     %-swapped comma "," and dot "."
28     %-more than one input commands in one line
29     %-missing comment exclamation mark "!" at beginning
30     %-no N,E,D,F at line beginning for elements
31     %-wrong spelling of "SOLVE" for end of text
32     %nothing is returned, programm is stopped when a such a major fault was
33     %found
34 structural_fault(input_text_lines,Output,folder_path);
35 
36 %function checks input information for identical commands in two lines and
37     %deletes one (no major fault)
38     %returned: faults_count and corrected N,E,D,F and input_text_lines
39 [input_text_lines,N,E,D,F,faults_count]=identical_elements_fault...
40     (input_text_lines,N,E,D,F,Output);
41 
42 %function checks the command arguments in N,E,D,F for missing informaions
43     %or wrong data format
44     %nothing is returned, programm is stopped when a such a major fault was
45     %found
46 argument_fault(input_text_lines,N,E,D,F,Output,folder_path);
47 
48 %function checks the command arguments for node numbering faults:
49     %Nodes with the same number
50    %check the node numbers in the other elements (E,D,F) exists in one node
51     %nothing is returned, programm is stopped when a such a major fault was
52     %found
53 numbering_fault(input_text_lines,N,D,E,F,Output,folder_path);
```

```
54
55 %function corrects minor faults in the command arguments or informs the
56     %user if no correction necessary:
57     %2 rods with the same rod number
58     %check for two same elements on top of each other and delete one
59     %correct the angle in support to an angle between -180° and 180°
60 [input_text_lines,N,D,E,F,faults_count]=correction(input_text_lines, ...
61     N,D,E,F,faults_count,Output);
62
63 %function checks the truss informations for assembly faults:
64     %different start and end node number in rods
65     %2 nodes on top of each other (same coordinates)
66     %more than 2 supports per node
67     %every node needs at least 2 rods or one rod and one support
68     %a node with only 2 rods attached and a unit vector in the same
69         %direction is moveable
70     %nothing is returned, programm is stopped when a such a major fault was
71     %found
72     %before stopping the programm the figure of the truss is created by
73     %running the sub programms Transfer_to_double_matrices and Draw_Truss
74     %-> also F,folder_path and input_name is needed in the function
75 assembly_fault(input_text_lines,N,D,E,F,Output,folder_path,input_name);
76
77 %function checks if the truss has the necessary condition for statically
78     %defined
79     %calculation of the degree of static (in-)determinacy
80     %nothing is returned, programm is stopped when a such a major fault was
81     %found
82     %before stopping the programm the figure of the truss is created by
83     %running the sub programms Transfer_to_double_matrices and Draw_Truss
84     %-> also F,folder_path and input_name is needed in the function
85 statically_defined(N,D,E,F,Output,folder_path,input_name);
86
87 %inform the user when no faults found
88 if faults_count==0
89    fprintf(Output,'Fault Analysis completed. No faults were found!\n\n');
90 end
91
92 %close the text file (necessary if no faults were found)
93 fclose(Output);
94
95 %% %capitalisation fault
96      %corrects capitalisation (lower case letter)fault
97      %e.g. n->N Dx->DX
98      %deletes spaces in the arguments and in the input_text_lines
99      %returned: corrected N,E,D,F,input_text_lines
100 function [input_text_lines,N,E,D,F]=correct_capitalisation_spaces_fault...
101     (input_text_lines,N,E,D,F)
102     %delete spaces in input_text_lines
103     input_text_lines=regexprep(input_text_lines,' ','');
104     %clear comment lines staring with "!" again, because when there was a
105         %space infront of exclamation mark comment was not identified " !"
106     input_text_lines=regexprep(input_text_lines, '^[!].*$','',...
```

```
107         'lineanchors','dotexceptnewline');
108     %delete spaces in the cell arrays N,E,D,F
109     N{1}=regexprep(N{1},' ','');N{2}=regexprep(N{2},' ','');
110     N{3}=regexprep(N{3},' ','');
111     E{1}=regexprep(E{1},' ','');E{2}=regexprep(E{2},' ','');
112     E{3}=regexprep(E{3},' ','');
113     D{1}=regexprep(D{1},' ','');D{2}=regexprep(D{2},' ','');
114     F{1}=regexprep(F{1},' ','');F{2}=regexprep(F{2},' ','');
115     F{3}=regexprep(F{3},' ','');
116 
117     %correct capitalisation faults Dx->DX f->F
118     D{1}=regexprep(D{1},'x','X');D{1}=regexprep(D{1},'y','Y');
119     F{1}=regexprep(F{1},'x','X');F{1}=regexprep(F{1},'y','Y');
120     N{1}=regexprep(N{1},'n','N');E{1}=regexprep(E{1},'e','E');
121     D{1}=regexprep(D{1},'d','D');F{1}=regexprep(F{1},'f','F');
122     %adjustment also for input_text_lines -> later fault line can be found
123     input_text_lines=regexprep(input_text_lines,'x','X');
124     input_text_lines=regexprep(input_text_lines,'y','Y');
125     input_text_lines=regexprep(input_text_lines,'n','N');
126     input_text_lines=regexprep(input_text_lines,'e','E');
127     input_text_lines=regexprep(input_text_lines,'d','D');
128     input_text_lines=regexprep(input_text_lines,'f','F');
129 
130 end
131 %% %structural fault
132     %check for missing or too many comma / wrong input command structure
133     %check for swapped comma "," and dot "."
134     %check for more than one input command in one line(e.g. DX,1 DY,2)
135     %check for missing comment exclamation mark "!" at beginning of line
136     %check for no N,E,D,F at line beginning for elements
137     %check for wrong spelling of "SOLVE" for end of commands
138     %nothing is returned, programm is stopped when a such a major fault was
139     %found
140 function structural_fault(input_text_lines,Output,folder_path)
141     %major_faults_count counts the number of major faults, further
142         %processing is not possible and programm stops
143     major_faults_count=0;
144     %create a char vector of input_text_lines-> char can be used easier
145     input_text_lines_char=char(input_text_lines);
146 
147     for n=1:height(input_text_lines)        %for every input text line
148         %check if the line doesn't begins with "N or "E" for Node or Rod
149             %and there are not 3 commas in the line
150         if (input_text_lines_char(n,1)=='N' ||...
151                 input_text_lines_char(n,1)=='E') &&...
152                 count(input_text_lines(n,1),',')~=3
153             %increase major faults count
154             major_faults_count=major_faults_count+1;
155             %print the fault into output file with the corresponding line
156                 %from the input text file (=current foor-loop index n)
157             fprintf(Output,['Line %d in input text file: Major '...
158                 'structural fault detected!\nWrong input argument '...
159                 'structure: %s\n\n'],n,char(input_text_lines(n,:)));
```

```
160
161         %check if the line doesn't begins with "F" for Forces and there
162             %are not 2 commas in the line
163         elseif input_text_lines_char(n,1)=='F' &&...
164                 count(input_text_lines(n,1),',')~=2
165             %increase major faults count
166             major_faults_count=major_faults_count+1;
167             %print the fault into output file with the corresponding line
168                 %from the input text file (=current foor-loop index n)
169             fprintf(Output,['Line %d in input text file: Major '...
170                 'structural fault detected!\nWrong input argument '...
171                 'structure: %s\n\n'],n,char(input_text_lines(n,:)));
172
173         %check if the line doesn't begins with "D" for Supports and
174             %there are not 1 comma  in the line
175         elseif input_text_lines_char(n,1)=='D' &&...
176                 count(input_text_lines(n,1),',')~=1
177             %increase major faults count
178             major_faults_count=major_faults_count+1;
179             %print the fault into output file with the corresponding line
180                 %from the input text file (=current foor-loop index n)
181             fprintf(Output,['Line %d in input text file: Major '...
182                 'structural fault detected!\nWrong input argument '...
183                 'structure: %s\n\n'],n,char(input_text_lines(n,:)));
184
185         %check if the line is not empty and doesn't begins with N,E,D,F
186         elseif isempty(char(input_text_lines(n,1)))==0 &&...
187                 (input_text_lines_char(n,1)~='N' &&...
188                 input_text_lines_char(n,1)~='E' &&...
189                 input_text_lines_char(n,1)~='D' &&...
190                 input_text_lines_char(n,1)~='F')
191             %increase major faults count
192             major_faults_count=major_faults_count+1;
193             %print the fault into output file with the corresponding line
194                 %from the input text file (=current foor-loop index n)
195             fprintf(Output,['Line %d in input text file: Major '...
196                 'structural fault detected!\nWrong input argument '...
197                 'structure: %s\n\n'],n,char(input_text_lines(n,:)));
198
199         end
200     end
201
202     %if major structural faults were found, further processing not possible
203         %-> programm is stopped
204     if major_faults_count >0
205         %write number of major faults and reason into Output file
206         fprintf(Output,['%d Major structural Fault(s) were detected.\n'...
207             'Further Truss processing stopped!\nCorrect the faults in '...
208           'the input text file and restart the programme.\n'],...
209           major_faults_count);
210         fprintf(Output,['Reasons for a structural fault:\n\t'...
211             '-too many or too few arguments\n\t'...
212             '-swapped comma "," and dot "."\n\t'...
```

```
213             '-more than two input commands in one line \n\t'...
214             '-missing comment exclamation mark "!" at beginning of line'...
215             '\n\t-no N,E,D,F at line beginning for elements\n\t'...
216             '-one command was splitted into more than one line\n\t'...
217             '-wrong spelling of "SOLVE" for end of commands']);
218         %close the Output file
219         fclose(Output);
220         %stop the programme with an error and inform the user in a message
221             %box, print the link to the ouput file in the error (link cant
222             %be printed to the message box)
223         msgbox(['Major Fault(s) detected, further processing stopped!'...
224             ' Regard to the output file for more informations.'...
225             ' Linnk in the Command Window.'],'Error','error');
226         error(['<strong>Major Fault(s) detected, further processing '...
227             'stopped! Link to Output file:</strong> <a href="',...
228             folder_path,'">path</a>']);
229     end
230 end
231 %% %identical elements fault
232     %checks input informations for identical commands in two lines and
233     %deletes one (no major fault)
234     %returned: fault_counts, corrected N,E,D,F,input_text_line
235 function [input_text_lines,N,E,D,F,faults_count]=identical_elements_fault...
236 (input_text_lines,N,E,D,F,Output)
237     %faults_count counts the number of minor faults than were corrected
238     faults_count=0;
239     %create a char vector of input_text_lines-> char can be used easier
240     input_text_lines_char=char(input_text_lines);
241 
242     %compare two different lines in the input_text_line for identical
243         %comands
244     for n=1:height(input_text_lines)         %for every line/row
245         for m=1:height(input_text_lines)     %for every line/row
246             %check if the line of the first and second loop are not the
247                 %same and the characters are the same in the 2 lines
248                 %and the lines are not empty
249             if n~=m && all(input_text_lines_char(n,:) == ...
250                 input_text_lines_char(m,:)) &&...
251                 ~isempty(char(input_text_lines(n,:)))
252                 %increase faults count61
253                 faults_count=faults_count+1;
254                %print the fault into output file with the corresponding
255                     %line from the input text file(=current foor-loop
256                     %index n) inform the user, that one command is deleted
257                 fprintf(Output,['Line %d and %d in input text file: '...
258                     'Identical element fault detected!\n'],n,m);
259                 fprintf(Output,['Two identical commands were inserted:'...
260                    ' %s\nOne element is deleted.\n\n'],...
261                    char(input_text_lines(n,1)));
262 
263                %search for the identical command in N,E,D,F
264                 for o=1:height(N{1})             %for every command in N
265                 %check if the read in node informations in N is the same
```

