



LEHRSTUHL UND INSTITUT FÜR LEICHTBAU

Univ.-Prof. Dr.-Ing. H.-G. Reimerdes

– Fakultät für Maschinenwesen –

## STUDIENARBEIT

SA-2012-06

### **ilbFE - Finite Elemente Methode in MATLAB**

Verfasser:

cand. Ing. Arnd Koeppe

Betreuer:

Dipl. Ing. M. Klaus

Aachen, September 2012

**LEHRSTUHL UND INSTITUT FÜR LEICHTBAU**  
**Rheinisch-Westfälische Technische Hochschule Aachen**  
**Universitätsprofessor Dr.-Ing. Hans-G. Reimerdes**

D 52062 Aachen  
Wüllnerstr. 7

## **Studienarbeit**

### **ilbFE - Finite Elemente Methode in MATLAB**

Die Finite Elemente Methode ist ein weit verbreitetes Verfahren zur numerischen Berechnung von ingenieurwissenschaftlichen Problemen.

Besonders in der Strukturberechnung und im Leichtbau bieten verschiedenste kommerzielle Anbieter etablierte Komplettlösungen zur Modellierung, Diskretisierung, Berechnung und Visualisierung von Ergebnissen an.

Die Software MATLAB eignet sich besonders gut zur numerischen Berechnung von linearen Gleichungssystemen in Matrizenform und damit zum Lösen von diskretisierten Finite Elemente Modellen.

Dabei zeichnet sich MATLAB besonders durch seine mathematisch-intuitive Bedienung und umfangreichen Bibliotheksfunktionen aus.

Um dem Studienarbeiter und interessierten Studenten der Vorlesungsreihe Finite Elemente Methode im Leichtbau einen Einblick in die internen Abläufe von kommerziellen FEM Programmen zu geben, soll im Rahmen der Studienarbeit ein exemplarischer FE Code in MATLAB programmiert werden.

Dazu gehört die strukturierte Eingabe des diskretisierten Modells, die Implementation von Elementen und Lösealgorithmen und die Ausgabe der Lösung. Dabei soll die Studienarbeit in Teilen auf dem Funktionspaket CALFEM (Computer Aided Learning of the Finite Element Method) für MATLAB aufbauen.

An Beispielen aus der Vorlesung Finite Elemente Methoden im Leichtbau sollen die Programmabläufe nachvollzogen und die Qualität der Lösungen des programmierten FE Code überprüft werden.

Die Dokumentation des Programms soll auf den Vorlesungsunterlagen aufbauend die grundlegende Theorie der einzelnen Elemente und Programmschritte erklären, sowie eine Anleitung zur Nutzung und eigenständigen Weiterentwicklung durch interessierte Studenten bieten.

# Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Studienarbeit selbständig verfasst habe. Ich versichere, dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Aachen, den 11. September 2012

A handwritten signature in blue ink, appearing to read 'A. Repp', is positioned above a horizontal line.

---

(Unterschrift)

# Abstract

The programm `ilbFE` is a finite element programm based on MATLAB. It offers the user a deep insight in the internals of a FE programm.

The implementation was focused on modular programming so the programm can be expanded with additional elements and solvers. It contains independently developed elements and elements from the finite element package CALFEM.

Additionally, the implemented elements are efficient and produce satisfactory results.

This thesis is divided in a user and a developer guide.

In the user guide the user receives all required informations to execute and understand the process of the FE programm. It contains:

- descriptions of the theory of the finite element method and the programm flow of `ilbFE`.
- a description of the solvers and elements regarding efficiency and limitations.
- a technical documentation of the input and output.

The developer guide adds a description of the internal processes of the programm to the user guide. It offers:

- descriptions of the implemented functions with detailed explanations of input and output parameters.
- instructions to develop the programm further and to add new elements and solvers.

# Kurzzusammenfassung

Das Programm ilbFE ist ein Finite Elemente Programm auf Basis von MATLAB, dass dem Anwender einen tieferen Einblick in die internen Abläufe eines FE-Programmes bietet.

Bei der Implementierung wurde ein besonderer Augenmerk auf einen modularen Programmierstil gelegt, damit das Programm einfach durch weitere Elemente und Löser erweitert werden kann. Exemplarisch wurde dies an selbst entwickelten Elementen und Elementen aus dem Finite Elemente Packet CALFEM gezeigt.

Zusätzlich dazu sind die implementierten Elemente leistungsfähig und ergeben in Tests gute Ergebnisse.

Diese Studienarbeit ist aufgeteilt in ein Benutzer- und ein Entwicklerhandbuch.

Im Benutzerhandbuch werden dem Anwender alle nötigen Informationen zum Ausführen und Verständnis des Ablaufs des FE-Programmes gegeben. Es beinhaltet:

- Beschreibungen der Theorie der Finiten Elemente Methode und des Programmlauf von ilbFE
- Eine Beschreibung der Löser und Elemente in Hinblick auf deren Leistungsfähigkeit und Einschränkungen
- Eine Dokumentation der Eingabe und Ausgabe

Das Entwicklerhandbuch baut auf dem Benutzerhandbuch auf und beschreibt die internen Abläufe des Programms. Es bietet:

- Beschreibungen der implementierten Funktionen mit einer ausführlichen Dokumentation der Eingabe- und Ausgabeparameter.
- Anleitungen zur Weiterentwicklung des Programms und Einbindung von neuen Elementen und Lösern.

# Inhaltsverzeichnis

Abstract	I
Kurzzusammenfassung	II
Abbildungsverzeichnis	VIII
Tabellenverzeichnis	IX
<b>I Benutzerhandbuch</b>	<b>1</b>
1 Einleitung	2
2 Finite Elemente Methode	4
3 Programmablauf	6
4 Kurzbeschreibung der Solver	11
4.1 Lineare Statik Analyse . . . . .	11
4.2 Lineare Modal- und Eigenfrequenzanalyse . . . . .	14
5 Kurzbeschreibung der Elemente	16
5.1 Formulierung der Finite-Elemente-Methode . . . . .	16
5.2 Klassifizierung und Kompatibilität . . . . .	18
5.3 Stabelemente . . . . .	20
5.3.1 2D - 2 Knoten - Stab . . . . .	20
5.3.2 3D - 2 Knoten - Stab . . . . .	21
5.4 Balkenelemente . . . . .	22
5.4.1 2D - 2 Knoten - Balken . . . . .	22
5.4.2 3D - 2 Knoten - Balken . . . . .	23
5.5 Scheibenelemente . . . . .	26
5.5.1 2D - 3 Knoten - Scheibe . . . . .	26

5.5.2	2D - 4 Knoten - Scheibe . . . . .	27
5.5.3	2D - 8 Knoten - Scheibe . . . . .	28
5.6	Schalenelemente . . . . .	29
5.6.1	3D - 4 Knoten - Schale . . . . .	30
5.7	Volumenelemente . . . . .	31
5.7.1	3D - 8 Knoten - Volumen . . . . .	31
<b>6</b>	<b>Eingabe und Ausgabe</b>	<b>32</b>
6.1	Eingabedatei . . . . .	33
6.1.1	Title . . . . .	33
6.1.2	Solver . . . . .	33
6.1.3	Nodes . . . . .	34
6.1.4	Elements . . . . .	35
6.1.5	Materials . . . . .	35
6.1.6	Properties . . . . .	36
6.1.7	BC . . . . .	36
6.1.8	Loads . . . . .	37
6.2	Ausgabedatei . . . . .	38
6.2.1	nDisp . . . . .	38
6.2.2	eStress . . . . .	38
6.2.3	mFreq . . . . .	38
6.2.4	mDisp . . . . .	39
<b>7</b>	<b>Beispiele</b>	<b>40</b>
7.1	Fachwerk aus 2D-Stabelementen . . . . .	40
7.2	Fachwerk aus 3D-Stabelementen . . . . .	42
7.3	Kragbalken aus Balkenelementen in 2D . . . . .	43
7.4	Kragbalken aus Balkenelementen in 3D . . . . .	44
7.5	Kragbalken aus 3-Knoten Scheibenelementen . . . . .	45
7.6	Kragbalken aus 4-Knoten Scheibenelementen . . . . .	46
7.7	Kragbalken aus 8-Knoten Scheibenelementen . . . . .	47
7.8	Kragbalken aus Volumenelementen . . . . .	48
7.9	Kragbalken aus Schalenelementen . . . . .	49
7.10	An zwei Seiten eingespannte, ebene Schale . . . . .	50

<b>II</b>	<b>Entwicklerhandbuch</b>	<b>51</b>
<b>8</b>	<b>Funktionsbeschreibung</b>	<b>52</b>
8.1	Hauptprozedur . . . . .	52
8.1.1	main.m . . . . .	52
8.2	Elemente . . . . .	53
8.2.1	beam2s.m . . . . .	53
8.2.2	beam3s.m . . . . .	54
8.2.3	shelli4e.m . . . . .	56
8.2.4	shelli4s.m . . . . .	57
8.2.5	shelli4d.m . . . . .	58
8.3	Materialgesetze . . . . .	59
8.3.1	constitutiveTrans.m . . . . .	59
8.3.2	iso3DConstitutive.m . . . . .	60
8.3.3	orth3DConstitutive.m . . . . .	61
8.3.4	rotMatrixStrain.m . . . . .	62
8.3.5	rotMatrixStress.m . . . . .	63
8.4	Solver . . . . .	64
8.4.1	solver1.m . . . . .	64
8.4.2	solver2.m . . . . .	65
8.4.3	coordXtr.m . . . . .	66
8.4.4	extractProperties.m . . . . .	67
8.4.5	solveig.m . . . . .	68
8.5	Präprozessor . . . . .	69
8.5.1	preprocess.m . . . . .	69
8.5.2	readInput.m . . . . .	71
8.5.3	prepareInput.m . . . . .	72
8.6	Postprozessor . . . . .	74
8.6.1	postprocess.m . . . . .	74
8.6.2	prepareOutput.m . . . . .	75
8.6.3	writeOutput.m . . . . .	76
8.7	Sonstiges . . . . .	77
8.7.1	checkFileName.m . . . . .	77
8.7.2	ID2Index.m . . . . .	78
8.7.3	index2ID.m . . . . .	79



<b>9</b>	<b>Einbindung Neuer Elemente und Solver</b>	<b>80</b>
9.1	Neue Elemente . . . . .	80
9.1.1	Anpassung des Präprozessors . . . . .	80
9.1.2	Anpassung der Elementeigenschaften-Ausleseprozedur . . . . .	83
9.1.3	Anpassung des Lösert . . . . .	83
9.2	Neue Solver . . . . .	85
<b>10</b>	<b>Zusammenfassung und Fazit</b>	<b>86</b>
	<b>Literaturverzeichnis</b>	<b>88</b>
	<b>Anhang</b>	<b>89</b>
<b>A</b>	<b>Eingebodateien zu den Beispielen</b>	<b>89</b>
A.1	Stab2D.in . . . . .	89
A.2	Stab3D.in . . . . .	90
A.3	Balken2D.in . . . . .	91
A.4	Balken3D.in . . . . .	92
A.5	Scheibe3K2D.in . . . . .	93
A.6	Scheibe4K2D.in . . . . .	95
A.7	Scheibe8K2D.in . . . . .	97
A.8	Solid3D.in . . . . .	99
A.9	Schale3D.in . . . . .	101
A.10	EFSchale3D.in . . . . .	103
<b>B</b>	<b>MATLAB - Hilfe</b>	<b>109</b>
<b>C</b>	<b>CALFEM - A Finite Element Toolbox</b>	<b>117</b>

# Abbildungsverzeichnis

3.1	Programmablaufplan von <code>main.m</code> . . . . .	6
3.2	Beispiel einer Eingabedatei . . . . .	7
3.3	Programmablaufplan der Vorlaufrechnung von <code>solver1.m</code> . . . . .	8
3.4	Programmablaufplan der Hauptrechnung von <code>solver1.m</code> . . . . .	9
3.5	Programmablaufplan der Nachlaufrechnung von <code>solver1.m</code> . . . . .	10
5.1	Aufbau der Element-ID . . . . .	18
5.2	Ein Stab . . . . .	20
5.3	Ein Balken . . . . .	22
5.4	Eine Scheibe . . . . .	26
5.5	Isoparametrische Formulierung einer Scheibe . . . . .	27
6.1	Beispiel-Header mit 3 Parametern . . . . .	32
6.2	Beispiel-Karte für <code>Title</code> . . . . .	33
6.3	Beispiel-Karte für <code>Solver</code> für Lineare Statik Analyse . . . . .	34
6.4	Beispiel-Karte für 3 <code>Nodes</code> in 2D . . . . .	34
6.5	Beispiel-Karte für 2 <code>Nodes</code> in 3D . . . . .	35
6.6	Beispiel-Karte für <code>Elements</code> mit 3 Stäben . . . . .	35
6.7	Beispiel-Karte für <code>Elements</code> mit Schale und Stab . . . . .	35
6.8	Beispiel-Karte für <code>Materials</code> bei einer dynamischen Analyse einer Schale . . . . .	36
6.9	Beispiel-Karte für <code>Properties</code> eines Stabes und eines 3D-Balkens . . . . .	36
6.10	Beispiel-Karte für <code>BC</code> bei 2D-Kontinuumselementen . . . . .	37
6.11	Beispiel-Karte für <code>BC</code> bei 3D-Strukturelemente . . . . .	37
6.12	Beispiel-Karte für <code>Loads</code> bei 2D-Kontinuumselementen . . . . .	37
6.13	Beispiel-Karte für <code>Loads</code> bei 3D-Strukturelementen . . . . .	37
6.14	Beispiel-Karte für <code>nDisp</code> bei 3D-Strukturelementen . . . . .	38
6.15	Beispiel-Karte für <code>eStress</code> bei 3D-Strukturelementen . . . . .	38
6.16	Beispiel-Karte für <code>mFreq</code> mit den 3 niedrigsten Eigenfrequenzen . . . . .	39