```
266                         %as the line with the identical command
267                     if contains(input_text_lines(n,1),['N,',char(N{1}(o,1)),...
268                         ',',char(N{2}(o,1)),',',char(N{3}(o,1))])
269                         %delete the arguments of the command in N
270                         N{1}(o,:)=[];N{2}(o,:)=[];N{3}(o,:)=[];
271                         %exit the for loop for every command -> only one of the
272                             %identical commands is deleted
273                         break
274                     end
275                     end
276 
277                     for o=1:height(E{1})            %for every command in E
278                     %check if the read in rod informations in E is the same
279                         %as the line with the identical command
280                     if contains(input_text_lines(n,1),['E,',char(E{1}(o,1)),...
281                         ',',char(E{2}(o,1)),',',char(E{3}(o,1))])
282                         %delete the arguments of the command in E
283                         E{1}(o,:)=[];E{2}(o,:)=[];E{3}(o,:)=[];
284                         %exit the for loop for every command -> only one of the
285                             %identical commands is deleted
286                         break
287                     end
288                     end
289 
290                     for o=1:height(D{1})            %for every command in D
291                     %check if the read in support informations in D is the same
292                         %as the line with the identical command
293                         %for the argument fault, locked direction missing:
294                         %e.g. D,2 node location is read in in cell D{1}
295                         %->also check for this identical fault line
296                     if contains(input_text_lines(n,1),['D',...
297                         char(D{1}(o,1)),',',char(D{2}(o,1))]) ||...
298                         contains(input_text_lines(n,1),['D,',char(D{1}(o,1))])
299                         %delete the arguments of the command in D
300                         D{1}(o,:)=[];D{2}(o,:)=[];
301                         %exit the for loop for every command -> only one of the
302                         %identical commands is deleted
303                         break
304                     end
305                     end
306 
307                     for o=1:height(F{1})            %for every command in F
308                     %check if the read in force informations in F is the same
309                         %as the line with the identical command
310                         %for the argument fault, locked direction missing:
311                         %e.g. F,2,10 node location is read in in cell F{1}
312                         %->also check for this identical fault line
313                     if contains(input_text_lines(n,1),['F',...
314                         char(F{1}(o,1)),',',char(F{2}(o,1)),',',char(F{3}(o,1))]) ||
315                         contains(input_text_lines(n,1),['F,',...
316                         char(F{1}(o,1)),',',char(F{2}(o,1))])
317                         %delete the arguments of the command in F
318                         F{1}(o,:)=[];F{2}(o,:)=[];F{3}(o,:)=[];
```

```
319                     %exit the for loop for every command -> only one of the
320                     %identical commands is deleted
321                     break
322                 end
323                 end
324                 %delete the second identical line by creating an empty cell
325                 input_text_lines(m,:)=cellstr('');
326             end
327         end
328     end
329 end
330 %% %argument fault
331     %check for missing arguments or wrong data format in the commands in
332     %N,E,D,F
333     %nothing is returned, programm is stopped when a such a major fault was
334     %found
335 function argument_fault(input_text_lines,N,E,D,F,Output,folder_path)
336     %major_faults_count counts the number of major faults, further
337         %processing is not possible and programm stops
338     major_faults_count=0;
339     %create a char vector of input_text_lines-> char can be used easier
340     input_text_lines_char=char(input_text_lines);
341 
342     %check for missing infomrations in the Nodes
343     for n=1:height(N{1})            %for every Node command in N
344         %if entry in cell one is not a numeric entry
345         if isnan(str2double(N{1}(n,1)))
346             %increase major faults count
347             major_faults_count=major_faults_count+1;
348             %search for the fault line of the node command in the
349                 %input text lines
350             fault_line=find(contains(input_text_lines,['N,',...
351                 char(N{1}(n,1)),',',char(N{2}(n,1)),',',char(N{3}(n,1))]));
352             %print the fault into output file with the corresponding line
353                 %from the input text file
354             fprintf(Output,['Line %d in input text file: Major argument'...
355                 ' fault detected!\n'],fault_line);
356             fprintf(Output,['The Node number is missing or no numeric '...
357                 'data was inserted: %s\n\n'],...
358                 char(input_text_lines(fault_line,1)));
359         end
360         %if entry in cell two is not a numeric entry
361         if isnan(str2double(N{2}(n,1)))
362             %increase major faults count
363             major_faults_count=major_faults_count+1;
364             %search for the fault line of the node command in the
365                 %input text lines
366             fault_line=find(contains(input_text_lines,['N,',...
367                 char(N{1}(n,1)),',',char(N{2}(n,1)),',',char(N{3}(n,1))]));
368             %print the fault into output file with the corresponding line
369                 %from the input text file
370             fprintf(Output,['Line %d in input text file: Major argument'...
371                 ' fault detected!\n'],fault_line);
```

```
372             fprintf(Output,['The x-coordinate is missing or no numeric '...
373                 'data was inserted: %s\n\n'],...
374                 char(input_text_lines(fault_line,1)));
375         end
376         %if entry in cell three is not a numeric entry
377         if isnan(str2double(N{3}(n,1)))
378             %increase major faults count
379             major_faults_count=major_faults_count+1;
380             %search for the fault line of the node command in the
381                 %input text lines
382             fault_line=find(contains(input_text_lines,['N,',...
383                 char(N{1}(n,1)),',',char(N{2}(n,1)),',',char(N{3}(n,1))]));
384             %if last argument is missing e.g. N,2,0, and there is another
385                 %Node with identical arguments N,2,0,1 two fault lines are
386                 %returned-> check which line is the right fault line
387             if height(fault_line)==1
388                 fprintf(Output,['Line %d in input text file: Major'...
389                         ' argument fault detected!\n'],fault_line);
390                 fprintf(Output,['The y-coordinate is missing or no'...
391                         ' numeric data was inserted: %s\n\n'],...
392                         char(input_text_lines(fault_line,1)));
393             elseif height(fault_line)>1
394              for m=1:height(fault_line)      %for every fault line
395                 %create a character line of the fault line
396                 fault_line_char=char(input_text_lines(fault_line(m,1)));
397                 %is the last character is a comma the last argument is
398                     %missing this is the right fault line
399                 if fault_line_char(1,end)==','
400                     %print the fault into output file with the
401                         %corresponding line from the input text file
402                     fprintf(Output,['Line %d in input text file: Major'...
403                         ' argument fault detected!\n'],fault_line(m,1));
404                     fprintf(Output,['The y-coordinate is missing or no'...
405                         ' numeric data was inserted: %s\n\n'],...
406                         char(input_text_lines(fault_line(m,1),1)));
407                 end
408              end
409             end
410         end
411     end
412 
413     %check for missing infomrations in the Rods
414     for n=1:height(E{1})                    %for every Rod command in R
415         %if entry in cell one is not a numeric entry
416         if isnan(str2double(E{1}(n,1)))
417             %increase major faults count
418             major_faults_count=major_faults_count+1;
419             %search for the fault line of the rod command in the
420                 %input text lines
421             fault_line=find(contains(input_text_lines,['E,',...
422                 char(E{1}(n,1)),',',char(E{2}(n,1)),',',char(E{3}(n,1))]));
423             %print the fault into output file with the corresponding line
424                 %from the input text file
```

```
425             fprintf(Output,['Line %d in input text file: Major argument'...
426                 ' fault detected!\n'],fault_line);
427             fprintf(Output,['The Rod number is missing or no numeric '...
428                 'data was inserted: %s\n\n'],...
429                 char(input_text_lines(fault_line,1)));
430         end
431         %if entry in cell two is not a numeric entry
432         if isnan(str2double(E{2}(n,1)))
433             %increase major faults count
434             major_faults_count=major_faults_count+1;
435             %search for the fault line of the rod command in the
436                 %input text lines
437             fault_line=find(contains(input_text_lines,['E,',...
438                 char(E{1}(n,1)),',',char(E{2}(n,1)),',',char(E{3}(n,1))]));
439             %print the fault into output file with the corresponding line
440                 %from the input text file
441             fprintf(Output,['Line %d in input text file: Major argument'...
442                 ' fault detected!\n'],fault_line);
443             fprintf(Output,['The beginning node number is missing or no'...
444                 ' numeric data was inserted: %s\n\n'],...
445                 char(input_text_lines(fault_line,1)));
446         end
447         %if entry in cell three is not a numeric entry
448         if isnan(str2double(E{3}(n,1)))
449             %increase major faults count
450             major_faults_count=major_faults_count+1;
451             %search for the fault line of the rod command in the
452                 %input text lines
453             fault_line=find(contains(input_text_lines,['E,',...
454                 char(E{1}(n,1)),',',char(E{2}(n,1)),',',char(E{3}(n,1))]));
455             %if last argument is missing e.g. E,2,1, and there is another
456                 %Rod with identical arguments N,2,1,2 two fault lines are
457                 %returned-> check which line is the right fault line
458             if height(fault_line)==1
459                 fprintf(Output,['Line %d in input text file: Major'...
460                     ' argument fault detected!\n'],fault_line);
461                 fprintf(Output,['The end node is missing or no'...
462                     ' numeric data was inserted: %s\n\n'],...
463                     char(input_text_lines(fault_line,1)));
464             elseif height(fault_line)>1
465              for m=1:height(fault_line)          %for every fault line
466                 %create a character line of the fault line
467                 fault_line_char=char(input_text_lines(fault_line(m,1)));
468                 %is the last character is a comma the last argument is
469                     %missing this is the right fault line
470                 if fault_line_char(1,end)==','
471                     %print the fault into output file with the
472                         %corresponding line from the input text file
473                     fprintf(Output,['Line %d in input text file: Major'...
474                         ' argument fault detected!\n'],fault_line(m,1));
475                     fprintf(Output,['The end node is missing or no'...
476                         ' numeric data was inserted: %s\n\n'],...
477                         char(input_text_lines(fault_line(m,1),1)));
```

```
478                 end
479               end
480             end
481         end
482     end
483 
484     %check for missing infomrations in the supports, for every support
485     for n=1:height(D{1})
486         %find the line in the input text lines of the command in D
487         fault_line=find(contains(input_text_lines,['D',char(D{1}(n,1)),...
488                 ',',char(D{2}(n,1))]));
489         %check if there was no line found (fault line empty) there is a
490             %missing locked direction and the node location was read in
491             %into cell D{1} e.g. D,2
492         if isempty(fault_line)
493             %search for the fault line for missing locked direction
494             fault_line=find(contains(input_text_lines,['D,',...
495                 char(D{1}(n,1))]));
496         end
497         %check if the first argument is not a X and not a Y and not a
498             %numeric entry for an angle -> wrong entry/format
499             %or check if the second character is an comma-> no entry
500         if (all(char(D{1}(n,1))~='X') && all(char(D{1}(n,1))~='Y')&&...
501                 isnan(str2double(D{1}(n,1)))) ||...
502                 any(input_text_lines_char(fault_line,2)==',')
503             %increase major faults count
504             major_faults_count=major_faults_count+1;
505             %print the fault into output file with the corresponding line
506                 %from the input text file
507             fprintf(Output,['Line %d in input text file: Major argument'...
508                 ' fault detected!\n'],fault_line);
509             fprintf(Output,['No "X", "Y" or numeric angle '...
510                 'was inserted: %s\n\n'],...
511                 char(input_text_lines(fault_line,1)));
512         end
513         %if entry in cell two is not a numeric entry and the second
514             %character of the fault line is not a comma or the first entry
515             %is not a numeric entry and the second character is a comma
516             %-> node location missing or no numeric entry
517         if (isnan(str2double(D{2}(n,1))) && ...
518                 any(input_text_lines_char(fault_line,2)~=',')) || ...
519                 (isnan(str2double(D{1}(n,1))) && ...
520                 any(input_text_lines_char(fault_line,2)==','))
521             %increase major faults count
522             major_faults_count=major_faults_count+1;
523             %if last argument is missing e.g. DX, and there is another
524                 %Rod with identical arguments DX,2 two fault lines are
525                 %returned-> check which line is the right fault line
526             if height(fault_line)==1
527                     fprintf(Output,['Line %d in input text file: Major'...
528                         ' argument fault detected!\n'],fault_line);
529                     fprintf(Output,['The node number is missing or no'...
530                         ' numeric data was inserted: %s\n\n'],...
```