6.17	Beispiel-Karte für <code>mDisp</code> mit den normierten Verschiebungen von 3 Knoten bei der ersten Eigenschwingung . . . . .	39
7.1	2D-Fachwerk . . . . .	41
7.2	3D-Fachwerk . . . . .	42
7.3	Kragbalken aus 10 Balkenelementen in 2D . . . . .	43
7.4	Kragbalken aus 10 Balkenelementen in 3D . . . . .	44
7.5	Kragbalken aus 20 3-Knoten-Scheibenelementen . . . . .	45
7.6	Kragbalken aus 10 4-Knoten-Scheibenelementen . . . . .	46
7.7	Kragbalken aus 10 8-Knoten-Scheibenelementen . . . . .	47
7.8	Kragbalken aus 10 Volumenelementen . . . . .	48
7.9	Kragbalken aus 10 Schalenelementen . . . . .	49
7.10	An zwei Seiten gelagerte, ebene Schale . . . . .	50
9.1	Setzen der Freiheitsgrade im Präprozessor . . . . .	81
9.2	Setzen der Knotenzahl im Präprozessor . . . . .	81
9.3	Setzen der Element-Topologie im Präprozessor . . . . .	82
9.4	Setzen der Element-Eigenschaften im Präprozessor . . . . .	82
9.5	Anpassung der Elementeigenschaften-Ausleseprozedur . . . . .	83
9.6	Anpassung der Vorlaufrechnung im Löser . . . . .	83
9.7	Anpassung der Nachlaufrechnung im Löser . . . . .	84
9.8	Anpassung der Hauptprozedur . . . . .	85

# Tabellenverzeichnis

5.1	Übersicht über die Elemente . . . . .	19
6.1	Solver in ilbFE . . . . .	34
6.2	Übersicht der Elemente in ilbFE und deren <b>Properties</b> . . . . .	36

---

# Teil I

## Benutzerhandbuch

# 1. Einleitung

Die Finite Elemente Methode ist ein weit verbreitetes Verfahren zur numerischen Berechnung von ingenieurwissenschaftlichen Problemen.

Besonders in der Strukturberechnung und im Leichtbau bieten verschiedenste kommerzielle Anbieter etablierte Komplettlösungen zur Modellierung, Diskretisierung, Berechnung und Visualisierung von Ergebnissen an.

Zur kompetenten Nutzung von FE-Programmen ist es für den Anwender erforderlich nicht nur die Bedienoberfläche eines oder mehrerer kommerzieller Programme zu kennen, sondern auch die grundlegenden Hintergründe der Finite Elemente Methode zu verstehen.

Besonders in Hinsicht auf die Auswahl von Elementen für verschiedene Problemstellungen, bezüglich Rechenaufwand und Leistungsfähigkeit, aber auch die Wahl geeigneter Lösungsmethoden, stellt die Finite Elemente Analyse eine große Herausforderung dar.

Mit der steigenden Rechenleistung bei Computern und der stetig fortschreitenden Entwicklung bei Präprozessor-Programmen für die FE-Rechnung (z.B. ABAQUS/CAE) werden dem Ingenieur immer mehr Werkzeuge zur einfachen, intuitiven Modellierung angeboten und Fehler werden eventuell gar nicht erst zugelassen.

Allerdings wird der Nutzer dabei stark eingeschränkt oder merkt unter Umständen gar nicht, dass eine Rechnung oder ihre Ergebnisse fehlerhaft sind. Bei vielen Freiheitsgraden brauchen selbst heutige Rechner noch zu lange, als dass eine Blockierung mehrerer Prozessoren durch eine fehlerhafte Rechnung mit hinterher wertlosen Ergebnissen ökonomisch tolerierbar ist.

Wird den falschen Ergebnissen dann auch noch blind vertraut, dann kann eine einzige falsche Eingabe katastrophale Folgen nach sich ziehen.

Darum ist es unerlässlich, dass angehende Berechnungsingenieure die Grundlagen der Finite Elemente Methode nicht nur theoretisch Verstehen, sondern auch die Brücke zwischen Theorie und Anwendung erfassen können.

Die Software MATLAB eignet sich besonders gut zur numerischen Berechnung von linearen Gleichungssystemen in Matrizenform und damit zum Lösen von diskretisierten Finite Elemente Modellen.

Dabei zeichnet sich MATLAB besonders durch seine mathematisch-intuitive Bedienung und umfangreichen Bibliotheksfunktionen aus.

Um interessierten Studenten der Vorlesungsreihe *Finite Elemente Methode im Leichtbau* [Rei+09] einen Einblick in die internen Abläufe von FE-Programmen zu geben, wurde im Rahmen der Studienarbeit ein exemplarischer FE-Code in MATLAB programmiert.

Dazu gehört die strukturierte Eingabe des diskretisierten Modells, die Implementation von Elementen und Lösealgorithmen und die Ausgabe der Lösung. Dabei baut die Studienarbeit in Teilen auf dem Funktionspaket CALFEM [Aus+04] für MATLAB auf.

Diese Studienarbeit, aufgeteilt in Benutzerhandbuch ([Teil I](#)) und Entwicklerhandbuch ([Teil II](#)), beschreibt, aufbauend auf die Vorlesungsunterlagen [Rei+09], die grundlegende Theorie der einzelnen Elemente und Programmschritte.

Im Benutzerhandbuch gibt [Kapitel 2](#) eine kurze, allgemeine Einführung in die Finite Elemente Methode.

[Kapitel 3](#) erläutert den Programmablauf von ilbFE anhand von Programmablaufplänen Eingabe- und Ausgabedatei.

Es folgt in [Kapitel 4](#) und [Kapitel 5](#) entsprechend eine Beschreibung der implementierten Löseverfahren und Elemente. Hier wurde ein besonderer Augenmerk auf das für Leichtbaustrukturen besonders relevante Schalenelement ([Abschnitt 5.6](#)) gelegt.

Zum Schluss des Benutzerhandbuches werden in [Kapitel 6](#) die Eingabe- und Ausgabedateien genauer erklärt. Mit Hilfe von Beispielen wird jede implementierte Karte erläutert.

Weitere Beispiele werden in [Kapitel 7](#) erläutert.

Das Entwicklerhandbuch ([Teil II](#)) dient als Referenz zur Weiterentwicklung von ilbFE.

In [Kapitel 8](#) werden die für ilbFE entwickelten Funktionen, inklusive der Eingabe- und Ausgabeparameter, beschrieben.

Darauf folgt in [Kapitel 9](#) eine Anleitung zur Einbindung neuer Elemente und Löser. Durch die Modularität von ilbFE, insbesondere des Prä- und Postprozessors, wurde die Einbindung möglichst unkompliziert gestaltet.

## 2. Finite Elemente Methode

Die Finite Elemente Methode (FEM) ist ein weit verbreitetes Verfahren zur numerischen Berechnung von ingenieurwissenschaftlichen Problemen.

Besonders in der Strukturberechnung und im Leichtbau bieten verschiedenste kommerzielle Anbieter etablierte Komplettlösungen zur Modellierung, Diskretisierung, Berechnung und Visualisierung von Ergebnissen an.

Allen Finite-Elemente-Programmen (FE-Programmen) ist gemein, dass eine Struktur in verschiedene Elemente aufgeteilt wird, die über Knoten miteinander verbunden sind.

Knoten und Elemente bilden zusammen ein Netz, welches die tatsächliche Struktur eines Bauteils näherungsweise beschreibt.

Jeder Knoten besitzt eine eindeutige ID und eine Position im globalen Koordinatensystem.

Bei Belastung kann dieser Verschiebungen, Verdrehungen und Zustandsänderungen entsprechend der Freiheitsgrade der angeschlossenen Elemente und aufgeprägten Randbedingungen erfahren.

Elemente besitzen, ähnlich wie Knoten, eine eindeutige ID, aber ihre Größe und Position ist über die Zuordnung von Knoten zu jedem Element beschrieben.

Hierbei können und müssen Knoten zu mehreren Elementen gehören, damit diese korrekt verbunden sind und das Modell der Struktur keine unzulässigen Lücken aufweist, wodurch Knotenbewegungen zugelassen würden, die bei der realen Struktur nicht möglich wären.

Zusätzlich zu den Knoten werden jedem Element je nach Typ verschiedene Attribute und Eigenschaften zugeordnet.

Man unterscheidet zwischen Materialeigenschaften, wie zum Beispiel der Elastizität, Dichte und Plastizität, und Geometrieeigenschaften, wie zum Beispiel der Dicke oder Fläche.

Zu beachten ist hierbei, dass Geometrieeigenschaften von Elementen reine Attribute sind, die auf Grund von Vereinfachungen und kinematischen Annahmen bei der Formulierung der Elemente entstehen. Sie sind in der Geometrie des Modells nicht erkennbar.

So hat zum Beispiel in Scheibenelement die Dicke (3. Dimension) als Geometrieeigenschaft, während die Höhe und Breite über die Knotenpositionen bestimmt sind. Im Kontrast dazu hat ein Stabelement die Querschnittsfläche (2. und 3. Dimension) als Geometrieeigenschaft, während nur die Länge über die Geometrie der Knoten dargestellt wird.



Bei der Wahl der Elemente kommt es auf die Problemstellung, vorhandene Rechenkapazitäten und zulässige Annahmen zur Vereinfachung an.

Mit Volumenelementen zum Beispiel kann eine dreidimensionale Struktur fast exakt und ohne zusätzliche Geometrieeigenschaften beschrieben werden und auch Scheuerungen und Verdrehung der Elemente sind ohne Verdrehungsfreiheitsgrade in den Knoten eindeutig.

Allerdings wird der Rechenaufwand durch die große Anzahl von Knoten pro Element und der großen Anzahl an Elementen die zur geometrischen Abbildung des Modells benötigt werden sehr hoch.

Zusätzlich können Volumenelemente erhebliche numerische Fehler (Sperren) erzeugen, sodass selbst ein feines Netz noch keine genaue Lösung garantiert. Darum werden dünnwandige Strukturen oft mit Schalenelementen idealisiert. Diese Elemente besitzen im Gegensatz zu Volumenelementen die Dicke als Geometrieeigenschaft, sodass in Dickenrichtung keine Knoten nötig sind.

Der durch die Knoteneinsparung deutlich niedrigere Rechenaufwand wird allerdings durch zusätzliche Verdrehungsfreiheitsgrade erkauft, da ohne die zusätzlichen Freiheitsgrade der Zustand der Struktur an jedem Knoten nicht eindeutig beschrieben werden kann.

Bei der Analyse von dünnwandigen Strukturen bieten Schalenelemente somit den Vorteil, dass in Dickenrichtung keine zusätzlichen Elemente benötigt werden, um die Struktur ausreichend anzunähern.

Für dickwandige Strukturen sind Volumenelemente allerdings die bessere Wahl, da die Verformungen und Spannungen in Dickenrichtung mit Volumenelementen dargestellt werden können.

Auch Schalenelemente haben (wenn auch weniger ausgeprägt) Probleme mit Schubsperrern, allerdings kann das Problem über Rechenverfahren (Reduzierte Integration) oder zusätzliche Knoten verhindert werden, sodass die numerischen Ergebnisse recht gut gegen die analytischen Werte konvergieren, zumal bereits mit gröberen Netzen eine zufriedenstellende Genauigkeit erreicht werden kann.

Bei symmetrischen Problemen, Belastung nur in einer Richtung, Belastung in einer Ebene oder makroskopischer Betrachtung könnten noch einfachere Elemente mit weniger Knoten, Freiheitsgraden und Rechenaufwand verwendet werden.

Eine genaue Beschreibung der Elemente mit deren Annahmen und Leistungsfähigkeit ist in [Kapitel 5](#) gegeben.

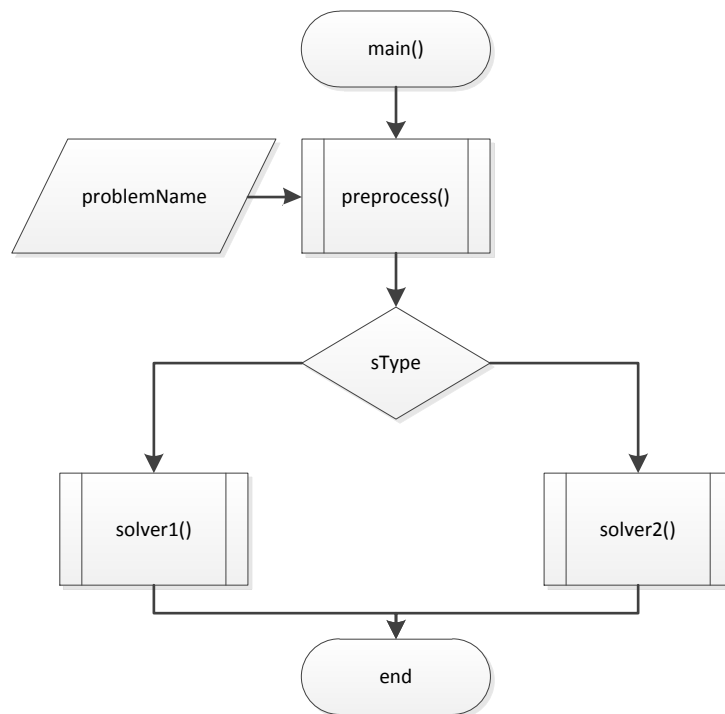
Da die Rechenzeit und Genauigkeit des Ergebnisses von der Wahl der Elemente, sowie der Anzahl und Positionierung der Knoten abhängt, ist die Erstellung eines FE-Modells nicht trivial und erfordert einen erfahrenen Anwender, der mit den Hintergründen der FEM vertraut ist.