```
531                         char(input_text_lines(fault_line,1)));
532             elseif height(fault_line)>1
533             for m=1:height(fault_line)          %for every fault line
534                 %create a character line of the fault line
535                 fault_line_char=char(input_text_lines(fault_line(m,1)));
536                 %is the last character is a comma the last argument is
537                     %missing this is the right fault line
538                 if fault_line_char(1,end)==','
539                     %print the fault into output file with the
540                         %corresponding line from the input text file
541                     fprintf(Output,['Line %d in input text file: Major'...
542                         ' argument fault detected!\n'],fault_line(m,1));
543                     fprintf(Output,['The node number is missing or no'...
544                         ' numeric data was inserted: %s\n\n'],...
545                         char(input_text_lines(fault_line(m,1),1)));
546                 end
547             end
548             end
549         end
550     end
551 
552 
553     %check for missing infomrations in the forces, for every force
554     for n=1:height(F{1})
555         %find the line in the input text lines of the command in D
556         fault_line=find(contains(input_text_lines,['F',char(F{1}(n,1)),...
557             ',',char(F{2}(n,1)),',',char(F{3}(n,1))]));
558         %check if there was no line found (fault line empty) there is a
559             %missing locked direction and the node location was read in
560             %into cell F{1} and the value into F{2} e.g. F,2,10
561         if isempty(fault_line)
562             %search for the fault line for missing locked direction
563             fault_line=find(contains(input_text_lines,['F,',...
564                 char(F{1}(n,1)),',',char(F{2}(n,1))]));
565         end
566         %check if the first argument is not a X and not a Y
567             %-> wrong entry/format
568             %or check if the second character is an comma-> no entry
569         if (any(string(F{1}(n,1))~='X') &&any(string(F{1}(n,1))~='Y'))||...
570                 any(input_text_lines_char(fault_line,2)==',')
571             %increase major faults count
572             major_faults_count=major_faults_count+1;
573             %print the fault into output file with the corresponding line
574                 %from the input text file
575             fprintf(Output,['Line %d in input text file: Major argument'...
576                 ' fault detected!\n'],fault_line);
577             fprintf(Output,'No "X" or "Y" was inserted: %s\n\n',...
578                 char(input_text_lines(fault_line,1)));
579         end
580         %if entry in cell two is not a numeric entry
581         if (isnan(str2double(F{2}(n,1))) && ...
582                 any(input_text_lines_char(fault_line,2)~=',')) ||...
583             (isnan(str2double(F{1}(n,1))) &&...
```

```
584             any(input_text_lines_char(fault_line,2)==','))
585             %increase major faults count
586             major_faults_count=major_faults_count+1;
587             %print the fault into output file with the corresponding line
588                 %from the input text file
589             fprintf(Output,['Line %d in input text file: Major argument'...
590                 ' fault detected!\n'],fault_line);
591             fprintf(Output,['The node number is missing or no'...
592                 ' numeric data was inserted: %s\n\n'],...
593                 char(input_text_lines(fault_line,1)));
594         end
595         %if entry in cell two is not a numeric entry and the second
596             %character of the fault line is not a comma -> locked direction
597             %missing and value was read in into F{2}
598         if (isnan(str2double(F{3}(n,1))) &&...
599                 any(input_text_lines_char(fault_line,2)~=',')) ||...
600                 (isnan(str2double(F{2}(n,1))) &&...
601                 any(input_text_lines_char(fault_line,2)==','))
602             %increase major faults count
603             major_faults_count=major_faults_count+1;
604             %if last argument is missing e.g. FX,2 and there is another
605                 %Force with identical arguments FX,2,10 two fault lines are
606                 %returned-> check which line is the right fault line
607             if height(fault_line)==1
608                     fprintf(Output,['Line %d in input text file: Major'...
609                         ' argument fault detected!\n'],fault_line);
610                     fprintf(Output,['The Force value is missing or no'...
611                         ' numeric data was inserted: %s\n\n'],...
612                         char(input_text_lines(fault_line,1)));
613             elseif height(fault_line)>1
614             for m=1:height(fault_line)           %for every fault line
615                 %create a character line of the fault line
616                 fault_line_char=char(input_text_lines(fault_line(m,1)));
617                 %is the last character is a comma the last argument is
618                     %missing this is the right fault line
619                 if fault_line_char(1,end)==','
620                     %print the fault into output file with the
621                         %corresponding line from the input text file
622                     fprintf(Output,['Line %d in input text file: Major'...
623                         ' argument fault detected!\n'],fault_line(m,1));
624                     fprintf(Output,['The Force value is missing or no'...
625                         ' numeric data was inserted: %s\n\n'],...
626                         char(input_text_lines(fault_line(m,1),1)));
627                 end
628             end
629             end
630         end
631     end
632
633
634     %if major argument faults found, further processing not possible
635     %-> stop
636     if major_faults_count >0
```

```
637         %write number of major faults and reason into Output file
638         fprintf(Output,['%d Major argument Fault(s) were detected.\n'...
639             'Further Truss processing stopped!\nCorrect the faults in '...
640           'the input text file and restart the programme.\n'],...
641           major_faults_count);
642         fprintf(Output,['Reasons for a argument fault:\n\t'...
643             '-no argument was inserted\n\t'...
644             '-wrong data format, e.g. no numeric entry\n\t'...
645             '-a unit was inserted\n\t']);
646         %close the output file
647         fclose(Output);
648         %stop the programme with an error and inform the user in a message
649             %box, print the link to the ouput file in the error (link cant
650             %be printed to the message box)
651         msgbox(['Major Fault(s) detected, further processing stopped!'...
652             ' Regard to the output file for more informations.'...
653             ' Linnk in the Command Window.'],'Error','error');
654         error(['<strong>Major Fault(s) detected, further processing '...
655             'stopped! Link to Output file:</strong> <a href="',...
656             folder_path,'">path</a>']);
657     end
658 end
659 %% %node numbering fault
660     %checks the command arguments for node nu,bering faults:
661     %Nodes with the same number
662    %check the node numbers in the other elements (E,D,F) exists in one node
663     %nothing is returned, programm is stopped when a such a major fault was
664     %found
665 function numbering_fault(input_text_lines,N,D,E,F,Output,folder_path)
666     %major_faults_count the number of major faults, further
667     %processing is not possible and programm stops
668     major_faults_count=0;
669 
670     %check for 2 nodes with the same node-number -> 2 for loops to compare
671         %2 different node numbers
672     for n=1:height(N{1})                       %for every node number
673         for m=1:height(N{1})                   %for every node number
674             %check if two different nodes are compared and make sure the
675                 %two nodes are only compared once -> n<m
676                 %check if the Node numbers are the same
677             if n<m && str2double(N{1}(n,1))==str2double(N{1}(m,1))
678                 %increase major faults count
679                 major_faults_count=major_faults_count+1;
680                 %search for the fault line of the first node command in the
681                     %input text lines
682                 fault_line=find(contains(input_text_lines,['N,',...
683                     char(N{1}(n,1)),',',char(N{2}(n,1)),',',...
684                     char(N{3}(n,1))]));
685                 %search for the fault line of the second node command in
686                     %theinput text lines
687                 fault_line_2=find(contains(input_text_lines,['N,',...
688                     char(N{1}(m,1)),',',char(N{2}(m,1)),',',...
689                     char(N{3}(m,1))]));
```

```
690                 %print the fault into output file with the two
691                     %corresponding lines from the input text file
692                 fprintf(Output,['Line %d and %d in input text file: '...
693                     'Major numbering fault detected!\n'],...
694                     fault_line,fault_line_2);
695                 fprintf(Output,['Two Nodes with the same Node number:'...
696                     ' %s  %s\n\n'],char(input_text_lines(fault_line,1)),...
697                     char(input_text_lines(fault_line_2,1)));
698             end
699         end
700     end
701 
702     %check for existing beginning and end node number in rods arguments
703         %E{1} and E{2}
704     for n=1:height(E{1})            %for every rod command in E
705         %check if the beginning node number in E{2} and end node number
706             %E{3} matches with any node number in N{1}
707             %str2num needs to be used because str2double can't transform a
708             %whole vector
709         if ~any(str2num(char(N{1}))==str2double(E{2}(n,1))) ||...
710                 ~any(str2num(char(N{1}))==str2double(E{3}(n,1)))
711             %increase major faults count
712             major_faults_count=major_faults_count+1;
713             %search for the fault line of the rod command in the input text
714                 %lines
715             fault_line=find(contains(input_text_lines,['E,',...
716                 char(E{1}(n,1)),',',char(E{2}(n,1)),',',char(E{3}(n,1))]));
717             %print the fault into output file with the corresponding
718                     %line from the input text file
719             fprintf(Output,['Line %d in input text file: Major '...
720                 'numbering fault detected!\n'],fault_line);
721             fprintf(Output,['The beginning or end node number doesn''t '...
722                 'exists: %s\n\n'],char(input_text_lines(fault_line,1)));
723         end
724     end
725 
726     %check for existing node location number in the force arguments F{2}
727     for n=1:height(F{1})            %for every force command in F
728         %check if the node location number in F{2} matches with any node
729             %number in N{1}
730             %str2num needs to be used because str2double can't transform a
731             %whole vector
732         if ~any(str2num(char(N{1}))==str2double(F{2}(n,1)))
733             %increase major faults count
734             major_faults_count=major_faults_count+1;
735             %search for the fault line of the force command in the input
736                 %text lines
737             fault_line=find(contains(input_text_lines,['F',...
738                 char(F{1}(n,1)),',',char(F{2}(n,1)),',',char(F{3}(n,1))]));
739             %print the fault into output file with the corresponding
740                     %line from the input text file
741             fprintf(Output,['Line %d in input text file: Major numbering'...
742                 ' fault detected!\n'],fault_line);
```

```
743             fprintf(Output,['The entered node number doesn''t '...
744                 'exists: %s\n\n'],char(input_text_lines(fault_line,1)));
745         end
746     end
747 
748     %check for existing node location number in the supports arguments D{2}
749     for n=1:height(D{1})            %for every support command in D
750         %check if the node location number in D{2} matches with any node
751             %number in N{1}
752             %str2num needs to be used because str2double can't transform a
753             %whole vector
754         if ~any(str2num(char(N{1}))==str2double(D{2}(n,1)))
755             %increase major faults count
756             major_faults_count=major_faults_count+1;
757             %search for the fault line of the support command in the input
758                 %text lines
759             fault_line=find(contains(input_text_lines,['D',...
760                 char(D{1}(n,1)),',',char(D{2}(n,1))]));
761             %print the fault into output file with the corresponding
762                     %line from the input text file
763             fprintf(Output,['Line %d in input text file: Major numbering'...
764                 ' fault detected!\n'],fault_line);
765             fprintf(Output,['The entered node number doesn''t '...
766                 'exists: %s\n\n'],char(input_text_lines(fault_line,1)));
767         end
768     end
769 
770         %if major argument faults found, further processing not possible
771         %-> stop
772     if major_faults_count >0
773         %write number of major faults and reason into output file
774         fprintf(Output,['%d Major numbering Fault(s) were detected.\n'...
775             'Further Truss processing stopped!\nCorrect the faults in the '...
776           'input text file and restart the programme.\n'], major_faults_count);
777         fprintf(Output,['Reasons for a numbering fault:\n\t'...
778             '-2 nodes with the same node number -> no differentiation\n\t'...
779             '-in the elements wasn''t entered an existing node number\n\t']);
780         %close the output file
781         fclose(Output);
782         %stop the programme with an error and inform the user in a message
783             %box, print the link to the ouput file in the error (link cant
784             %be printed to the message box)
785         msgbox(['Major Fault(s) detected, further processing stopped!'...
786             ' Regard to the output file for more informations.'...
787             ' Linnk in the Command Window.'],'Error','error');
788         error(['<strong>Major Fault(s) detected, further processing '...
789             'stopped! Link to Output file:</strong> <a href="',...
790             folder_path,'">path</a>']);
791     end
792 end
793 %% %corrections (no major faults)
794     %check for correctable faultsNach
795     %2 rods with the same rod number
```

```
796     %check for two same elements on top of each other and delete one
797     %correct the angle in support to an angle between -180° and 180°
798 function [input_text_lines,N,D,E,F,faults_count]=...
799     correction(input_text_lines,N,D,E,F,faults_count,Output)
800 
801     %check for 2 Rods with the same Rod-number -> 2 for loops are needed to
802     %compare 2 rod numbers
803     %no major fault because rod nu,ber is only used for display in figure
804     %and is printed into output file after calculation
805     for n=1:height(E{1})                    %for every rod command in R
806         for m=1:height(E{1})                %for every rod command in R
807             %check if two different rods are compared and make sure the
808                 %two rods are only compared once -> n<m
809                 %check if the rod numbers are the same
810             if n<m && str2double(E{1}(n,1))==str2double(E{1}(m,1))
811                 %increase the faults count
812                 faults_count=faults_count+1;
813                 %search for the fault line of the first rod command in the
814                     %input text lines
815                 fault_line=find(contains(input_text_lines,['E,',...
816                     char(E{1}(n,1)),',',char(E{2}(n,1)),',',...
817                     char(E{3}(n,1))]));
818                 %search for the fault line of the second rod command in the
819                     %input text lines
820                 fault_line_2=find(contains(input_text_lines,['E,',...
821                     char(E{1}(m,1)),',',char(E{2}(m,1)),',',...
822                     char(E{3}(m,1))]));
823                 %inform the user of the two rods with the same rod number
824                     %and encourage the user to replace the rod for better
825                     %indentification
826                 fprintf(Output,['Line %d and %d in input text file: Num'...
827                     'bering fault detected!\n'],fault_line,fault_line_2);
828                 fprintf(Output,['Two Rods with the same rode number:'...
829                     ' %s  %s\nConsider replacing one Rod number for '...
830                     'better identification.\n\n'],...
831                     char(input_text_lines(fault_line,1)),...
832                     char(input_text_lines(fault_line_2,1)));
833             end
834         end
835     end
836 
837     %check for 2 rods on top of each other -> 2 rods with same start and
838         %end node number -> 2 for loops are needed to compare 2 rod
839     for n=1:height(E{1})                    %for every rod command in R
840         for m=1:height(E{1})                %for every rod command in R
841             %check if two different rods are compared and make sure the
842                 %two rods are only compared once -> n<m
843                 %check if the start and end node numbers are the same
844             if (n<m && str2double(E{2}(n,1))==str2double(E{2}(m,1)) &&...
845                 str2double(E{3}(n,1))==str2double(E{3}(m,1))) || (n<m &&...
846                 str2double(E{2}(n,1))==str2double(E{3}(m,1)) &&...
847                 str2double(E{3}(n,1))==str2double(E{2}(m,1)))
848                 %increase the faults count
```

```
849                 faults_count=faults_count+1;
850                 %search for the fault line of the first rod command in the
851                     %input text lines
852                 fault_line=find(contains(input_text_lines,['E,',...
853                     char(E{1}(n,1)),',',char(E{2}(n,1)),',',...
854                     char(E{3}(n,1))]));
855                 %search for the fault line of the second rod command in the
856                     %input text lines
857                 fault_line_2=find(contains(input_text_lines,['E,',...
858                     char(E{1}(m,1)),',',char(E{2}(m,1)),',',...
859                     char(E{3}(m,1))]));
860                 %print the fault into output file with the two
861                     %corresponding lines from the input text file
862                     %inform the user that the second rod is deleted and
863                     %print the rod into the output file
864                 fprintf(Output,['Line %d and %d in input text file:'...
865                     ' Truss assembly fault detected!\n'],...
866                     fault_line,fault_line_2);
867                 fprintf(Output,['Two Rods with the same start and end '...
868                     'node numbers (on top of each other):\n\t %s  %s\n'...
869                         'Rod %s is deleted.\n\n'],...
870                     char(input_text_lines(fault_line,1)),...
871                     char(input_text_lines(fault_line_2,1)),...
872                     char(input_text_lines(fault_line_2,1)));
873                 %delete the arguments of the second rod command by creating
874                     %empty arguments, deleted after for loop because there
875                     %would be an error in for loop addressing m>heightE{1}
876                 E{1}(m,:)=cellstr('');E{2}(m,:)=cellstr('');
877                 E{3}(m,:)=cellstr('');
878             end
879         end
880     end
881     %delete the empty arguments in the cells
882     E{1}=E{1}(~cellfun('isempty',E{1}));
883     E{2}=E{2}(~cellfun('isempty',E{2}));
884     E{3}=E{3}(~cellfun('isempty',E{3}));
885 
886     %check for 2 forces on top of each other -> 2 forces with same
887         %direction of action and same node location number
888         %add the value together
889         %-> 2 for loops are needed to compare 2 rod
890     for n=1:height(F{1})                %for every force command in F
891         for m=1:height(F{1})            %for every force command in F
892             %check if two different forces are compared and make sure the
893                 %two forces are only compared once -> n<m
894                 %check if the direction of action and node number are the
895                 %same
896             if n<m && str2double(F{2}(m,1))==str2double(F{2}(n,1)) &&...
897                     char(F{1}(m,1))==char(F{1}(n,1))
898                 %increase the faults count
899                 faults_count=faults_count+1;
900                 %search for the fault line of the first force command in
901                     %the input text lines
```

```
902                 fault_line=find(contains(input_text_lines,['F',...
903                     char(F{1}(n,1)),',',char(F{2}(n,1)),',',...
904                     char(F{3}(n,1))]));
905                 %search for the fault line of the second force command in
906                     %the input text lines
907                 fault_line_2=find(contains(input_text_lines,['F',...
908                     char(F{1}(m,1)),',',char(F{2}(m,1)),',',...
909                     char(F{3}(m,1))]));
910                 %print the fault into output file with the two
911                     %corresponding lines from the input text file
912                 fprintf(Output,['Line %d and %d in input text file: '...
913                     'Truss assembly fault detected!\n'],fault_line,...
914                     fault_line_2);
915                 fprintf(Output,['Two forces on top of each other (same '...
916                     'node and direction):\n\t %s  %s\n'],...
917                     char(input_text_lines(fault_line,1)), ...
918                     char(input_text_lines(fault_line_2,1)));
919                 %add the values together into the first force
920                 F{3}(n,:)=cellstr(string(str2double(F{3}(n,1)) ...
921                    +str2double(F{3}(m,1))));
922                 %delete the arguments of the second force command by
923                     %creating empty arguments, deleted after for loop
924                     %because there  would be an error in for loop
925                     %addressing m>heightF{1}
926                 F{1}(m,:)=cellstr('');F{2}(m,:)=cellstr('');
927                 F{3}(m,:)=cellstr('');
928                 %(no correction in input_text_lines necessary, because not
929                     %further needed in Fault_Analysis)
930                 %print the new force with the added together values into
931                     %the output file
932                 fprintf(Output,['Force values are added together:'...
933                     ' F%s,%s,%s\n\n'],...
934                     char(F{1}(n,1)),char(F{2}(n,1)),char(F{3}(n,1)));
935             end
936         end
937     end
938     %delete the empty arguments in the cells
939     F{1}=F{1}(~cellfun('isempty',F{1}));
940     F{2}=F{2}(~cellfun('isempty',F{2}));
941     F{3}=F{3}(~cellfun('isempty',F{3}));
942
943     %correct the bearing angle in the supports, if it is greater than 180°
944         %or smaller than -180°
945     for n=1:height(D{1})                  %for every support command in D
946         %check if the locked direction input is numeral(->angle input) and
947             %the angle is greater than 180° or smaller than -180°
948         if ~isnan(str2double(D{1}(n,1))) && (str2double(D{1}(n,1))>180||...
949                 str2double(D{1}(n,1))<-180)
950             %increase the faults count
951             faults_count=faults_count+1;
952             %search for the fault line of the support command in
953                     %the input text lines
954             fault_line=find(contains(input_text_lines,['D',...
```

```
955                     char(D{1}(n,1)),',',char(D{2}(n,1))]));
956                 %print the fault into output file with the
957                         %corresponding line from the input text file
958                 fprintf(Output,['Line %d in input text file: Truss assembly'...
959                     ' fault detected!\n'],fault_line);
960                 fprintf(Output,['The angle to the vertical axis is greater '...
961                     'than 180° \nor smaller than -180°:  %s\n'],...
962                     char(input_text_lines(fault_line,1)));
963                 %correct the angle by dividing the angle by 360 and rounding
964                     %it to a full number the angle it is checked how many times
965                     %360° needs to be subtract or added to get an angle between
966                     %-180° and 180°, print it as a cell back into the argument
967                 D{1}(n,1)=cellstr(string(str2double(D{1}(n,1))-...
968                     round(str2double(D{1}(n,1))/360)*360));
969                 %the fault is also corrected in the input text lines to be able
970                     %to find further faults
971                 input_text_lines(fault_line,1)=...
972                     cellstr(string(['D',char(D{1}(n,1)),',',char(D{2}(n,1))]));
973                 %print the new angle to the output file
974                 fprintf(Output,'The angle is corrected to:  %s\n\n',...
975                     char(input_text_lines(fault_line,1)));
976             end
977         end
978 
979         %check for 2 supports on top of each other and delete one -> 2 supports
980             %with the same locked direction and the same node location number
981             %identical commands were already deleted but also possible:
982             %e.g. DX,2 and D90,2 -> 2 for loops to compare 2 supports
983         for n=1:height(D{1})                %for every support command in D
984             for m=1:height(D{1})            %for every support command in D
985                 %check if two different supports are compared and check if the
986                     %first support has a numeral entry (angle entry) and the
987                     %node location number is the same
988                 if n~=m && all(char(D{2}(m,1))==char(D{2}(n,1)))
989                     %check if the angle is the same,options for the same angle:
990                         %1. same angle entry (one angle was corrected above)
991                         %2. same lcoked direction, angle is 180° rotated
992                         %3. one locked direction is Y and other one is 0°,180°
993                             %or -180° -> angle divided by 180 and rounded is a
994                             %whole number
995                         %4. one locked direction is X and other one is 90° or
996                             %-90°-> angle divided by 180 and rounded is a
997                             %whole number
998                     if str2double(D{1}(m,1))==str2double(D{1}(n,1)) ||...
999                          str2double(D{1}(m,1))==str2double(D{1}(n,1))+180 ||...
1000                         str2double(D{1}(m,1))==str2double(D{1}(n,1))+360 ||...
1001                         (round(str2double(D{1}(n,1))/180)==...
1002                         str2double(D{1}(n,1))/180 && all(char(D{1}(m,1))=='Y'))...
1003                         ||(round(str2double(D{1}(n,1))/90)==...
1004                          str2double(D{1}(n,1))/90 && all(char(D{1}(m,1))=='X'))
1005                         %increase the faults count
1006                         faults_count=faults_count+1;
1007                         %search for the fault line of the first support command
```