## 3. Programmablauf

Der Programmablauf von ilbFE orientiert sich am Ablauf vorgestellt in der Vorlesung Finite Berechnungsmethoden im Leichtbau.

Die meisten kommerziellen FE-Programme folgen in verschiedenen Ausprägungen diesem Programmablauf, allerdings werden durch optimierte Algorithmen und zusätzliche Zwischenschritte die Rechnung beschleunigt und das Ergebnis verbessert.

Die Finite Elemente Rechnung wird mit dem Befehl `main('eingabe.in')` gestartet, wobei `'eingabe.in'` der Name einer Eingabedatei ist.



**Abbildung 3.1:** Programmablaufplan von `main.m`

Zu Beginn von ilbFE ([Abbildung 3.1](#)) steht der Präprozessor (`preprocess()`). Der Präprozessor hat die Aufgabe Datensätze aus einer Eingabedatei (Endung: `.in`) einzulesen und für die eigentliche FE-Rechnung aufzubereiten.

In der Eingabedatei ([Abbildung 3.2](#)) sind die verschiedenen Knoten, Elemente, Materialeigenschaften, Lasten, Randbedingungen und so weiter in sogenannten Karten hinterlegt.

Die einzelnen Karten besitzen verschiedene Spalten, denen über eine Header-Zeile Parameternamen zugeordnet werden, sodass die Spalten der Datensätze eindeutig einer Bedeutung zugeordnet werden kann.

Title This is a Test!										
H Solver	Type	Steps	Error							
C Type = 1	Linear	Static	Analysis							
C Type = 2	Eigenfrequency	and Modal	Analysis							
Solver	1	10	1e-12							
H Nodes	ID	X	Y							
Nodes	1	0	0							
Nodes	2	1000	0							
Nodes	3	500	1000							
H Elements	ID	Type	MatID	PropID	N1	N2				
Elements	1	122	1	1	1	2				
Elements	2	122	1	1	2	3				
Elements	3	122	1	1	3	1				
H Materials	ID	Ep	Es	nue	Gq	phi	rho	a	b	
Materials	1	2.1e5	2.1e5	0.3	1.05e5	0	1	0.1	0.05	
H Properties	ID	A	I							
Properties	1	100	10000							
H BC	NodeID	XDir	YDir							
C anyDir = 0		fixed								
C anyDir = i		free								
BC	1	0	0							
BC	2	i	0							
H Loads	NodeID	ForceX	ForceY							
Loads	2	0	0							
Loads	3	1000	1000							

**Abbildung 3.2:** Beispiel einer Eingabedatei

Das Erstellen der Eingabedatei erfolgt entweder per Hand durch den Anwender oder über zusätzliche Präprozessor-Programme, die dem Anwender den großen Aufwand zum Erstellen von Karten bei feinen Netzen (große Anzahl von Datensätzen für Knoten und Elemente) oder komplexen Strukturen (komplizierte Modellierung) abnehmen. Zusätzlich enthalten kommerzielle FE-Präprozessor-Programme Elemente aus dem Computer Aided Engineering (CAE), wie graphischer Modellierung und parameterbasierender Konstruktion.

In ilbFE erfolgt die Eingabe von Strukturen manuell durch den Nutzer. Um eine möglichst große Modularität zu gewährleisten kann der Anwender die in Karten enthaltenen Informationen und Parameter selbst definieren. Auch das definieren von neuen Karten ist möglich.

Falls also zum Beispiel eine weitere Eingangsgröße oder sogar eine neue Klasse zur Implementierung von einem neuen Element oder Löser nötig ist, muss der Anwender nur die Prozedur zur Datenaufbereitung modifizieren, die Einlese-Prozedur braucht

nicht angepasst zu werden.

Auf den Präprozessor folgt je nach geforderter Analyseart das Kernprogramm. Dazu wird in ilbFE nach dem Eingabeparameter `sType` zwischen Linearen Statik Analyse (`solver1()`) und der Eigenfrequenzanalyse (`solver2()`) unterschieden.

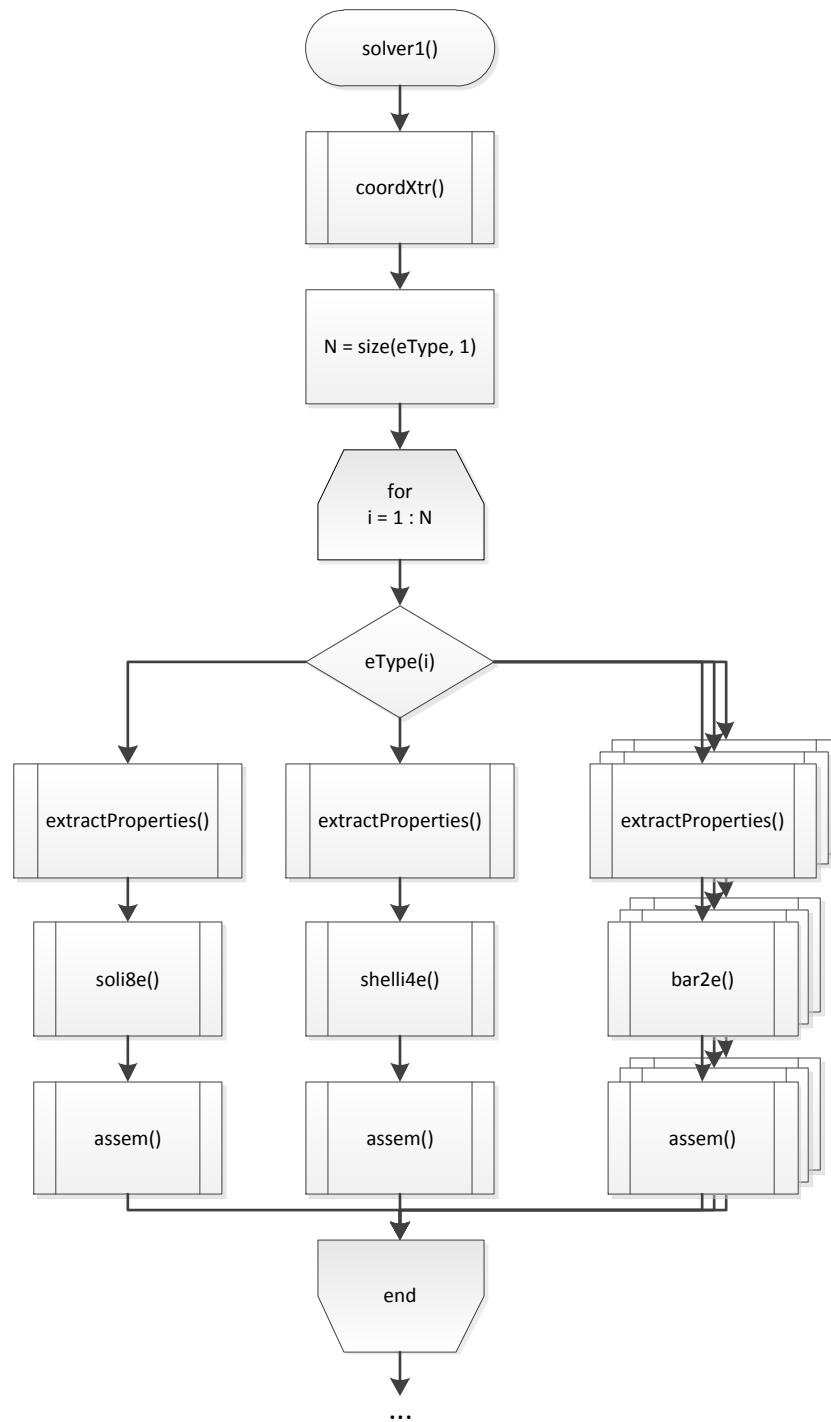
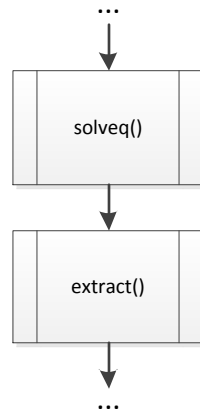


Abbildung 3.3: Programmablaufplan der Vorlaufrechnung von `solver1.m`

Bei einer Linearen Statik Analyse ([Abbildung 3.3](#)) werden zunächst die Koordinaten extrahiert (`coordXtr()`).

Anschließend werden über eine Zählschleife über alle Elemente für jedes Element je nach Elementtyp zunächst die Elementeigenschaften aufbereitet (`extractProperties()`), die globale Elementsteifigkeitsmatrix berechnet (`shell14e()`, `solid8e()`, `bar2e()`, ...) und in die globale Gesamtsteifigkeitsmatrix einsortiert (`assem()`).



**Abbildung 3.4:** Programmablaufplan der Hauptrechnung von solver1.m

Nachdem die globale Gesamtsteifigkeitsmatrix bestimmt wurde, kommt der eigentliche Löser zum Einsatz ([Abbildung 3.4](#)). Die Steifigkeitsmatrix, Randbedingungen und Lasten werden an den Löser übergeben (`solveq()`), der das Problem nach (3.1) löst.

$$\vec{F} = \mathbf{K}\vec{U} \quad (3.1)$$

In einer Nachlaufrechnung ([Abbildung 3.5](#)) werden anschließend aus dem globalen Verformungsvektor  $\vec{U}$  die Verformungen der einzelnen Elemente extrahiert.

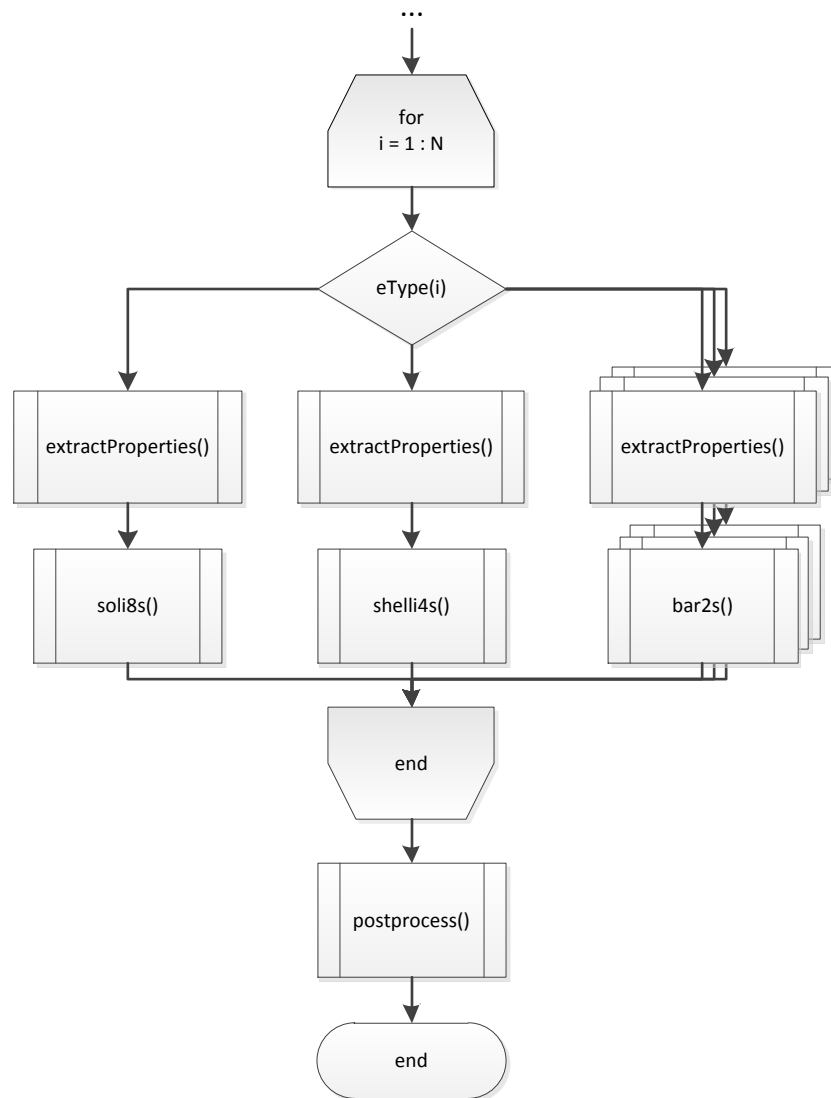
Hieraus berechnet `ilbFE` dann die Spannungen an den Integrationspunkten im Element (`shell14s()`, `solid8s()`, `bar2s()`, ...).

Die Eigenfrequenz- und Modalanalyse läuft analog zur Linearen Statik Analyse ab. Neben der Elementsteifigkeitsmatrizen werden allerdings noch Elementmassenmatrizen von den Elementen (`shell14d()`, `beam2d()`) berechnet.

Diese werden dann über die Funktion `assem()` zur globalen Gesamtsteifigkeits- und Gesamtmassenmatrix zusammengesetzt.

Die Ergebnisse der FE-Rechnung werden zum Schluss an den Postprozessor (`postprocess()`) übergeben.

In dieser Prozedur werden die Ergebnisse gefiltert, sortiert und aufbereitet, um anschließend in die Ausgabedatei geschrieben zu werden.



**Abbildung 3.5:** Programmablaufplan der Nachlaufrechnung von solver1.m

Diese Ausgabedatei (Endung: .out) wird im Anschluss an die Rechnung im graphischen Postprozessor visualisiert. Um eine bereits berechnete Ausgabedatei zu öffnen, wird der Befel `openOutputGUI('ausgabe.out')` verwendet, wobei 'ausgabe.out' der Name der Ausgabedatei ist.

## 4. Kurzbeschreibung der Solver

### 4.1 Lineare Statik Analyse

Der Lineare Statik Solver dient der Lösung linearer Probleme. Nichtlinearitäten im Material, große Verformungen oder Veränderungen an den Randbedingungen werden nicht berücksichtigt.

Die Lasten werden als konstant und die Verformungen als klein im Vergleich zu den Strukturabmaßen angenommen.

Nachdem die benötigten Elementeigenschaften ermittelt wurden, berechnet der Lineare Statik Solver die Elementsteifigkeitsmatrizen  $\mathbf{k}_e$  und ordnet diese in die globale Steifigkeitsmatrix  $\mathbf{K}$  ein. Die globalen und Elementsteifigkeitsmatrizen sind quadratisch und symmetrisch, wobei die Dimensionen und Matrizeneinträge entsprechend ausgewählten Elementen variieren.

Die globale Steifigkeitsmatrix  $\mathbf{K}$  wird nun, zusammen mit den äußeren Lasten und Randbedingungen, an den eigentlichen Löser übergeben.

Sollten keine Randbedingungen (Fesselungen oder vorgegebene Verschiebungen) vorhanden sein versucht der Löser die Grundgleichung (4.1) mit Gauss-Elimination zu lösen.

$$\vec{F} = \mathbf{K}\vec{U} \quad (4.1)$$

Dabei ist zu beachten, dass Steifigkeitsmatrizen ohne Einführung von Randbedingungen immer singulär und demnach nicht invertierbar sind. Das System ist unbestimmt, wodurch die Rechnung abbricht.

Bei vorhandenen Randbedingungen werden zunächst die zugehörigen Zeilen und Spalten (hier:  $i$  und  $j$ ) in der Steifigkeitsmatrix  $\mathbf{K}$  beziehungsweise dem Kraft- ( $\vec{F}$ ) und Verschiebungsvektor ( $\vec{U}$ ) gestrichen.

$$\begin{pmatrix} F_1 \\ \vdots \\ F_i \\ \vdots \\ F_j \\ \vdots \\ F_n \end{pmatrix} = \begin{pmatrix} K_{11} & \cdots & K_{1i} & \cdots & K_{1j} & \cdots & K_{1n} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ K_{i1} & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & K_{in} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ K_{j1} & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & K_{jn} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ K_{n1} & \cdots & K_{ni} & \cdots & K_{nj} & \cdots & K_{nn} \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_i \\ \vdots \\ U_j \\ \vdots \\ U_n \end{pmatrix} \quad (4.2)$$

Damit wird das Problem auf (4.3) reduziert.

$$\vec{F}_{red} = \mathbf{K}_{red} \vec{U}_{red} \quad (4.3)$$

Durch von Null verschiedene Verschiebungs-Randbedingungen entstehen zusätzliche innere Kräfte, die vor der Gauss-Elimination zum reduzierten Kraftvektor addiert werden müssen.

$$\vec{F}^* = \vec{F}_{red} + \begin{pmatrix} K_{i1} & K_{j1} \\ \vdots & \vdots \\ K_{ii-1} & K_{ji-1} \\ K_{ii+1} & K_{ji+1} \\ \vdots & \vdots \\ K_{ij-1} & K_{jj-1} \\ K_{ij+1} & K_{jj+1} \\ \vdots & \vdots \\ K_{in} & K_{jn} \end{pmatrix} \begin{pmatrix} U_i \\ U_j \end{pmatrix} \quad (4.4)$$

Das resultierende Lineare Gleichungssystem (4.5) ist quadratisch und lässt sich somit mit Gauss-Elimination lösen.

$$\vec{F}^* = \mathbf{K}_{red} \vec{U}_{red} \quad (4.5)$$

Nachdem die Verschiebungen aller freien Knoten  $\vec{U}_{red}$  berechnet wurden, können diese mit den bekannten Verschiebungen der gefesselten Knoten ( $U_i, U_j$ ) kombiniert werden.

Wird der nun bekannte Verschiebungsvektor  $\vec{U}$  mit den vollständigen Zeilen der Steifigkeitsmatrix multipliziert, die wegen der Randbedingungen in  $\mathbf{K}_{red}$  gestrichen wurden (hier:  $i$  und  $j$ ), folgen die unbekannten Reaktionskräfte an der Einspannung  $\vec{F}_1$ .

$$\underbrace{\begin{pmatrix} F_i \\ F_j \end{pmatrix}}_{\vec{F}_1} = \underbrace{\begin{pmatrix} K_{i1} & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & K_{in} \\ K_{j1} & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & K_{jn} \end{pmatrix}}_{\mathbf{K}_1} \underbrace{\begin{pmatrix} U_1 \\ \vdots \\ U_i \\ \vdots \\ U_j \\ \vdots \\ U_n \end{pmatrix}}_{\vec{U}} \quad (4.6)$$



Aus dem Vektor der gesamten Knoten-Verschiebungen  $\vec{U}$  extrahiert der Lineare Statik Löser zu jedem Element die Verschiebungen der zugehörigen Knoten aus denen wiederum die entsprechenden Elementspannungen an den Integrationspunkten der Elemente berechnet werden.

Standardmäßig werden zusätzlich zu den Eingabeinformationen - insbesondere der Strukturtopologie - die Knotenverschiebungen und die Elementspannungen zur Ausgabe übergeben.

## 4.2 Lineare Modal- und Eigenfrequenzanalyse

Freie, ungedämpfte Schwingungen folgen der Differentialgleichung (4.7).

$$\mathbf{M}\ddot{\vec{U}} + \mathbf{K}\vec{U} = \vec{0} \quad (4.7)$$

Zum Lösen der Differentialgleichung wird eine Ansatzfunktion nach (4.8) eingeführt und zweimal abgeleitet.

$$\vec{U} = \vec{U}_i \cos(\omega t) \quad (4.8)$$

$$\dot{\vec{U}} = -\omega \vec{U}_i \sin(\omega t) \quad (4.9)$$

$$\ddot{\vec{U}} = -\omega^2 \vec{U}_i \cos(\omega t) \quad (4.10)$$

Setzt man (4.8) und (4.10) in (4.7) ein, erhält man für alle  $\cos(\omega t) \neq 0$  das Eigenwertproblem (4.11) für ungedämpfte Eigenschwingungen von Systemen aus finiten Elementen.

$$(\mathbf{K} - \omega^2 \mathbf{M}) \vec{U}_i = \vec{0} \quad (4.11)$$

Bei Linearer Modal und Eigenfrequenzanalyse muss (4.11) für alle Eigenvektoren  $\vec{U}_i$  erfüllt sein. Die Eigenvektoren  $\vec{U}_i$  stellen hierbei die normierten Knotenverschiebungen der Gesamtstruktur dar.

Der Löser für die Lineare Modal- und Eigenfrequenzanalyse bestimmt die Steifigkeitsmatrix  $\mathbf{K}$  analog zum Linearen Statik Löser (Abschnitt 4.1). Zusätzlich wird allerdings noch analog zur Steifigkeitsmatrix die Massenmatrix  $\mathbf{M}$  berechnet.

Auch die Einführung der Randbedingungen für die Steifigkeitsmatrix und Massenmatrix geschieht analog, indem Zeilen- und Spalten (hier:  $i$  und  $j$ ) gestrichen werden.

$$\mathbf{K}_{red} = \begin{pmatrix} K_{11} & \cdots & K_{1i} & \cdots & K_{1j} & \cdots & K_{1n} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ K_{i1} & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & K_{in} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ K_{j1} & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & K_{jn} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ K_{n1} & \cdots & K_{ni} & \cdots & K_{nj} & \cdots & K_{nn} \end{pmatrix} \quad (4.12)$$

$$\mathbf{M}_{red} = \begin{pmatrix} M_{11} & \cdots & M_{1i} & \cdots & M_{1j} & \cdots & M_{1n} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ M_{i1} & \cdots & M_{ii} & \cdots & M_{ij} & \cdots & M_{in} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ M_{j1} & \cdots & M_{ji} & \cdots & M_{jj} & \cdots & M_{jn} \\ \vdots & \ddots & & \ddots & & \ddots & \vdots \\ M_{n1} & \cdots & M_{ni} & \cdots & M_{nj} & \cdots & M_{nn} \end{pmatrix} \quad (4.13)$$

Zu beachten ist hierbei, dass Versteifungen der Struktur durch Vorspannungen nicht berücksichtigt werden. Verschiebungs- und Verdrehungs-Randbedingungen können deshalb nur Null oder frei sein. Vorgegebene von Null verschiedene Randbedingungen werden ignoriert.

Sind die reduzierten Steifigkeits- und Massenmatrizen  $(\mathbf{K}_{red}, \mathbf{M}_{red})$  bekannt, kann (4.11) angepasst und das Eigenwertproblem kann gelöst werden.

$$(\mathbf{K}_{red} - \omega^2 \mathbf{M}_{red}) \vec{U}_{red} = \vec{0} \quad (4.14)$$

Die gefesselten und vernachlässigten Knotenverschiebungen ergeben zusammen mit den berechneten Knotenverschiebungen  $\vec{U}_{red,i}$  für jeden Mode die normierten Knotenverschiebungen des Gesamtsystems  $\vec{U}_i$ .

Jeder Eigenschwingung mit der Frequenz  $f_i$  (siehe (4.15)) entspricht einer normierten Knotenverschiebung  $\vec{U}_i$ .

$$f_i = \frac{\omega_i}{2\pi} \quad (4.15)$$

Die Frequenzen der Eigenschwingung und die Knotenverschiebung der berechneten Moden werden an die Ausgabeprozedur übergeben.

## 5. Kurzbeschreibung der Elemente

### 5.1 Formulierung der Finite-Elemente-Methode

Nach [Bat02] wurden die in ilbFE implementierten Elemente nach dem Prinzip der virtuellen Verschiebungen formuliert.

Demnach befindet sich eine Struktur im Gleichgewicht, wenn innere und äußere virtuelle Arbeit gleich groß sind.

$$\delta W_a = \delta W_i \quad (5.1)$$

Die innere virtuelle Arbeit ist definiert als das Produkt der realen Spannungen und der virtuellen Verzerrungen integriert über dem Körpervolumen  $V$  nach (5.2).

$$\delta W_i = \int_V \vec{\sigma}^T \delta \vec{\varepsilon} dV \quad (5.2)$$

Die äußere virtuelle Arbeit ergibt sich aus dem Produkt der realen äußeren Kräfte mit den zugehörigen virtuellen Verschiebungen nach (5.3).

$$\delta W_a = \vec{F}^T \delta \vec{U} + \int_O \vec{p}_O^T \delta \vec{u}_O dO + \int_V \vec{p}^T \delta \vec{u} dV \quad (5.3)$$

Für eine Finite-Elemente-Berechnung wird die Struktur als eine Gruppierung diskreter finiter Elemente angenähert, die an den Knotenpunkten auf den Elementgrenzen miteinander verbunden sind.

Mit dem Prinzip der virtuellen Verschiebungen nach (5.1) ergibt sich die Energiegleichung (5.4) für ein Element  $e$ .

$$\int_V \delta \vec{\varepsilon}^T \vec{\sigma} dV = \vec{F}_e^T \delta \vec{U}_e + \int_O \vec{p}_O^T \delta \vec{u}_O dO + \int_V \vec{p}^T \delta \vec{u} dV \quad (5.4)$$

Dabei sind  $\vec{F}_e$ ,  $\vec{p}_O$  und  $\vec{p}$  die äußeren Diskreten-, Oberflächen- und Volumenkräfte und  $\vec{U}_e$ ,  $\vec{u}_O$  und  $\vec{u}$  die korrespondierenden Verschiebungen.

Die Verschiebungen innerhalb des Elementes und auf den Elementgrenzen werden nach (5.5) und (5.6) als Funktionen der Verschiebungen in den Knotenpunkten beschrieben.

$$\vec{u}(x, y, z) = \mathbf{N}(x, y, z) \vec{U}_e \quad (5.5)$$

$$\vec{u}_O(x_O, y_O, z_O) = \mathbf{N}_O(x_O, y_O, z_O) \vec{U}_e \quad (5.6)$$

$\mathbf{N}(x, y, z)$  ist die Interpolationsmatrix für die Verschiebungen im Element.  
 $\mathbf{N}_O(x_O, y_O, z_O)$  erhält man aus  $\mathbf{N}(x, y, z)$ , in dem die passenden Element-Oberflächenkoordinaten eingesetzt werden.

Die Verzerrungs-Verschiebungs-Beziehung des Elements lässt sich schreiben als (5.7).

$$\vec{\varepsilon} = \mathbf{D}\vec{u} \quad (5.7)$$

Dabei ist  $\mathbf{D}$  eine geeignete Differential-Operatormatrix (z.B. für ein Stabelement:  $\mathbf{D} = \frac{\partial}{\partial x}$ ).

Mit der Verschiebungsannahme ergibt sich für die Verzerrungen die Gleichung (5.8).

$$\vec{\varepsilon} = \underbrace{\mathbf{D}\mathbf{N}}_{\mathbf{B}} \vec{U}_e \quad (5.8)$$

$\mathbf{B} = \mathbf{D}\mathbf{N}$  ist die Verzerrungs-Verschiebungs-Matrix des Elements.