```
1008                        %in the input text lines
1009                    fault_line=find(contains(input_text_lines,['D',...
1010                        char(D{1}(n,1)),',',char(D{2}(n,1))]));
1011                    %search for the fault line of the second support
1012                        %command in the input text lines
1013                    fault_line_2=find(contains(input_text_lines,['D',...
1014                        char(D{1}(m,1)),',',char(D{2}(m,1))]));
1015                    %print the fault into output file with the
1016                        %corresponding line from the input text file
1017                        %inform the user the second support is deleted
1018                    fprintf(Output,['Line %d and %d in input text file: '...
1019                        'Truss assembly fault detected!\n'],...
1020                        fault_line,fault_line_2);
1021                    fprintf(Output,['Two supports are locking the same '...
1022                        'direction at the same node: %s  %s\nSupport '...
1023                        '%s is deleted.\n\n'],...
1024                        char(input_text_lines(fault_line,1)),...
1025                        char(input_text_lines(fault_line_2,1)),...
1026                        char(input_text_lines(fault_line_2,1)));
1027                    %delete the arguments of the second support command by
1028                        %creating empty arguments, deleted after for loop
1029                        %because there  would be an error in for loop
1030                        %addressing m>heightD{1}
1031                    D{1}(m,:)=cellstr('');D{2}(m,:)=cellstr('');
1032                end
1033            end
1034        end
1035    end
1036    %delete the empty arguments in the cells
1037    D{1}=D{1}(~cellfun('isempty',D{1}));
1038    D{2}=D{2}(~cellfun('isempty',D{2}));
1039
1040    %inform the user on the amount of minor faults that were correced if
1041        %there were any corrected
1042    if faults_count >0
1043        fprintf(Output,'%d minor Fault(s) were corrected.\n\n',...
1044            faults_count);
1045    end
1046 end
1047 %% %assembly fault
1048 %check for truss assembly fault -> elements wrong assembled
1049    %different start and end node number in rods
1050    %2 nodes on top of each other (same coordinates)
1051    %more than 2 supports per node
1052    %every node needs at least 2 rods or one rod and one support
1053    %a node with only 2 rods attached and a unit vector in the same
1054        %direction is moveable
1055    %nothing is returned, programm is stopped when a such a major fault was
1056    %found
1057    %before stopping the programm the figure of the truss is created by
1058    %running the sub programms Transfer_to_double_matrices and Draw_Truss
1059    %-> also F,folder_path and input_name is needed in the function
1060 function assembly_fault(input_text_lines,N,D,E,F,Output,folder_path,...
```

```
1061     input_name)
1062     %major_faults_count the number of major faults, further
1063     %processing is not possible and programm stops
1064     major_faults_count=0;
1065
1066     %check for different start and end node number in rods
1067     for n=1:height(E{1})         %for every rod command in E
1068         %check if the start and end node numbers are the same
1069         if str2double(E{2}(n,1))==str2double(E{3}(n,1))
1070             %increase the major faults count
1071             major_faults_count=major_faults_count+1;
1072             %search for the fault line of the command in the
1073                     %input text lines
1074             fault_line=find(contains(input_text_lines,['E,',...
1075                 char(E{1}(n,1)),',',char(E{2}(n,1)),',',char(E{3}(n,1))]));
1076             %print the fault into output file with the
1077                     %corresponding lines from the input text file
1078             fprintf(Output,['Line %d in input text file: Major Truss '...
1079                 'assembly fault detected!\n'],fault_line);
1080             fprintf(Output,['The beginning and end node are the same:'...
1081                 ' %s\n\n'],char(input_text_lines(fault_line,1)));
1082         end
1083     end
1084
1085     %check for 2 nodes on top of each other (same coordinates)
1086         %one node can't be deleted because the node number might be used in
1087         %another elements, 2 nodes are compared -> 2 for loops
1088     for n=1:height(N{1})                 %for every node command in N
1089         for m=1:height(N{1})             %for every node command in N
1090             %check if two different nodes are compared and make sure the
1091                 %two nodes are only compared once -> n<m
1092                 %check if the x-coordinates and y-coordinates are the same
1093             if n<m &&...
1094                 str2double(N{2}(n,1))==str2double(N{2}(m,1)) &&...
1095                 str2double(N{3}(n,1))==str2double(N{3}(m,1))
1096                 %increase the major faults count
1097                 major_faults_count=major_faults_count+1;
1098                 %search for the fault line of the first node command in the
1099                     %input text lines
1100                 fault_line=find(contains(input_text_lines,['N,',...
1101                     char(N{1}(n,1)),',',char(N{2}(n,1)),',',...
1102                     char(N{3}(n,1))]));
1103                 %search for the fault line of the second node command in
1104                     %the input text lines
1105                 fault_line_2=find(contains(input_text_lines,['N,',...
1106                     char(N{1}(m,1)),',',char(N{2}(m,1)),',',...
1107                     char(N{3}(m,1))]));
1108                 %print the fault into output file with the
1109                     %corresponding lines from the input text file
1110                 fprintf(Output,['Line %d and %d in input text file: '...
1111                     'Major Truss assembly fault detected!\n'],...
1112                     fault_line,fault_line_2);
1113                 fprintf(Output,['Two Nodes with the same coordinates '...
```

```
1114                     '(on top of each other):\n\t %s  %s\n\n'],...
1115                     char(input_text_lines(fault_line,1)),...
1116                     char(input_text_lines(fault_line_2,1)));
1117             end
1118         end
1119     end
1120 
1121     %check for more than 2 supports per node -> no statically defined truss
1122         %possible, 3 for loops are needed to search for 3 supports per node
1123     for n=1:height(D{1})                        %for every support command in D
1124         for m=1:height(D{1})                    %for every support command in D
1125             for o=1:height(D{1})                %for every support command in D
1126                %check if the node locations of the three nodes are the same
1127                     %(first & second and first & third are the same)
1128                     %check if three different nodes are compared and make
1129                     %sure the three nodes are only compared once -> n<m &
1130                     %m<o
1131                 if str2double(D{2}(n,1))==str2double(D{2}(m,1)) && ...
1132                     str2double(D{2}(n,1))==str2double(D{2}(o,1))&&n<m&&m<o
1133                 %increase the major faults count
1134                 major_faults_count=major_faults_count+1;
1135                 %search for the fault line of the command in the
1136                     %input text lines
1137                 fault_line=find(contains(input_text_lines,['D',...
1138                     char(D{1}(n,1)),',',char(D{2}(n,1))]));
1139                 fault_line_2=find(contains(input_text_lines,['D',...
1140                     char(D{1}(m,1)),',',char(D{2}(m,1))]));
1141                 fault_line_3=find(contains(input_text_lines,['D',...
1142                     char(D{1}(o,1)),',',char(D{2}(o,1))]));
1143                 %print the fault into output file with the
1144                     %corresponding lines from the input text file
1145                 fprintf(Output,['Line %d , %d and %d in input text file'...
1146                     ': Major Truss assembly fault detected!\n'],...
1147                     fault_line,fault_line_2,fault_line_3);
1148                 fprintf(Output,['One node with 3 supports, no staticall'...
1149                     'y defined truss: %s  %s  %s\n\n'],...
1150                     char(input_text_lines(fault_line,1)),...
1151                     char(input_text_lines(fault_line_2,1)),...
1152                     char(input_text_lines(fault_line_3,1)));
1153                 end
1154             end
1155         end
1156     end
1157 
1158     %check that every Node has at least 2 Rods or one Rod and a bearing
1159         %-> otherwise it is moveable and the truss it not kinematically
1160         %defined
1161     for n=1:height(N{1})                %for every node command in N
1162         %check if the sum of the one node number in the beginning node
1163             %numbers in rods, in the end node numbers in rods and node
1164             %numbers in support are less than 2
1165             %str2num is used because str2double can't convert vectors
1166        if sum([str2num(char(E{2}));str2num(char(E{3}));...
```

```
1167                 str2num(char(D{2}))]==str2double(N{1}(n,1)))<2
1168              %increase the major faults count
1169              major_faults_count=major_faults_count+1;
1170              %search for the fault line of the node command in the
1171                  %input text lines
1172              fault_line=find(contains(input_text_lines,['N,',char(N{1}(n,1)),...
1173                  ',',char(N{2}(n,1)),',',char(N{3}(n,1))]));
1174              %print the fault into output file with the
1175                  %corresponding line from the input text file
1176              fprintf(Output,['Line %d in input text file: Major Truss assembly'..
1177                  ' fault detected!\n'],fault_line);
1178              fprintf(Output,['One node is moveable, no kinematically defined trus
1179                  ': %s \n\n'],char(input_text_lines(fault_line,1)));
1180         end
1181      end
1182
1183      %if major faults were found, calculation not possible-> stop
1184          %still possible to generate visual output -> run sub programms
1185          %Transfer_to_double_matrices and Draw_Truss
1186      if major_faults_count >0
1187          %sub programme to transfer the input informations from
1188              %cell arrays into two dimensional double matrices
1189          run('Transfer_to_Double_Matrices.m')
1190          %sub programme to draw the Truss with all Nodes, Rods, Support and
1191              %Forces into an diagramme and saves it as a svg file
1192          run('Draw_Truss.m')
1193
1194          %write number of major faults into output file
1195          fprintf(Output,['%d Major Fault(s) were detected.\n'...
1196              'Further Truss processing stopped!\nCorrect the faults in the '...
1197              'input text file and restart the programme.'], major_faults_count);
1198          %close the text file
1199          fclose(Output);
1200          %stop the programme with an error and inform the user in a message
1201              %box, print the link to the ouput file in the error (link cant
1202              %be printed to the message box)
1203          msgbox(['Major assembly Fault(s) detected, further processing '...
1204             'stopped! Regard to the output file and the graphical output'...
1205              ' for more informations. Linnk in the Command Window.'],...
1206              'Error','error');
1207          error(['<strong>Major Fault(s) detected, further processing '...
1208              'stopped! Link to Output files:</strong> <a href="',...
1209              folder_path,'">path</a>']);
1210      end
1211  end
1212  %% % degree of static (in-)determinacy
1213      %checks if the truss has the necessary condition for statically
1214      %defined
1215      %calculation of the degree of static (in-)determinacy
1216      %nothing is returned, programm is stopped when a such a major fault was
1217      %found
1218      %before stopping the programm the figure of the truss is created by
1219      %running the sub programms Transfer_to_double_matrices and Draw_Truss
```