Die Spannungen im Element  $\vec{\sigma}$  sind mit den Element-Verzerrungen  $\vec{\varepsilon}$  über die Beziehung (5.9) verknüpft.

$$\vec{\sigma} = \mathbf{E}\vec{\varepsilon} \quad (5.9)$$

Dabei ist  $\mathbf{E}$  die Elastizitätsmatrix des Elements.

Werden die Gleichungen (5.5), (5.6), (5.7), (5.8) und (5.9) in (5.4) eingesetzt und wird  $\delta\vec{U}_e$  ausgeklammert, ergibt sich:

$$\delta\vec{U}_e^T \left( \int_V \mathbf{B}^T \mathbf{E} \mathbf{B} dV \vec{U}_e - \vec{F}_e - \int_O \mathbf{N}_O^T \vec{p}_O dO - \int_V \mathbf{N}^T \vec{p} dV \right) = 0 \quad (5.10)$$

Da  $\delta\vec{U}_e \neq 0$  eine beliebige virtuelle Verschiebung darstellt, muss der Klammerausdruck verschwinden. Daraus folgt die Kraft-Verschiebungsbeziehung:

$$\underbrace{\int_V \mathbf{B}^T \mathbf{E} \mathbf{B} dV}_{\mathbf{k}_e} \vec{U}_e = \vec{F}_e + \underbrace{\int_O \mathbf{N}_O^T \vec{p}_O dO}_{\vec{F}_e(\vec{p}_O)} + \underbrace{\int_V \mathbf{N}^T \vec{p} dV}_{\vec{F}_e(\vec{p})} \quad (5.11)$$

$\mathbf{k}_e$  : die Steifigkeitsmatrix

$\vec{F}_e$  : der Vektor der Knotenkräfte

$\vec{F}_e(\vec{p}_O)$  : der äquivalente Lastvektor der Oberflächenlasten

$\vec{F}_e(\vec{p})$  : der äquivalente Lastvektor der Volumenkräfte

## 5.2 Klassifizierung und Kompatibilität

Die Elemente in ilbFE, wie auch in anderen FE-Programmen lassen sich nach mehreren Kriterien klassifizieren.

Elemente können nach deren Typ (Stab, Balken, Scheibe, Platte, Schale, Volumen), nach der Anzahl der Knoten und dem Analyserraum (2D, 3D) eingeteilt werden. Entsprechend dieser Klassifizierung wird jedem Element in ilbFE nach [Abbildung 5.1](#) eine eindeutige, dreistellige ID zugeordnet.

1	2	3
<i>Element-Typ</i>	<i>Knotenzahl</i>	<i>Analyserraum</i>
1: Stab		2: 2D
2: Balken		3: 3D
3: Scheibe		
4: Platte		
5: Schale		
6: Volumen		

**Abbildung 5.1:** Aufbau der Element-ID

Um verschiedene Elemente zu kombinieren, müssen die Freiheitsgrade an den Knoten kompatibel sein.

Elemente für 2D und 3D sollten nicht gemischt werden, da 2D Elemente in z-Richtung keine Freiheitsgrade besitzen.

Außerdem wird zwischen Kontinuums-elementen (Stäbe, Scheiben, Volumen, ...) und Strukturelementen (Balken, Platten, Schalen, ...) unterschieden.

Kontinuums-elemente besitzen in den Raumrichtungen nur Verschiebungsfreiheitsgrade  $(u_i, v_i, (w_i))$ . Strukturelemente hingegen verwenden kinematischen Annahmen und besitzen neben den Verschiebungsfreiheitsgraden  $(u_i, v_i, (w_i))$  noch zusätzliche Verdrehungsfreiheitsgrade  $(\varphi_{xi}, \varphi_{yi}, (\varphi_{zi}))$ .

Während ilbFE grundsätzlich die Kombination von Kontinuums- und Strukturelementen unterstützt, erfordert die gemeinsame Nutzung einen Mehraufwand bei der Vernetzung durch den Anwender.

Die rotatorischen Freiheitsgrade werden an gemeinsamen Knoten ignoriert, sodass in diesen Freiheitsgraden keine vollständige Verbindung zwischen den Elementen hergestellt wird. Die Struktur kann dadurch statisch unterbestimmt werden, was sich in einer singulären Steifigkeitsmatrix bemerkbar macht.

Dem Problem kann man durch intelligente Vernetzung (Überlappung an Übergängen von Strukturelement- und Kontinuums-elementbereichen) begegnen.

Tabelle 5.1 bietet eine Übersicht über die in ilbFE implementierten Elemente, deren IDs, Funktionsnamen (siehe Kapitel 8) und die Freiheitsgrade der Elementknoten.

Klassifizierung	ID	Funktionen	Freiheitsgrade der Knoten
2D - 2 Knoten - Stab	122	bar2	$(u_i \ v_i)^T$
3D - 2 Knoten - Stab	123	bar3	$(u_i \ v_i \ w_i)^T$
2D - 2 Knoten - Balken	222	beam2	$(u_i \ v_i \ \varphi_{zi})^T$
3D - 2 Knoten - Balken	223	beam3	$(u_i \ v_i \ w_i \ \varphi_{xi} \ \varphi_{yi} \ \varphi_{zi})^T$
2D - 3 Knoten - Scheibe	332	plant	$(u_i \ v_i)^T$
2D - 4 Knoten - Scheibe	343	plani4	$(u_i \ v_i \ w_i)^T$
2D - 8 Knoten - Scheibe	383	plani8	$(u_i \ v_i \ w_i)^T$
3D - 4 Knoten - Schale	543	shelli4	$(u_i \ v_i \ w_i \ \varphi_{xi} \ \varphi_{yi} \ \varphi_{zi})^T$
3D - 8 Knoten - Volumen	683	solis8	$(u_i \ v_i \ w_i)^T$

**Tabelle 5.1:** Übersicht über die Elemente

## 5.3 Stabelemente

Bei Stabelementen handelt es sich um linienförmige Elemente, die nur in ihrer Längsrichtung Kräfte übertragen können.

Die zu integrierende Matrix  $\mathbf{B}$  ist konstant, wodurch die Steifigkeitsmatrix nicht jedes mal aus Ansatzfunktionen und Differentialmatrix integriert werden muss, sondern direkt entsprechend der Elementgeometrie und Eigenschaften berechnet werden kann.

Stäbe benötigen die Querschnittsfläche  $A$  und den E-Modul  $E$  als Geometrie- beziehungsweise Materialeigenschaften.

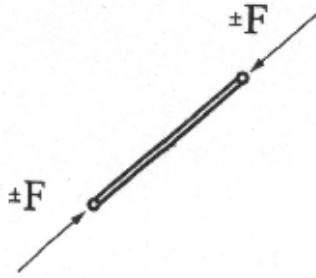


Abbildung 5.2: Ein Stab

Es wird angenommen, dass der E-Modul und die Querschnittsfläche über die Länge des Stabes konstant ist.

Die in ilbFE eingebundenen Stabelemente eignen sich zur Linearen Statik Analyse.

### 5.3.1 2D - 2 Knoten - Stab

Der **2D - 2 Knoten - Stab** ist ein Stabelement aus dem Programmpaket CALFEM [Aus+04] mit 2 Knoten für den 2D-Raum.

Da  $\mathbf{B}$  für linienförmige Elemente konstant ist, wird sie nicht für jedes Element explizit berechnet. Stattdessen wird direkt die Steifigkeitsmatrix  $\mathbf{k}_e$  bestimmt.

$$\underbrace{\begin{pmatrix} F_{x1} \\ F_{y1} \\ F_{x2} \\ F_{y2} \end{pmatrix}}_{\vec{F}_e} = \frac{EA}{l} \underbrace{\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{\mathbf{k}_e} \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{pmatrix}}_{\vec{U}_e} \quad (5.12)$$

Zur Kompatibilität der Elemente im 2D-Raum wurde  $\mathbf{k}_e$  mit Null-Zeilen und -Spalten aufgefüllt und die Freiheitsgrade  $v_1$  und  $v_2$  eingeführt.

Weiterführende Informationen können [Anhang C](#) entnommen werden.



### 5.3.2 3D - 2 Knoten - Stab

Der **3D - 2 Knoten - Stab** ist ein Stabelement aus dem Programmpaket CALFEM [Aus+04] mit 2 Knoten für den 3D-Raum.

Wie bei anderen Stabelementen kann die Steifigkeitsmatrix direkt – ohne Integration – bestimmt werden.

$$\underbrace{\begin{pmatrix} F_{x1} \\ F_{y1} \\ F_{z1} \\ F_{x2} \\ F_{y2} \\ F_{z2} \end{pmatrix}}_{\vec{F}_e} = \frac{EA}{l} \underbrace{\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\mathbf{k}_e} \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \end{pmatrix}}_{\vec{U}_e} \quad (5.13)$$

Hierzu wurden die Freiheitsgrade  $v_1$ ,  $w_1$ ,  $v_2$  und  $w_2$  eingeführt und die Matrix  $\mathbf{k}_e$  mit Nullen aufgebläht.

Weiterführende Informationen können [Anhang C](#) entnommen werden.

## 5.4 Balkenelemente

Balkenelemente sind linienförmige Elemente, die sowohl in Längs- als auch in Querrichtung Kräfte und Momente übertragen können.

Da bei linienförmigen Elementen nur eine Integrationsrichtung zur Berechnung der Steifigkeitsmatrix existiert, muss die Steifigkeitsmatrix nicht über Ansatzfunktion und Differentialmatrix berechnet werden, sondern wird direkt aus der Elementgeometrie bestimmt.

Balken benötigen die Flächenträgheitsmomente ( $I_y$  und für 3D zusätzlich  $I_z$ ) als Geometrie-Eigenschaft und den E-Modul  $E$  als Material-Eigenschaft.

Für einen zusätzlichen Stab-Anteil wird gegebenenfalls die Querschnittsfläche  $A$  und für einen zusätzlichen Torsionsstab-Anteil das Torsionsflächenträgheitsmoment  $I_p$  und der Schubmodul  $G$  benötigt.

Damit beim 3D-Stab die Hauptachsen eindeutig bestimmt sind, muss die Richtung lokalen  $z$ -Achse über  $\mathbf{eo} = [\mathbf{xz} \ \mathbf{yz} \ \mathbf{zz}]$  vorgegeben werden.

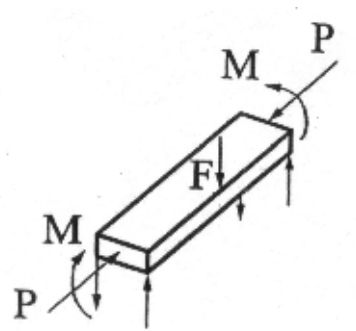


Abbildung 5.3: Ein Balken

Es werden folgende Annahmen getroffen:

- Die Querkräfte greifen im Schubmittelpunkt an.
- Der E-Modul und die Querschnittsfläche sind über die Länge des Stabes konstant.
- Der Querschnitt befindet sich im Hauptachsensystem.

Die in ilbFE eingebundenen Balkenelemente eignen sich zur Linearen Statik Analyse. Das [2D - 2 Knoten - Balken](#) ([Unterabschnitt 5.4.1](#)) eignet sich außerdem zur linearen Eigenwert- und Modalanalyse.

### 5.4.1 2D - 2 Knoten - Balken

Der [2D - 2 Knoten - Balken](#) ist ein Balkenelement mit Stabanteil mit 2 Knoten für den 2D-Raum. Es eignet sich sowohl zur Statik- als auch zur Dynamik-Analyse.

Die Steifigkeitsmatrix kann, wie bei anderen linienförmigen Elementen, direkt bestimmt werden und muss nicht integriert werden.

$$\underbrace{\begin{pmatrix} F_{x1} \\ F_{y1} \\ M_{z1} \\ F_{x2} \\ F_{y2} \\ M_{z2} \end{pmatrix}}_{\vec{F}_e} = \underbrace{\begin{pmatrix} \frac{EA}{l} & 0 & 0 & -\frac{EA}{l} & 0 & 0 \\ 0 & 12\frac{EI_z}{l^3} & 6\frac{EI_z}{l^2} & 0 & -12\frac{EI_z}{l^3} & 6\frac{EI_z}{l^2} \\ 0 & 6\frac{EI_z}{l^2} & 4\frac{EI_z}{l} & 0 & -6\frac{EI_z}{l^2} & 2\frac{EI_z}{l} \\ -\frac{EA}{l} & 0 & 0 & \frac{EA}{l} & 0 & 0 \\ 0 & -12\frac{EI_z}{l^3} & -6\frac{EI_z}{l^2} & 0 & 12\frac{EI_z}{l^3} & -6\frac{EI_z}{l^2} \\ 0 & 6\frac{EI_z}{l^2} & 2\frac{EI_z}{l} & 0 & -6\frac{EI_z}{l^2} & 4\frac{EI_z}{l} \end{pmatrix}}_{\mathbf{k}_e} \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ \varphi_{z1} \\ u_2 \\ v_2 \\ \varphi_{z2} \end{pmatrix}}_{\vec{U}_e} \quad (5.14)$$

Die Längsspannungen werden für den Stab- und Balkenanteil getrennt berechnet und anschließend entsprechend dem Vorzeichen der Stab-Längsspannung superpositioniert. Durch dieses Vorgehen berechnet das Element immer die maximale Längsspannung für einen Querschnitt.

Ist keine Längskraft vorhanden, wird automatisch die höchste Zugspannung ausgegeben.

Bei Dynamik-Analyse wird analog zur Steifigkeitsmatrix noch die Massenmatrix  $\mathbf{m}_e$  bestimmt. Die Dämpfungsmatrix  $\mathbf{c}_e$  wird aus Steifigkeitsmatrix und Massenmatrix über die Parameter  $a$  und  $b$  nach (5.15) berechnet.

$$\mathbf{c}_e = a\mathbf{k}_e + b\mathbf{m}_e \quad (5.15)$$

Das Element basiert teilweise auf dem Programmpaket CALFEM [Aus+04]. Weiterführende Informationen können Anhang C entnommen werden.

### 5.4.2 3D - 2 Knoten - Balken

Der 3D - 2 Knoten - Balken ist ein Balkenelement mit Stabanteil mit 2 Knoten für den 3D-Raum.

Wie bei den anderen linienförmigen Elementen kann die Steifigkeitsmatrix  $\mathbf{k}_e$  ohne Integration bestimmt werden. Zur Übersicht ist  $\mathbf{k}_e$  hier in  $\mathbf{k}_{e1}$  und  $\mathbf{k}_{e2}$  aufgeteilt.

$$\underbrace{\begin{pmatrix} F_{x1} \\ F_{y1} \\ F_{z1} \\ M_{x1} \\ M_{y1} \\ M_{z1} \\ F_{x2} \\ F_{y2} \\ F_{z2} \\ M_{x2} \\ M_{y2} \\ M_{z2} \end{pmatrix}}_{\vec{F}_e} = \underbrace{\begin{pmatrix} \mathbf{k}_{e1} & \mathbf{k}_{e2} \\ \mathbf{k}_{e2}^T & \mathbf{k}_{e1} \end{pmatrix}}_{\mathbf{k}_e} \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ w_1 \\ \varphi_{x1} \\ \varphi_{y1} \\ \varphi_{z1} \\ u_2 \\ v_2 \\ w_2 \\ \varphi_{x2} \\ \varphi_{y2} \\ \varphi_{z2} \end{pmatrix}}_{\vec{U}_e} \quad (5.16)$$

$$\mathbf{k}_{e1} = \begin{pmatrix} \frac{EA}{l} & 0 & 0 & 0 & 0 & 0 \\ 0 & 12\frac{EI_z}{l^3} & 0 & 0 & 0 & 6 \\ 0 & 0 & 12\frac{EI_y}{l^3} & 0 & -6\frac{EI_y}{l^2} & 0 \\ 0 & 0 & 0 & \frac{GI_t}{l} & 0 & 0 \\ 0 & 0 & -6\frac{EI_y}{l^2} & 0 & 4\frac{EI_y}{l} & 0 \\ 0 & 6\frac{EI_z}{l^2} & 0 & 0 & 0 & 4\frac{EI_z}{l} \end{pmatrix} \quad (5.17)$$

$$\mathbf{k}_{e2} = \begin{pmatrix} -\frac{EA}{l} & 0 & 0 & 0 & 0 & 0 \\ 0 & -12\frac{EI_z}{l^3} & 0 & 0 & 0 & 6\frac{EI_z}{l^2} \\ 0 & 0 & -12\frac{EI_y}{l^3} & 0 & -6\frac{EI_y}{l^2} & 0 \\ 0 & 0 & 0 & -\frac{GI_t}{l} & 0 & 0 \\ 0 & 0 & 6\frac{EI_y}{l^2} & 0 & 2\frac{EI_y}{l} & 0 \\ 0 & -6\frac{EI_z}{l^2} & 0 & 0 & 0 & 2\frac{EI_z}{l} \end{pmatrix} \quad (5.18)$$

Der Stabanteil des Balkens wurde analog zu [Abschnitt 5.3 Stabelemente](#) in die Steifigkeitsmatrix eingefügt.

Damit ebenfalls Freiheitsgrade für die Rotation um die lokale x-Achse ( $\varphi_{x1}$  und  $\varphi_{x2}$ ) eingeführt werden konnten, wurde die die Torsionssteifigkeit  $k_{e44} = k_{e10 \ 10} =$

$-k_{e4\ 10} = -k_{e10\ 4}$  als Torsionsstab eingeführt.

Um die maximale Normalspannung in Längsrichtung für einen Querschnitt zu erhalten, werden die Normalspannungen für den Stab und die Biege-Anteile unabhängig voneinander berechnet. Anschließend werden sie entsprechend dem Vorzeichen der Stab-Normalspannung zueinander addiert, sodass der Anteil der Biegung in Richtung der Längskraft berücksichtigt wird.

Ist keine Längskraft vorhanden, wird automatisch die höchste Zugspannung ausgegeben.

Das Element basiert teilweise auf dem Programmpaket CALFEM [[Aus+04](#)]. Weiterführende Informationen können [Anhang C](#) entnommen werden.

## 5.5 Scheibenelemente

Scheibenelemente sind flächige Kontinuumsэлеmente im 2D oder 3D-Raum. Sie können nur in ihrer Ebene Kräfte aufnehmen.

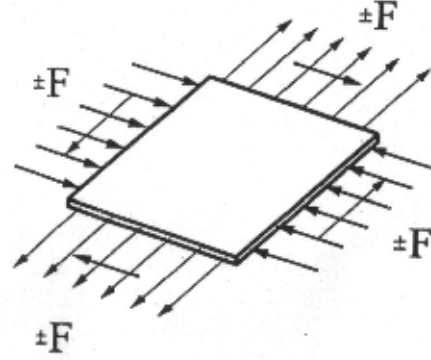


Abbildung 5.4: Eine Scheibe

Ihre einzige Geometrie-Eigenschaft ist die Dicke  $t$ . Beim ebenen Spannungszustand und isotropem Material wird die Werkstoffmatrix  $\mathbf{E}$  nach (5.19) berechnet.

$$\mathbf{E} = \frac{E}{1 - \nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \quad (5.19)$$

Bei symmetrisch, orthotropem Materialverhalten wird Gleichung (5.20) benutzt.

$$\mathbf{E} = \begin{pmatrix} \frac{E_{\parallel}}{1-\nu^2} & \frac{\nu E_{\perp}}{1-\nu^2} & 0 \\ \frac{\nu E_{\perp}}{1-\nu^2} & \frac{E_{\perp}}{1-\nu^2} & 0 \\ 0 & 0 & G_{\#} \end{pmatrix} \quad (5.20)$$

### 5.5.1 2D - 3 Knoten - Scheibe

Die 2D - 3 Knoten - Scheibe ist ein Scheibenelement aus dem Programmpaket CALFEM [Aus+04] mit 3 Knoten für den 2D-Raum.

$$\underbrace{\begin{pmatrix} F_{x1} \\ F_{y1} \\ F_{x2} \\ F_{y2} \\ F_{x3} \\ F_{y3} \end{pmatrix}}_{\vec{F}_e} = \underbrace{t \cdot \Delta \cdot \mathbf{B}^T \mathbf{E} \mathbf{B}}_{\mathbf{k}_e} \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix}}_{\vec{U}_e} \quad (5.21)$$

Da die Verzerrungs-Verschiebungs-Matrix **B** (5.22) unabhängig vom zu integrierenden Volumen  $V$  ist muss die Steifigkeitsmatrix nicht numerisch integriert werden sondern kann nach (5.21) per Matrixmultiplikation berechnet werden.

$$\mathbf{B} = \frac{1}{2\Delta} \begin{pmatrix} y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \\ x_3 - x_2 & y_2 - y_3 & x_1 - x_3 & y_3 - y_1 & x_2 - x_1 & y_1 - y_2 \end{pmatrix} \quad (5.22)$$

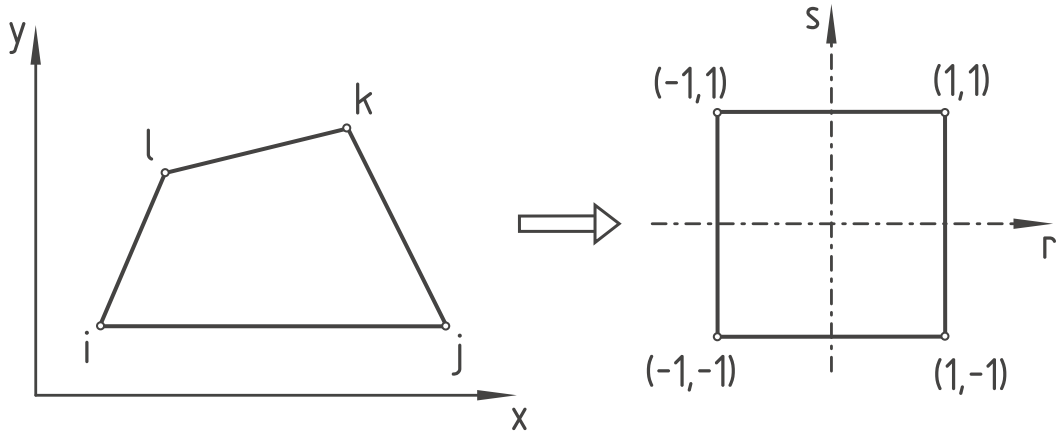
Dabei ist  $\Delta$  der Flächeninhalt des Dreieckelements, die Hälfte der Determinante in (5.23).

$$\Delta = \frac{1}{2} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \quad (5.23)$$

Weiterführende Informationen können [Anhang C](#) entnommen werden.

### 5.5.2 2D - 4 Knoten - Scheibe

Die [2D - 4 Knoten - Scheibe](#) ist ein isoparametrisches Scheibenelement aus dem Programmpaket CALFEM [Aus+04] mit 4 Knoten für den 2D-Raum. Die Formfunktionen interpolieren die Verschiebungen zwischen den Knoten linear.



**Abbildung 5.5:** Isoparametrische Formulierung einer Scheibe

Dazu wird die Form der Scheibe über die Jacobi-Matrix **J** auf ein Quadrat mit der Seitenlänge Zwei zurückgeführt und ein lokales Koordinatensystem  $(r, s)$  mit Ursprung in der Mitte des Elementes eingeführt.

Die Integration wird dann numerisch mit Hilfe der Gauss-Quadratur (Standardmäßig  $2 \times 2$ ) ausgeführt.

Weiterführende Informationen können [Anhang C](#) entnommen werden.

### 5.5.3 2D - 8 Knoten - Scheibe

Die [2D - 8 Knoten - Scheibe](#) ist ein isoparametrisches Scheibenelement aus dem Programmpacket CALFEM [[Aus+04](#)] mit 8 Knoten für den 2D-Raum. Die Formfunktionen interpolieren die Verschiebungen zwischen den Knoten quadratisch.

Die Integration wird dann numerisch mit Hilfe der Gauss-Quadratur (Standardmäßig  $3 \times 3$ ) ausgeführt.

Weiterführende Informationen können [Anhang C](#) entnommen werden.



## 5.6 Schalenelemente

Schalenelemente sind flächige Strukturelemente im 3D-Raum. Sie können in der Ebene Kräfte und Momente, sowie Kräfte senkrecht zur Ebene aufnehmen. Momente, deren Vektor senkrecht zur Fläche steht können nicht übertragen werden.

Wie bei der Scheibe ist die einzige Geometrie-eigenschaft die Dicke  $t$ . Die Werkstoffmatrix  $\mathbf{E}$  wird für isotrope Werkstoffe nach (5.24) berechnet.

$$\mathbf{E} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1 & \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-\nu} & 1 & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix} \quad (5.24)$$

Für symmetrische, orthotrope UD-Schichten ergibt sich  $\mathbf{E}$  nach Gleichung (5.25).

$$\mathbf{E} = \frac{1}{\alpha} \begin{pmatrix} E_{\parallel} & \nu E_{\perp} & \nu E_{\perp} & 0 & 0 & 0 \\ \nu E_{\perp} & E_{\perp} & \nu E_{\perp} & 0 & 0 & 0 \\ \nu E_{\perp} & \nu E_{\perp} & E_{\perp} & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha G_{\#} & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha G_{\#} & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha G_{\#} \end{pmatrix} \quad (5.25)$$

mit  $\alpha = 1 - \nu^2 \frac{E_{\perp}}{E_{\parallel}}$

Es wird angenommen, dass die Dicke  $t$  der Schale sich über dem Element nicht verändert.

Da die zu integrierenden Matrizen  $\mathbf{B}^T \mathbf{E} \mathbf{B}$  abhängig von mehreren Integrationsvariablen sind, muss die Integration der Steifigkeitsmatrizen für jedes Element numerisch ausgeführt werden.

Das in ilbFE eingebundene [3D - 4 Knoten - Schale](#) ([Unterabschnitt 5.6.1](#)) eignet sich zur Linearen Statik Analyse und zur linearen Eigenwert- und Modalanalyse.

### 5.6.1 3D - 4 Knoten - Schale

Die **3D - 4 Knoten - Schale** ist ein isoparametrisches, ebenes Schalenelement für dreidimensionale Strukturen mit 4 Knoten. Die Formfunktionen interpolieren die Verschiebungen zwischen den Knoten linear.

Da die Schale ungekrümmt ist, können Scheibenanteil ( $k_{eij}^S$  in (5.26)) und Plattenanteil ( $k_{eij}^P$  in (5.26)) getrennt betrachtet werden. Die Berechnung des Plattenanteils erfolgt analog zu einer schubweichen Reissner-Mindlin-Platte.