```
1220     %-> also F,folder_path and input_name is needed in the function
1221 function statically_defined(N,D,E,F,Output,folder_path,input_name)
1222     %calculate the degree of static (in-)determinacy
1223     j=height(E{1})+height(D{1})-2*height(N{1});
1224
1225     %check if degree is greater than 0 -> statically overdetermined
1226         %no calculation is possible -> stop
1227         %still possible to generate visual output -> run sub programms
1228         %Transfer_to_double_matrices and Draw_Trus
1229     if j>0
1230         %sub programme to transfer the input informations from
1231             %cell arrays into two dimensional double matrices
1232         run('Transfer_to_Double_Matrices.m')
1233         %sub programme to draw the Truss with all Nodes, Rods, Support and
1234             %Forces into an diagramme and saves it as a svg file
1235         run('Draw_Truss.m')
1236
1237         %inform the user of overdetermined truss in output file
1238         fprintf(Output,['Major Fault: The truss is %d times statically '...
1239             'overdetermined.\nThe truss can''t be solved using the node'...
1240             ' methode.\nReduce the numbers of rods and supports or '...
1241             'create more nodes!\n\n'], j);
1242         %close the text file
1243         fclose(Output);
1244         %stop the programme with an error and inform the user in a message
1245             %box, print the link to the ouput file in the error (link cant
1246             %be printed to the message box)
1247         msgbox(['Truss is not statically defined, further processing '...
1248             'stopped! Regard to the output file for more informations.'...
1249             ' Linnk in the Command Window.'],'Error','error');
1250         error(['<strong>Major Fault(s) detected, further processing '...
1251             'stopped! Link to Output file:</strong> <a href="',...
1252             folder_path,'">path</a>']);
1253     %else check if degree is greater than 0 -> statically underdetermined
1254         %no calculation is possible -> stop
1255         %still possible to generate visual output -> run sub programms
1256         %Transfer_to_double_matrices and Draw_Trus
1257     elseif j<0
1258         %sub programme to transfer the input informations from
1259             %cell arrays into two dimensional double matrices
1260         run('Transfer_to_Double_Matrices.m')
1261         %sub programme to draw the Truss with all Nodes, Rods, Support and
1262             %Forces into an diagramme and saves it as a svg file
1263         run('Draw_Truss.m')
1264
1265         %inform the user of underdetermined truss in output file
1266         fprintf(Output,['Major Fault: The truss is %d times statically'...
1267             ' underdetermined.\nThe truss is moveable.\nCreate more'...
1268             ' rods and supports or reduce the number of nodes!\n\n'], -j);
1269         %close the text file
1270         fclose(Output);
1271         %stop the programme with an error and inform the user in a message
1272             %box, print the link to the ouput file in the error (link cant
```

```
1273            %be printed to the message box)
1274        msgbox(['Truss is not statically defined, further processing '...
1275            'stopped! Regard to the output file for more informations'...
1276            ' and a graphical output. Linnk in the Command Window.'],...
1277            'Error','error');
1278        error(['<strong>Major Fault(s) detected, further processing '...
1279            'stopped! Link to Output file:</strong> <a href="',...
1280            folder_path,'">path</a>']);
1281    end
1282 end
```

C:\Users\maxjo\OneDr...\Transfer to Double Matrices.m 1 of 3

```
1 %% %2D-Truss Analysis and Calclulation Software
2 %Bachelor Thesis, Max Stegmann
3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
4
5 %% %sub programme: Transfer_to_Double_Matrices
6     %the sub programme transfers the information from
7     %the cell arrays into double matrices,
8     %calculate the length and direction of the Rods
9
10 %str2num function needs to be used when a cell vector is copied
11     %because str2double can converts only one
12     %cell argument and not a cell vector
13
14 %copy node information into the Nodes-matrix
15     %1.column: node number (from N); 2.column: x-coordinate(from N);
16     %3.column: y-coordinate (from N)
17 Nodes=[str2num(char(N{1})) str2num(char(N{2})) str2num(char(N{3}))];
18 %sort nodes in the matrix by node-number
19 Nodes = sortrows(Nodes, 1);
20
21 %copy rod information into the Rods-matrix
22     %1.column: rod number(from E); 2.column: beginning node(from E);
23     %3.column: end node(from E);
24 Rods=[str2num(char(E{1})),str2num(char(E{2})),str2num(char(E{3}))];
25 %sort rods in the matrix by rod-number
26 Rods = sortrows(Rods, 1);
27 %if a node number is left out the beginning or end node number in the Rods
28     %matrix doesn't match with the row of the node in the Nodes matrix
29     %the beginning node and end node should refer to the row of the node in
30     %the Nodes matrix
31 for n=1:height(Rods)                %for every rode
32    %find searches for the row of the node number in the Nodes matrix that
33         %equals the beginning and end node number
34    Rods(n,2)=find(Nodes(:,1)==Rods(n,2));
35    Rods(n,3)=find(Nodes(:,1)==Rods(n,3));
36 end
37 %loop calculates the length and direction of every Rod
38     %column 4: length of the rod
39     %column 5: x component of the unit vector of the direction
40     %column 6: y component of the unit vector of the direction
41 for n=1:height(Rods)        %for every rod
42     %calculate length of rods and write it into a new column
43         %in the Rods-matrix, vector from the rod is calculated by
44         %subtracting the x and y value of the start & end node
45         %the length is calculated with the absolute value of the vector
46     Rods(n,4)=sqrt((Nodes(Rods(n,2),2)-Nodes(Rods(n,3),2))^2+...
47         (Nodes(Rods(n,2),3)-Nodes(Rods(n,3),3))^2);
48     %calculate the normalised direction vector and put the proportion
49         %of the x- and y-direction in a new column
50     %x-direction component into 5th column
51         %coordinate of end Node minus coordinate of start Node, the
52         %vector is normalised by dividing by the Rod length
53     Rods(n,5)=(Nodes(Rods(n,3),2)-Nodes(Rods(n,2),2))/Rods(n,4);
```

```
54      %y-direction proportion into 6th column
55      Rods(n,6)=(Nodes(Rods(n,3),3)-Nodes(Rods(n,2),3))/Rods(n,4);
56  end
57
58  %copy Support information into the Support-matrix
59      %1.column: node location (from D)
60  Supports(:,1)=str2num(char(D{2}));
61  %column 2 and 3 containe the normalised normal vector of the
62      %support planes, the plane in which the support is able to slide
63      %column 2 the x component and column 3 the y component of the vector
64  for n=1:height(D{1,1})          %for every Support
65     %find searches for the row of the node number in the Nodes matrix that
66          %equals the node number where support is located (same as Rods)
67     Supports(n,1)=find(Nodes(:,1)==Supports(n,1));
68       %check if in the n'th row is a X as locked direction
69       if char(D{1}(n,1))=='X'
70           %the normalised normal vector from the support plane of a
71                  %support locking the x-direction is [1,0]
72           Supports(n,2)=1;
73       %else check if in the n'th row is a Y as locked direction
74       elseif char(D{1}(n,1))=='Y'
75           %the normalised normal vector from the support plane of a
76                  %support locking the y-direction is [0,1]
77           Supports(n,3)=1;
78       %else the support normal vector was defined with the angle in
79              %degreese to the vertical line
80       else
81           %the normalised normal vector from the support plane is the
82               %cosinus (y-component) and sinus (x-component) of the angle
83               %to the vertical line
84           Supports(n,2)=-sind(str2double(D{1}(n,:)));
85           Supports(n,3)=cosd(str2double(D{1}(n,:)));
86       end
87  end
88
89  %copy force information into force-matrix
90      %1.column: node location (from F) 4.column: force value
91  Forces(:,1)=str2num(char(F{2}));
92  Forces(:,4)=str2num(char(F{3}));
93  %column 2 and 3 containe the normalised vector of the direction in
94      %which the force acts
95      %column 2 the x component and column 3 the y component of the vector
96   for n=1:height(F{1,1})                     %for every force
97     %find searches for the row of the node number in the Nodes matrix that
98          %equals the node number where the force engages (same as Rods)
99     Forces(n,1)=find(Nodes(:,1)==Forces(n,1));
100      %check if in the n'th row is a X as acting direction
101      if char(F{1}(n,1))=='X'
102          %set the x-component of the normalized direction vector
103             %to 1 in the Forces-matrix
104          Forces(n,2)=1;
105      %else check if in the n'th row is a Y as acting direction
106      elseif char(F{1}(n,1))=='Y'
```

```
107         %set the y-component of the normalized direction vector
108            %to 1 in the Forces-matrix
109         Forces(n,3)=1;
110     end
111 end
```

```
1 %% %2D-Truss Analysis and Calclulation Software
2 %Bachelor Thesis, Max Stegmann
3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
4
5 %% %sub programme: Draw_Truss
6     %the sub programme draws the truss with the truss information
7     %from the double matrices into a diagramme
8
9 %Truss is the diagramme
10 Truss=figure('visible','off');
11 %write as titel: Truss of (file name), turning off Interpreter will
12     %read underscores or circ
13 title(['Truss of ', input_name],'Interpreter','none','FontSize',18);
14 %plot will not be overwritten
15 hold on;
16 %display grid lines
17 grid on;
18 %background white
19 set(Truss,'Color',[1 1 1]);
20 %adjust the size and position of the figure-window to full screen
21 set(Truss,'Units','normalized', 'Position', [0, 0, 1, 1]);
22
23 %functions draws all the elements and the necessary text into the Truss
24     %figure, returned is the figure Truss
25 Truss=elements_into_figure(Nodes,Rods,Supports,Forces,Truss);
26
27 %thin margin from axis to Truss
28 axis padded;
29 %Use the same length for the data units along each axis.
30 axis equal;
31 %save the figure as a scalable vector graphic (.svg) in the path of the
32     %input text file
33 saveas(Truss,[folder_path,input_name(1:end-4),'-FIGURE.svg'])
34 %close the Truss
35 close(Truss);
36 %%
37 %functions draws all the elements and the necessary text into the Truss
38     %figure
39     %returned is the figure Truss
40 function Truss=elements_into_figure(Nodes,Rods,Supports,Forces,Truss)
41      %draw the Nodes as circles
42     plot(Nodes(:,2),Nodes(:,3),'o','MarkerSize',10,'MarkerFaceColor',...
43         'b','MarkerEdgeColor','b');
44
45     %number of rows in Rod-matrix=number of Rods
46         %for every Rod draw a line and write the Rod number in the middle
47     for n=1:height(Rods)
48         %plot line/Rod from start- to end-Node
49         plot([Nodes(Rods(n,2),2),Nodes(Rods(n,3),2)],...
50             [Nodes(Rods(n,2),3),Nodes(Rods(n,3),3)],'b',...
51             'LineWidth',1 );
52         %write a "R" and the number of the Rod in the middle of the Rod
53             %line, the middle is calculated by the Rod-direction in column
```