Um Kompatibilität mit senkrecht angeschlossenen Schalen zu gewährleisten, wurden Freiheitsgrade senkrecht zur Schalenebene ( $\varphi_{z1}, \varphi_{z2}, \varphi_{z3}, \varphi_{z4}$ ) eingeführt.

Da die Steifigkeit der Schale in dieser Richtung aber Null ist, wäre der Rang der resultierenden Steifigkeitsmatrix zu niedrig und das Problem wäre singulär. Darum wurde eine fiktive Steifigkeit ( $k_{eij}^K$  in (5.26)) für diese Freiheitsgrade zu einem Tausendstel des kleinsten Diagonaleintrages von  $\mathbf{k}_e$  gewählt [Rei+09].

$$\underbrace{\begin{pmatrix} F_{x1} \\ F_{y1} \\ F_{z1} \\ M_{x1} \\ M_{y1} \\ M_{z1} \\ \vdots \\ M_{z4} \end{pmatrix}}_{\vec{F}_e} = \underbrace{\begin{pmatrix} k_{e11}^S & k_{e12}^S & 0 & 0 & 0 & 0 & \cdots & 0 \\ k_{e11}^S & k_{e12}^S & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & k_{e33}^P & k_{e34}^P & k_{e35}^P & 0 & \cdots & 0 \\ 0 & 0 & k_{e43}^P & k_{e44}^P & k_{e45}^P & 0 & \cdots & 0 \\ 0 & 0 & k_{e53}^P & k_{e54}^P & k_{e55}^P & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{e66}^K & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & k_{e24}^K \end{pmatrix}}_{\mathbf{k}_e} \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ w_1 \\ \varphi_{x1} \\ \varphi_{y1} \\ \varphi_{z1} \\ \vdots \\ \varphi_{z4} \end{pmatrix}}_{\vec{U}_e} \quad (5.26)$$

Die Integration der Steifigkeitsmatrix  $\mathbf{k}_e$  (`shell14e()`) findet numerisch mit Hilfe einer Gauss-Quadratur mit bis zu 3 Integrationspunkten statt. In jedem Fall wird der Schubanteil der Steifigkeitsmatrix reduziert mit einer Ordnung von  $1 \times 1$  integriert, um Schubsperrern zu vermeiden.

Der Scheiben- und Kirchhoffsche-Plattenanteil der Steifigkeitsmatrix werden standardmäßig mit einer Integrationsordnung von  $2 \times 2$  integriert.

Zur Bestimmung der Elementspannungen (`shell14s()`) wird die Verzerrungs-Verschiebungsmatrix  $\mathbf{B}$  an den Eckpunkten ausgewertet.

Bei dynamischer Analyse (`shell14d()`) für die Massenmatrix  $\mathbf{m}_e$  wird die Masse der Schale in die Eckpunkte konzentriert und die Rotationsanteile werden vernachlässigt. Die Dämpfungsmatrix  $\mathbf{c}_e$  wird aus der Steifigkeitsmatrix  $\mathbf{k}_e$  und der Massenmatrix  $\mathbf{m}_e$  über die Gleichung (5.27) bestimmt.

$$\mathbf{c}_e = a\mathbf{k}_e + b\mathbf{m}_e \quad (5.27)$$

## 5.7 Volumenelemente

Volumenelemente sind räumliche Kontinuumsselemente für den 3D-Raum. Sie können in jedem Knoten Kräfte aufnehmen, allerdings können Momente nur über die Kombination von Kräften über mehrere Knoten eingebracht werden.

Da die Geometrie durch die Knotenkoordinaten vollständig beschrieben wird, müssen keine Element-Geometrieeigenschaften an Volumenelemente übergeben werden. Bei isotropem Materialverhalten wird die Werkstoffmatrix  $\mathbf{E}$  nach Gleichung (5.28) bestimmt.

$$\mathbf{E} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1 & \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-\nu} & 1 & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix} \quad (5.28)$$

### 5.7.1 3D - 8 Knoten - Volumen

Das [3D - 8 Knoten - Volumen](#) ist ein isoparametrisches Volumenelement aus dem Programmpaket CALFEM [[Aus+04](#)] mit 8 Knoten für den 3D-Raum. Die Formfunktionen interpolieren die Verschiebungen zwischen den Knoten quadratisch.

Die Integration wird dann numerisch mit Hilfe der Gauss-Quadratur (Standardmäßig  $2 \times 2 \times 2$ ) ausgeführt.

Weiterführende Informationen können [Anhang C](#) entnommen werden.

## 6. Eingabe und Ausgabe

Die Eingabe- und Ausgabe in `ilbFE` ist über ein flexibles, modulares Eingabekartensystem implementiert.

Jede Karte ist über eine Header-Zeile, die mit dem Buchstaben `H` beginnt, dann den Kartennamen definiert und anschließend der Reihe nach die einzelnen Kartenparameter auflistet.

H	Kartennamen	Parameter1	Parameter2	Parameter3
C	Kommentare folgen auf ein C!			
	Kartennamen	123	10E-5	0.314

**Abbildung 6.1:** Beispiel-Header mit 3 Parametern

Auf die Header-Zeile folgen die einzelnen Karten, die den Parametern für jede Karte einen eindeutigen Wert zuweisen.

Der Wert eines Parameters kann dabei sowohl eine Zahl (z.B. ein Index zur Referenzierung eines Knotens) als auch ein String sein, allerdings speichert MATLAB keine Zahlen und Strings gleichzeitig in einer Matrix.

Darum nutzt die Einleseprozedur *struct*-Datenstrukturen, die flexibler auf verschiedene Datentypen reagieren. Da man aber im Hauptprogramm mit Matrizen rechnet, bietet es sich an schon in der Eingabedatei Zahlen (Integer, Double, ...) zu verwenden.

Bei der Formatierung von Karten wird nicht zwischen einem oder mehreren Leerzeichen oder Tabulatoren unterschieden. Entscheidend ist die Reihenfolge der Parameter und dass mindestens ein Leerzeichen oder Tabulator zwischen den Parametern steht.

Die Anzahl der Parameter in einer Karte ist nicht begrenzt, allerdings werden Karten bei zu vielen Parametern schnell sehr unübersichtlich, sodass man erwägen sollte lange Karten in mehrere Karten aufzuteilen. Ein manueller Zeilenumbruch beendet in jedem Fall die Karte.

Kommentare können mit einem vorgestellten `C` gefolgt von mindestens einem Leerzeichen eingefügt werden.

## 6.1 Eingabedatei

In der Eingabedatei (Endung: .in) ist die zu untersuchende Struktur gespeichert. Sie beinhaltet die Position der Knoten, die Elemente, die die Knoten verbinden, Element-Eigenschaften und Materialien. Zusätzlich werden Randbedingungen und Lasten vorgegeben.

Je nach verwendeten Elementen und Analyseart müssen unterschiedliche Parameter und Variablen angegeben werden, damit die Rechnung durchgeführt werden kann. So braucht zum Beispiel ein 2D-Stab nur X- und Y-Koordinaten und die Fläche  $A$  als Eigenschaft, während ein 3D-Balken X-, Y- und Z-Koordinaten, die Fläche  $A$ , Flächenträgheitsmomente ( $I_y$ ,  $I_z$ ,  $I_T$ ) und die Orientierung der lokalen z-Achse ( $x_z$ ,  $y_z$ ,  $z_z$ ) braucht.

Elemente, Knoten, Material- und Geometrie-Eigenschaften sind immer über eine ID eindeutig definiert. Diese ID darf innerhalb einer Klasse nicht mehrmals vorkommen, kann aber einen beliebigen ganzzahligen Wert haben. Die Reihenfolge oder Vollständigkeit der IDs ist irrelevant.

Es folgt eine Liste der bereits implementierten Karten mit Beispielen und Erläuterungen für verschiedene Elemente und Analysearten. Für eigene Elemente und Analysearten kann der Nutzer zusätzlich eigene Karten definieren, deren Parameter er im Präprozessor auslesen kann.

Eine genaue Beschreibung der Implementation von Elementen und Solvern befindet sich in [Kapitel 9](#).

### 6.1.1 Title

Mit dem Befehl **Title** kann man dem Problem einen Namen oder eine kurze Beschreibung geben.

Title Hierhin kommt der Name des Problems!
--

**Abbildung 6.2:** Beispiel-Karte für **Title**

Der **Title**-Befehl muss in der Eingabedatei vorhanden sein, kann aber auch einem leeren String vorangestellt sein.

### 6.1.2 Solver

Die Karte **Solver** bestimmt den verwendeten Löser und die Analysemethode.

Neben dem Typ (**Type**) des Löfers kann auch für iterative Verfahren eine maximale Anzahl von Schritten (**Steps**) oder eine Fehlertoleranzschranke (**Error**) eingegeben werden.

Für die Lineare Eigenwert- und Modalanalyse gibt **Steps** die Anzahl der gesuchten Eigenfrequenzen und deren Moden an.

H Solver	Type	Steps	Error
Solver	1	10	1e-12

**Abbildung 6.3:** Beispiel-Karte für **Solver** für Lineare Statik Analyse

Die in ilbFE implementierten Solver sind in [Tabelle 6.1](#) aufgelistet.

Typ	Analyseart	Beschreibung	Funktion	Beschreibung
1	Lineare Statik	<a href="#">Abschnitt 4.1</a>	<code>solver1()</code>	<a href="#">Unterabschnitt 8.4.1</a>
2	Eigenfrequenz	<a href="#">Abschnitt 4.2</a>	<code>solver2()</code>	<a href="#">Unterabschnitt 8.4.2</a>

**Tabelle 6.1:** Solver in ilbFE

Nicht benötigte Parameter (z.B. **Steps** bei nicht-iterativen Verfahren) können entweder aus der Header-Zeile entfernt werden oder können einen beliebigen Wert bekommen. Wird der Wert im Löser nicht vorgesehen, wird er ignoriert.

### 6.1.3 Nodes

Die einzelnen Knotenpunkte mit ihren Koordinaten ( $X, Y, \dots$ ) und einer eindeutigen ID werden über die Karte **Nodes** eingegeben. Entsprechend dem Problem-Raum und den genutzten Elementen muss die Header-Zeile um Z-Koordinaten ( $Z$ ) erweitert werden.

H Nodes	ID	X	Y
Nodes	1	0	0
Nodes	2	1000	0
Nodes	3	500	1000

**Abbildung 6.4:** Beispiel-Karte für 3 **Nodes** in 2D

Der Präprozessor von ilbFE versucht die benötigten Dimensionen aus den genutzten Elementen zu erkennen und füllt gegebenenfalls die Koordinaten Matrix entsprechend mit Null-Einträgen für die Z-Koordinate.

Allerdings sollte der Nutzer in der Regel dafür sorgen, dass die Eingabedatei schlüssige und eindeutige Daten enthält.

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	11	0	100	0

**Abbildung 6.5:** Beispiel-Karte für 2 Nodes in 3D

### 6.1.4 Elements

Elemente werden über die **Elements**-Karte eingelesen. Genau wie bei den Knoten besitzt jedes Element eine eindeutige ID, sowie einen Typ (**Type**), ein Material (**MatID**) und Geometrie-Eigenschaften (**PropID**).

Die Knotenzahl variiert bei verschiedenen Elementen, sodass der Nutzer bei Bedarf die Header-Zeile mit weiteren Knoten (genannt **N3**, **N4**, ...) für die einzelnen Elemente erweitern muss.

H Elements	ID	Type	MatID	PropID	N1	N2
Elements	1	122	1	1	1	2
Elements	2	122	1	1	2	3
Elements	3	122	1	1	3	1

**Abbildung 6.6:** Beispiel-Karte für Elements mit 3 Stäben

Zur Kombination mehrerer Elemente mit unterschiedlichen Knotenzahlen sollte die Header-Zeile die maximale Anzahl von Knoten pro Element enthalten und die Elemente mit weniger Knoten mit Nullen aufgefüllt werden.

H Elements	ID	Type	MatID	PropID	N1	N2	N3	N4
Elements	1	543	1	1	1	102	113	3
Elements	13	223	3	2	102	103	0	0

**Abbildung 6.7:** Beispiel-Karte für Elements mit Schale und Stab

Eine Zusammenfassung der verfügbaren Elemente in ilbFE kann in [Tabelle 6.2](#) auf Seite [36](#) gefunden werden.

Ausführliche Beschreibungen der Elemente befinden sich in [Kapitel 5](#).

### 6.1.5 Materials

Materialien in ilbFE werden als orthotrop eingegeben. Jedes Material hat eine eindeutige ID, auf die die Elemente über **MatID** zugreifen können.

Neben den Elastizitäts- und Gleitmoduln, sowie der Querkontraktionszahl ist bei orthotropen Materialien der Winkel **phi** der Materialorientierung nötig.

Bei isotropen Materialien sollten **Ep** und **Es** gleich sein und **Gq** nach Materialgesetzen berechnet werden.

Zusätzlich sind bei dynamischen Analysen die Dichte (**rho**) und die Dämpfungsfaktoren **a** und **b** ( $\mathbf{c}_e = a\mathbf{k}_e + b\mathbf{m}_e$ ) von Bedeutung.

H Materials	ID	Ep	Es	nue	Gq	phi	rho	a	b
Materials	1	2.1 e5	2.1 e5	0.3	1.05 e5	0	1	0.1	0.05

**Abbildung 6.8:** Beispiel-Karte für **Materials** bei einer dynamischen Analyse einer Schale

### 6.1.6 Properties

Mit der **Properties**-Karte werden Geometrie-Eigenschaften der Elemente eingelesen. Da diese zwischen den Element-Typen variieren, kann die Karte sehr stark von dem Beispiel abweichen. Die ID hingegen muss eindeutig vorhanden sein, sodass Elemente über ihre **PropID** auf diese zugreifen können.