```
54              %5 and 6 and the Rod length in column 4 (direction is a
55              %normalised vector and needs to be multiplied by the length)
56          text(Nodes(Rods(n,2),2)+Rods(n,4)*0.5*Rods(n,5),...
57                  Nodes(Rods(n,2),3)+Rods(n,4)*0.5*Rods(n,6),...
58                  num2str(Rods(n,1)),'FontSize',13,...
59                  'HorizontalAlignment','center');
60      end
61
62      %write the maximum Force into maxforce to later calculate the
63          %force arrow length depending on the magnitude of the force
64      maxforce=max(abs(Forces(:,4)));
65      %calculate the average Rods length to
66          %later adjust the length of the force arrows
67      Rodsl=mean(Rods(:,4));
68
69      %number of rows in Forces-matrix=number of Forces
70          %for every force draw a arrow in the the direction in which the
71          %force works
72      for n=1:height(Forces)
73          %force arrow length depends on the ratio to the maximum force and
74          %the average rods length
75          forcel=abs(Forces(n,4))/maxforce*Rodsl/2;
76          %if number in column one is a 1, force works in x-Direction
77          if Forces(n,2)==1
78              %draw a arrow depending on the size of the force from the node
79                  %location to the right in x-direction, maximum force has
80                  %the length of the average Rods length
81              plot([Nodes(Forces(n,1),2),...
82                  Nodes(Forces(n,1),2)+forcel],...
83                  [Nodes(Forces(n,1),3),Nodes(Forces(n,1),3)],...
84                  'r->','MarkerIndices',2,'LineWidth',2,'MarkerSize',9,...
85                  'MarkerFaceColor','r','MarkerEdgeColor','r')
86          %else if number in column one is a 2, force works in y-Direction
87          elseif Forces(n,3)==1
88              %draw a arrow depending on the size of the force from the node
89                  %location up in y-direction, maximum force has the length
90                  %of the average Rods length
91               plot([Nodes(Forces(n,1),2),Nodes(Forces(n,1),2)],...
92                   [Nodes(Forces(n,1),3),...
93                   Nodes(Forces(n,1),3)+forcel],...
94                  'r-^','MarkerIndices',2,'LineWidth',2,'MarkerSize',9,...
95                  'MarkerFaceColor','r','MarkerEdgeColor','r')
96          end
97      end
98
99      %thin margin from axis to Truss
100     axis padded;
101     %Use the same length for the data units along each axis.
102     axis equal;
103     %xlim containes the maximum and minimum x-coordinate of the graph
104     xl = xlim;
105     %difference between these values=width and height of the graph to later
106         %adjust the offset of the Support-triangles and text next to Forces
```

```
107         %for different sections of the axis, the offset is still the same
108     xl=xl(1,2)-xl(1,1);
109
110     %number of rows in Supports-matrix=number of Supports
111         %for every locked direction by a Support draw a triangle
112     for n=1:height(Supports)
113         %alpha is the angle of the normal vector of the support to the
114             %vertical axis
115         alpha=-atan2d(Supports(n,2),Supports(n,3));
116         %fill a isosceles triangle(60° angles) at the corresponding node
117             %first point of the triangle is the middle of the corresponding
118             %node
119             %other 2 points are rotated around the corresponding node
120             %depending on the angle of the normal vector to the horizontal
121             %line
122         fill([Nodes(Supports(n,1),2),...
123             Nodes(Supports(n,1),2)-sind(alpha+60+90)*xl/30,...
124             Nodes(Supports(n,1),2)-sind(alpha+120+90)*xl/30],...
125             [Nodes(Supports(n,1),3),...
126             Nodes(Supports(n,1),3)+cosd(alpha+60+90)*xl/30,...
127             Nodes(Supports(n,1),3)+cosd(alpha+120+90)*xl/30],[0 0.8 0],...
128             'LineStyle','none')
129     end
130
131     %write Node number as roman numeral right of every Node
132         %therefore latex interpreter need to be used with the functions
133         %\MakeUppercase and \romannumeral
134     %a for loop needs to be introduced with latex interpreter
135     for n=1:height(Nodes)                  %for every node
136         text(Nodes(n,2)+0.012*xl,...
137             Nodes(n,3)+0.012*xl,...
138             ['\MakeUppercase{\romannumeral',num2str(Nodes(n,1)),'}'],...
139             'FontSize',14,'HorizontalAlignment','left','Interpreter',...
140             'latex');
141     end
142
143     %number of rows in Forces-matrix=number of Forces
144         %for every force draw a arrow in the the direction in which the
145         %force works
146     for n=1:height(Forces)
147         %force arrow length depends on the ratio to the maximum force and
148         %the average rods length
149         forcel=abs(Forces(n,4))/maxforce*Rodsl/2;
150         %if number in column one is a 1, force works in x-Direction
151         if Forces(n,2)==1
152             %write the size of the force over the arrow
153             text(Nodes(Forces(n,1),2)+forcel,...
154                 Nodes(Forces(n,1),3)+0.020*xl,...
155                 string(Forces(n,4)),'FontSize',14,...
156                 'HorizontalAlignment','center');
157         %else if number in column one is a 2, force works in y-Direction
158         elseif Forces(n,3)==1
159             %write the size of the force right to the arrow
```

C:\Users\maxjo\OneDrive\Desktop\Studi...\Draw Truss.m

```
160             text(Nodes(Forces(n,1),2)+0.015*xl,...
161                 Nodes(Forces(n,1),3)+forcel,...
162                 string(Forces(n,4)),'FontSize',14,...
163                 'HorizontalAlignment','left');
164         end
165     end
166 end
```

C:\Users\maxjo\OneDrive\Desktop\Stud...\Calculation.m 1 of 2

```
1 %% %2D-Truss Analysis and Calclulation Software
2 %Bachelor Thesis, Max Stegmann
3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
4
5 %% %sub programme: Calculation
6     %the sub programme sets up the linear system of equotation of the Truss
7     %K*a=f
8     %with coefficient matrix K and external force vector f
9     %solving the linear systeme of equotation returning the Rod forces and
10    %Support reactions
11
12 %function returns the koefficient matrix K and external force vector f
13   %to set up the linear system of equations of the Truss K*a=f
14 [K,f]=assembly_of_K_f(Nodes, Rods, Supports, Forces);
15
16 %the system of equations can be solved unambiguously if the determinant of
17     %the coefficient matrix is not equal to zero
18 if det(K)~=0
19     %solve the system of lineare equotations
20         %solution vector a contains the scalar unknown Rod Forces and
21         %support reactions
22     a=K\f;
23     %copy the solution for the Support reactions into a new column in
24         %Supports
25     Supports(:,4)= a(1:height(Supports),1);
26     %copy the solution for the Rod Forces into a new column into Rods
27     Rods(:,7)=a(height(Supports)+1:height(Supports)+height(Rods),1);
28 end
29 %% %
30 function [K,f]=assembly_of_K_f(Nodes, Rods, Supports, Forces)
31     %K is the coefficient Matrix with one force balance equotation for
32         %every Node(cutting free every Node) in x- and y-Direction
33         %-> number of rows = two times the number of Nodes;
34         %K describes how Rod Forces and Support Reactions are adjoining at
35         %the Nodes in x- and y-direction -> number of columns = number
36         %of Rods plus number of Support Reactions
37     K=zeros(2*height(Nodes),height(Rods)+height(Supports));
38     %the external Forces vector contains the proportion of the external
39         %Forces at every Node-equotation
40         %->number of rows = number of Node equotations
41     f=zeros(2*height(Nodes),1);
42
43     for n=1:height(Nodes)         %for every Node
44
45         %kx ist the row in the Matrix K corresponding to the force balance
46         %of Node n x-Direction
47         kx=(n-1)*2+1;
48         %ky ist the row in the Matrix K corresponding to the force balance
49         %of Node n y-Direction
50         ky=(n-1)*2+2;
51
52         for m=1:height(Supports)         %for every Support
53             %if Support  reaction is adjoining at the Node
```

C:\Users\maxjo\OneDrive\Desktop\Stud...\Calculation.m 2 of 2

```
54              if Supports(m,1)==n
55                  %put the normalised x-direction proportion of the support
56                      %reaction vector in the the K matrix
57                  K(kx,m)=Supports(m,2);
58                  %put the normalised y-direction proportion of the support
59                      %reaction vector in the the K matrix
60                  K(ky,m)=Supports(m,3);
61              end
62          end
63
64          for m=1:height(Rods)        %for every Rod
65              %if the Rod containes the Node-number as start point
66              if Rods(m,2)==n
67                  %put the normalised x-direction proportion of the Rod force
68                      %vector in the K matrix
69                  K(kx,height(Supports)+m)=Rods(m,5);
70                  %put the normalised y-direction proportion of the Rod force
71                      %vector in the K matrix
72                  K(ky,height(Supports)+m)=Rods(m,6);
73              %if the Rod containes the Node-number as end point
74              elseif Rods(m,3)==n
75                   %do the same as above with the difference that the
76                      %negative value needs to be put in the K matrix
77                   K(kx,height(Supports)+m)=-Rods(m,5);
78                   K(ky,height(Supports)+m)=-Rods(m,6);
79              end
80          end
81
82         %set up the external force vector
83         for m=1:height(Forces)       %for every force
84             %if force engages at the Node
85             if Forces(m,1)==n
86                 %if force acts in x-direction
87                 if Forces(m,2)==1
88                    %put the negative force value into the ex. force vector
89                    f(kx,1)=-Forces(m,4);
90                 %if force acts in y-direction
91                 elseif Forces(m,3)==1
92                    %put the negative force value into the ex. force vector
93                     f(ky,1)=-Forces(m,4);
94                 end
95             end
96         end
97     end
98 end
```

```
 1 %% %2D-Truss Analysis and Calclulation Software
 2 %Bachelor Thesis, Max Stegmann
 3 %HAW Hamburg, Department of Automotive and Aeronautical Engineering
 4 
 5 %% %sub programme: Write_into_Output_File
 6     %the sub programme sets up the coefficient Matrix with one
 7     %equotation for every Node and then solves the linear systeme of
 8     %equotation returning the Rod and Support Forces
 9     %it saves the koefficientmatrix K, the external force vektor f and the
10     %solution a in a .mat file
11 
12 %open the output text file
13 Output=fopen([folder_path,output_name],'a');
14 
15 %check if the determinant of the coefficient matrix K is zero or close to
16     %zero -> if zero the equation system is not solvable
17 if abs(det(K))<1.0e-10
18     %inform the user that rod forces and support reactions are not
19         %solveable
20     fprintf(Output,'%s\n%s\n',['          ATTENTION: SYSTEM OF '...
21         'EQUATIONS CANNOT BE SOLVED!'],...
22         ['*************************************************************'...
23         '*********']);
24     fprintf(Output,'%s\n%s\n%s\n%s\n\n',...
25         'The determinant of the coefficient matrix is zero: det(K)=0.',...
26        'The linear system of equations cannot be solved unambiguously!',...
27         'The truss is probably movable.',...
28         ['Check the pictorial output and adjust the truss in the input'...
29         ' text file.']);
30 % if determinant is smaller than 0.1 (close to zero) there will be high
31     %forces in the truss,
32 elseif abs(det(K))<0.1
33     %inform the user that there are high forces and the truss can possibly
34         %adjusted
35     fprintf(Output,'%s\n%s\n','                              Attention',...
36         ['*************************************************************'...
37         '*********']);
38     fprintf(Output,'%s%d\n%s\n%s\n\n',...
39         'The determinant of the coefficient matrix is close to zero: ',...
40        det(K),'There will be high rod forces and or bearing reactions!',...
41         ['The truss can possibly be adjusted for better power '...
42         'transmission. ']);
43 end
44 
45 %print output title
46 fprintf(Output,'%s\n%s\n','                              OUTPUT',...
47     ['*****************************************************************'...
48     '*****']);
49 
50 %function prints the input informations: external forces and nodes into the
51     %output file
52     %returned is nothing, the nodes and forces informations are printed
53     %to the output file
```

```
54 nodes_forces_output(Output, Nodes, Forces)
55
56 %function prints the rods and support informaions into the output file
57     %the solution of the equotation system is available and printed to the
58     %output file if the determinant of the coefficient matrix is not zero
59     %returned is nothing, the rods and support informations are printed
60     %to the output file
61 rods_supports_output(Output,K,Rods,Supports);
62
63 %inform the user about the determinant and the system of equations
64 fprintf(Output,'\n%s\n%s\n%s%d\n\n%s\n%s\n%s',['The system of equatio'...
65     'ns K*a=f can be solved unambiguously if the'],['determinant of the'...
66     ' coefficient matrix is not equal to zero:'],'det(K)=',det(K),...
67     'The coefficient matrix K, the external force vector f and the',...
68     ['solution a (when system of equations is solveable) was '...
69     'saved'],' as a -.mat file to the folder.');
70
71 %function prints the whole linear system of equations of the truss to the
72     %output file
73     %returned is nothing, the coefficient matrix, the external force vector
74     %and force vector is printed to the output file
75 %equotation_system_output(Output,Nodes,Rods,Supports,K,f)
76
77 if abs(det(K))<1.0e-10
78     %save the variables K,f and a (linear system of equotation)
79     save([folder_path,input_name(1,1:end-4),'-VAR.mat'],'K','f')
80 else
81     %save the variables K,f and a (linear system of equotation)
82     save([folder_path,input_name(1,1:end-4),'-VAR.mat'],'K','f','a')
83 end
84
85 %close the solution text file
86 fclose(Output);
87 %%
88 %function prints the input informations: external forces and nodes into the
89     %output file
90     %returned is nothing, the nodes and forces informationsare  printed
91     %to the output file
92 function nodes_forces_output(Output, Nodes, Forces)
93     %print the input informations Nodes and Forces
94         %print the table titel of the input Node Coordinates to the
95         %output text file: Node Number, x-Coordinate, y-Coordinate
96     fprintf(Output,'%s\n','Node Coordinates');
97     fprintf(Output,'%11s\t%11s\t%11s\n','Node Number','x-Coordinate',...
98         'y-Coordinate');
99     %next loop prints the node informations from the corresponding column
100            %for every node into the table
101     for n=1:height(Nodes)
102        fprintf(Output,'%-11.9g\t%-11.9g\t%-11.9g\n', Nodes(n,1),...
103            Nodes(n,2),Nodes(n,3));
104     end
105
106     %print the table titel of the input force Coordinates to the output
```

```
107         %text file: At Node, Direction[x,y], Value
108     fprintf(Output,'\n%s\n','External Forces');
109     fprintf(Output,'%7s\t\t%14s\t\t%5s\n','At Node','Direction[x,y]',...
110         'Value');
111     %next loop prints the force informations from the corresponding column
112             %for every force into the table
113     for n=1:height(Forces)
114        fprintf(Output,'%-8.6g\t[%7.4g, %7.4g]\t%+-8.6g\n', Forces(n,1),...
115            Forces(n,2), Forces(n,3),Forces(n,4));
116     end
117 end
118 
119 %function prints the rods and support informaions into the output file
120     %the solution of the equotation system is available and printed to the
121     %output file if the determinant of the coefficient matrix is not zero
122     %returned is nothing, the rods and support informations are printed
123     %to the output file
124 function rods_supports_output(Output,K,Rods,Supports)
125     %if the deteminant of the coefficient matrix is zero or close to zero
126         %there is no solution for rod forces and support reactions
127         %-> solution needs to be left out
128     if abs(det(K))>1.0e-10
129         %print the table titel of the Support Reactions to the output text
130             %file At Node, Direction[x,y], Reaction
131         fprintf(Output,'\n%s\n','Support Reactions');
132         fprintf(Output,'%7s\t\t%14s\t\t%8s\n','At Node',...
133             'Direction[x,y]','Reaction');
134        %next loop prints the support informations from the corresponding
135             %column for every support into the table
136         for n=1:height(Supports)
137            fprintf(Output,'%-8.6g\t[%7.4g, %7.4g]\t%+-8.6g\n',...
138                 Supports(n,1),Supports(n,2), Supports(n,3),Supports(n,4));
139         end
140         %print the table titel of the Rods to the output text file
141             %Rod Number, Start-Node, End-Node, Direction[x,y], Force
142         fprintf(Output,'\n%s\n','Rod-Forces');
143         fprintf(Output,'%10s\t%10s\t%8s\t%14s\t\t%5s\n','Rod Number',...
144             'Start-Node','End-Node','Direction[x,y]','Force');
145         %next loop prints the rod informations from the corresponding
146             %column for every rod into the table
147         for n=1:height(Rods)
148            fprintf(Output,['%-11.9g\t%-11.9g\t%-9.7g\t[%7.4g, %7.4g]',...
149                 '\t%+-8.6g\n'],Rods(n,1),Rods(n,2),Rods(n,3),Rods(n,5),...
150                 Rods(n,6),Rods(n,7));
151         end
152     %else the determinant is not zero and there are solutions for the rod
153         %forces and support reaction that can be printed to the output file
154     else
155         %print the table titel of the Support Reactions to the output text
156             %file At Node, Direction[x,y], Reaction
157         fprintf(Output,'\n%s\n','Supportss');
158         fprintf(Output,'%7s\t\t%14s\n','At Node','Direction[x,y]');
159        %next loop prints the support informations from the corresponding
```

```
160             %column for every support into the table
161         for n=1:height(Supports)
162            fprintf(Output,'%-8.6g\t[%7.4g, %7.4g]\n',Supports(n,1),...
163                 Supports(n,2), Supports(n,3));
164         end
165         %print the table titel of the Rods to the output text file
166             %Rod Number, Start-Node, End-Node, Direction[x,y], Force
167         fprintf(Output,'\n%s\n','Rods');
168         fprintf(Output,'%10s\t%10s\t%8s\t%14s\n','Rod Number',...
169             'Start-Node','End-Node','Direction[x,y]');
170         %next loop prints the rod informations from the corresponding
171             %column for every rod into the table
172         for n=1:height(Rods)
173           fprintf(Output,'%-11.9g\t%-11.9g\t%-9.7g\t[%7.4g, %7.4g]\n',...
174                 Rods(n,1),Rods(n,2),Rods(n,3),Rods(n,5),Rods(n,6));
175         end
176     end
177 end
178 
179 %function prints the whole linear system of equations of the truss to the
180     %output file
181     %returned is nothing, the coefficient matrix, the external force vector
182     %and force vector is printed to the output file
183 function equotation_system_output(Output,Nodes,Rods,Supports,K,f)
184     %write the Truss Matrix to the output text file
185         %print the title and description
186     fprintf(Output,'\n%s\n%s\n%s\n%s%d\n\n%s',...
187         'Linear system of equations of the truss',...
188         ['One row correspond to one equotation for every Node in '...
189         'x- and y-direction.'],['The system of equations can be solved '...
190         'unambiguously if the determinant of'],...
191         'the coefficient matrix is not equal to zero: det(K)=',det(K),...
192         ' Node  ');
193     %print the titel for each column over the Matrix
194         %for every support print the "Supp." and the support number over the
195         %corresponding column
196     for n=1:height(Supports)
197         fprintf(Output,'%5s%-2d','Supp.', n);
198     end
199     %for every Rod print "Rod" and the Rod number over the corresponding
200         %column
201     for n=1:height(Rods)
202         fprintf(Output,'%4s%-3d','Rod', Rods(n,1));
203     end
204     fprintf(Output,'%9s%5s\n',' Solution  ','Force');
205 
206     %(cutting free every Node) in x- and y-Direction;
207         %for every row in the Matrix
208     for n=1:height(K)
209         %print the Node-number that correspondents to the n'th row in M
210             %at the beginning of a line, ceil() rounds up n/2 to the next
211             %integer, this is the Node-number that corresponds to that row
212         fprintf(Output,'%3d',Nodes(ceil(n/2),1));
```

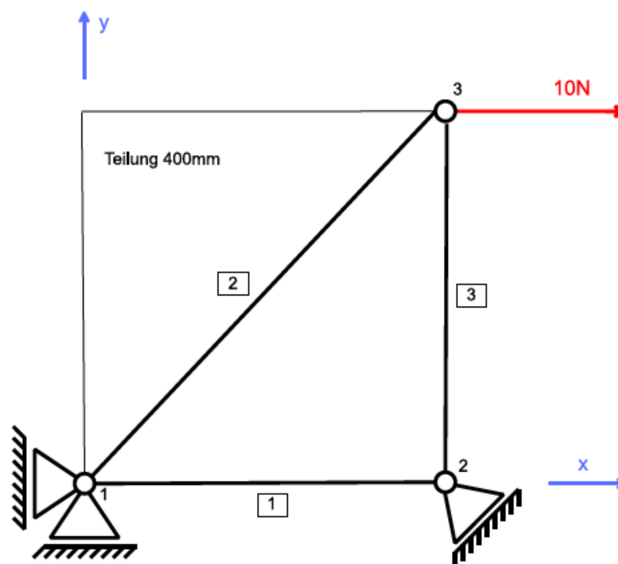
```
213         %print the direction of the Node equotation that correspond to the
214             %n'th row
215             %if the n'th row is a even number the equotation was made in
216             %y-direction, print a "Y"
217             %even number is checked:floor rounds down n/2 to the next
218             %integer if this equals n/2 it is a even number
219         if floor(n/2) == n/2
220             fprintf(Output,'%1s:    ','Y');
221         %else the equotation was made in x-direction, print a "X"
222         else
223             fprintf(Output,'%1s:    ','X');
224         end
225         %print the row in the coefficient Matrix (for every number in the
226             %row n)
227         for m=1:width(K)
228             fprintf(Output,'%-7.3g',K(n,m));
229         end
230         %behind the coefficient Matrix print the rest of the equotation
231             %at the half row print the multiplication sign, solution and the
232             %equotation sign
233         if n==height(K)/2
234             fprintf(Output,'%11s','  *  a  =  ');
235         %if it is not the half row, leave the same amount of fields free
236         else
237             fprintf(Output,'%11s',' ');
238         end
239         %print the force vector right to the equotation sign
240         fprintf(Output,'%-8.8g\n',f(n,1));
241         %fprintf(Output,'\n');
242     end
243 end
```

B Kurzspezifikation des Befehlssatzes

Die Codierung des Befehlssatzes ist an ANSYS-APDL angelehnt:

Befehl	**Bedeutung**
! Kommentar	Eine Zeile Kommentar
N, Knotennummer, x-Koordinate, y-Koordinate	Erzeugt einen Knoten an den angegebenen Koordianten
E,Stabnummer,Anfangsknoten,Endknoten	Erzeugt einen Stab zwischen zwei Knoten
DX,Knotennummer	Sperrt die x-Verschiebung am Knoten auf null
DY,Knotennummer	Sperrt die y-Verschiebung am Knoten
DWinkel,Knotennummer	Sperrt die Verschiebungs-Richtung mit dem Winkel zu vertikalen Achse am Knoten
FX,Knotennummer,Wert	Erzeugt eine Kraft, die in x-Richtung wirkt, an den Knoten
FY,Knotennummer,Wert	Erzeugt eine Kraft, die in y-Richtung wirkt, an den Knoten
SOLVE	Zeigt das Ende des Datensatzes an

```
! Einfaches Demo-Fachwerk
N,1,0,0
N,2,0.4,0
N,3,0.4,0.4
E,1,1,2
E,2,1,3
E,3,2,3
DX,1
DY,1
D45,2
FX,3,10
SOLVE
```



Der Code rechnet dimensionslos.

C Selbstständigkeitserklärung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: ______________________

Vorname: ______________________

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Programmierung eines textbasierten Systems zur Berechnung von ebenen starren und statisch bestimmt gelagerten Fachwerken in MATLAB.

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Ort	Datum	