Werden unterschiedliche Elemente kombiniert, muss die Header-Zeile von **Properties** die Parameter aller genutzten Elemente enthalten. Für eine Geometrie-Eigenschaft nicht benötigte Parameter sollten zu Null gesetzt werden.

H Properties	ID	A	Iy	Iz	Kv	xz	yz	zz
Properties	1	1000	0	0	0	0	0	0
Properties	2	1000	10000	5000	0.8	0	-1	0

**Abbildung 6.9:** Beispiel-Karte für **Properties** eines Stabes und eines 3D-Balkens

Die benötigten Parameter für die Elemente sind in [Kapitel 5](#) erläutert. Eine Übersicht ist [Tabelle 6.2](#) zu entnehmen.

Klassifizierung	ID	Geometrie-Eigenschaften
<a href="#">2D - 2 Knoten - Stab</a>	122	A
<a href="#">3D - 2 Knoten - Stab</a>	123	A
<a href="#">2D - 2 Knoten - Balken</a>	222	A, I
<a href="#">3D - 2 Knoten - Balken</a>	223	A, Iy, Iz, Kv, xz, yz, zz
<a href="#">2D - 3 Knoten - Scheibe</a>	332	t
<a href="#">2D - 4 Knoten - Scheibe</a>	343	t
<a href="#">2D - 8 Knoten - Scheibe</a>	383	t
<a href="#">3D - 4 Knoten - Schale</a>	543	t
<a href="#">3D - 8 Knoten - Volumen</a>	683	

**Tabelle 6.2:** Übersicht der Elemente in ilbFE und deren **Properties**

### 6.1.7 BC

Die Karte **BC** beschreibt die Randbedingungen der Struktur. **NodeID** gibt hierbei den gefesselten Knoten an, während **XDir**, **YDir**, **ZDir** und **rXDir**, **rYDir**, **rZDir** die



jeweiligen Verschiebungen beziehungsweise Verdrehungen in dieser Raumrichtung definieren.

Freie Verschiebungs- oder Drehrichtungen werden mit dem komplexen  $i$  gekennzeichnet.

H BC	NodeID	XDir	YDir
BC	1	0	0
BC	2	$i$	0

**Abbildung 6.10:** Beispiel-Karte für BC bei 2D-Kontinuumselementen

Die BC-Karte ist unbedingt an den untersuchten Raum und die verwendeten Elemente anzupassen. Benötigte beziehungsweise nicht-benötigte Freiheitsgrade sind in der Header-Zeile anzugeben beziehungsweise auszulassen.

H BC	NodeID	XDir	YDir	ZDir	rXDir	rYDir	rZDir
BC	1	0	0	0	0	0	0
BC	2	0	0	0	$i$	$i$	$i$
BC	3	0	0	0	0	0	0

**Abbildung 6.11:** Beispiel-Karte für BC bei 3D-Strukturelemente

### 6.1.8 Loads

Über die Loads-Karte werden die angreifenden, äußeren Lasten aufgebracht. **NodeID** definiert den Knoten in dem die Kraft angreift, während **ForceX**, **ForceY**, **ForceZ** und **MomentX**, **MomentY**, **MomentZ** die Komponenten der angreifenden Kraft beziehungsweise des angreifenden Momentes bestimmen.

H Loads	NodeID	ForceX	ForceY
Loads	2	0	1000
Loads	3	1000	2000

**Abbildung 6.12:** Beispiel-Karte für Loads bei 2D-Kontinuumselementen

Die Loads-Karte ist unbedingt an den untersuchten Raum und die verwendeten Elemente anzupassen. Benötigte beziehungsweise nicht-benötigte Freiheitsgrade sind in der Header-Zeile anzugeben beziehungsweise auszulassen.

H Loads	NodeID	ForceX	ForceY	ForceZ	MomentX	MomentY	MomentZ
Loads	101	100	$-1e4$	0	$-200$	5.0	0

**Abbildung 6.13:** Beispiel-Karte für Loads bei 3D-Strukturelementen

## 6.2 Ausgabedatei

Die Ausgabedatei (Endung: .out) speichert die Ergebnisse der Rechnung. Je nach genutzten Löser unterscheiden sich die Ausgabekarten und Ergebnisse.

Neben den Ergebnissen ist außerdem die komplette Struktur aus der Eingabedatei abgespeichert. Die Definition der Karten findet man in [Abschnitt 6.1](#).

### 6.2.1 nDisp

Die Karte **nDisp** beschreibt bei Linearen Statik Analyse die Verschiebungen (U, V, W) und Verdrehungen (rX, rY, rZ) der Knoten mit der Knoten-ID **nID**.

Bei nur zwei Raumdimensionen und Kontinuumselementen (keine Rotation der Knoten) werden die entsprechenden Verschiebungen und Verdrehungen mit Nullen aufgefüllt.

H	nDisp	nID	U	V	W	rX	rY	rZ
nDisp		103	0.322	-0	0.68	0	0	0
nDisp		104	0.456	-0	1.47	0	0	0

**Abbildung 6.14:** Beispiel-Karte für **nDisp** bei 3D-Strukturelementen

### 6.2.2 eStress

Die Spannungen in einem Element an den Knoten werden über die Karte **eStress** ausgegeben. Es wird die Element-ID (**eID**), der Element-Typ (**eType**), die Nummer des Knotens im Element (**eNode**) und die globale Knoten-ID (**nID**) zusätzlich zu den Spannungen (**sigX**, **sigY**, **sigZ**, **tauXY**, **tauYZ**, **tauZX**) ausgegeben.

H	eStress	eID	eType	eNode	nID	sigX	sigY	sigZ	tauXY	tauYZ	tauZX
eStress		1	543	1	1	392	118	0	188	0	0
eStress		1	543	2	102	392	118	0	-168	0	0
eStress		1	543	3	113	-392	-118	0	-168	0	0
eStress		1	543	4	3	-392	-118	0	188	0	0

**Abbildung 6.15:** Beispiel-Karte für **eStress** bei 3D-Strukturelementen

Bei Balkenelementen ist zu beachten, dass die maximalen Spannungen ausgegeben werden, da die Spannungen über den Querschnitt variieren.

### 6.2.3 mFreq

Über die Karte **mFreq** wird bei Modal- und Eigenschwingungsanalyse die Eigenfrequenz (**f**) der Schwingung und der zugehörige **Mode** ausgegeben.

Die Eigenfrequenzen sind aufsteigend vom kleinsten zum größten Wert sortiert.

H	mFreq	Mode	f
	mFreq	1	0.0231
	mFreq	2	0.136
	mFreq	3	0.203

**Abbildung 6.16:** Beispiel-Karte für **mFreq** mit den 3 niedrigsten Eigenfrequenzen

## 6.2.4 mDisp

Bei Modal- und Eigenschwingungsanalysen gibt **mDisp** die normierten Verschiebungen (**U**, **V**, **W**) und Verdrehungen (**rX**, **rY**, **rZ**) der Knoten mit der ID **nID** aus. **Mode** definiert dabei den Schwingungsmodus.

H	mDisp	Mode	nID	U	V	W	rX	rY	rZ
	mDisp	1	1	0	0	0	0	0	0
	mDisp	1	2	0.0043	$-2.17 \times 10^{-20}$	0.0625	0	0	0
	mDisp	1	3	0	0	0	0	0	0
	mDisp	1	4	$-0.0043$	$3.21 \times 10^{-20}$	0.0625	0	0	0

**Abbildung 6.17:** Beispiel-Karte für **mDisp** mit den normierten Verschiebungen von 3 Knoten bei der ersten Eigenschwingung

## 7. Beispiele

In diesem Kapitel werden einige Beispiele vorgestellt, die sich an [\[Rei+09\]](#) orientieren.

Zunächst werden in [Abschnitt 7.1](#) und [Abschnitt 7.2](#) Fachwerke aus den beiden implementierten Stabelementen unter einfachen Lasten betrachtet.

Die Beispiele von [Abschnitt 7.3](#) bis [Abschnitt 7.8](#) vergleichen den gleichen, einseitig eingespannten Kragbalken bei der Modellierung mit unterschiedlichen Elementen. Als Referenz dienen hier die Balkenelemente, die das Problem an ihren Knotenpunkten analytisch exakt abbilden.

Zum Abschluss ([Abschnitt 7.10](#)) wird noch ein Beispiel zur Eigenfrequenz- und Modalanalyse bei einer an zwei Seiten eingespannte, ebenen Schale beschrieben.

Die [Eingabedateien zu den Beispielen](#) befinden sich in [Anhang A](#).

### 7.1 Fachwerk aus 2D-Stabelementen

Das in [Abbildung 7.1](#) gezeigte Fachwerk wird an Knoten 4 mit einer Kraft  $F_{y4} = -1000N$  belastet. Der Querschnitt aller Stäbe beträgt  $A = 50mm^2$  und der E-Modul ist  $E = 70000N/mm^2$ .

Nach der statischen Analyse ergibt sich eine Verschiebung des Knotens 4 in x-Richtung von  $u = 0.2544$  und in y-Richtung von  $v = -0.4826$ . Beide Werte stimmen mit den Ergebnissen der Berechnung mit FEAP aus [\[Rei+09\]](#) überein.

[Abbildung 7.1](#) zeigt das Original und das unter der Last verformte Fachwerk. Die Gesamtverschiebung an jedem Punkt ist farbig aufgetragen.

Die zugehörige Eingabedatei [Stab2D.in](#) befindet sich in [Abschnitt A.1](#).

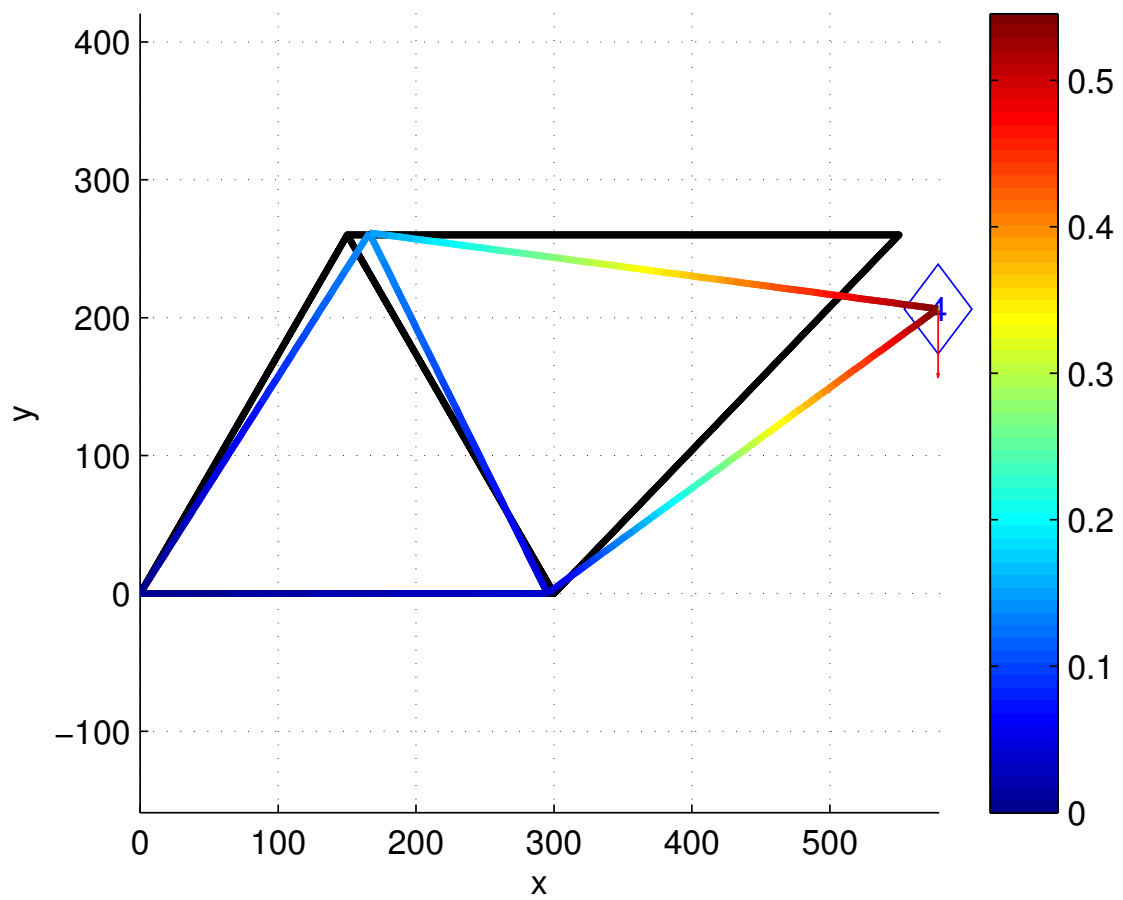


Abbildung 7.1: 2D-Fachwerk

## 7.2 Fachwerk aus 3D-Stabelementen

Das Fachwerk aus 3D-Stabelementen beschreibt eine Pyramide mit viereckigem Grundriss unter einer Zugkraft in der Spitze von  $F_{y5} = 1000N$ . Der Querschnitt der Stäbe beträgt  $A = 100mm^2$  und der E-Modul  $E = 70000N/mm^2$ .

Es ergibt sich eine Verschiebung des Knotens 5 um  $v = 0.075mm$  in Kraftrichtung und eine maximale Spannung in Stablängsrichtung von  $\sigma_x = 3,06N/mm^2$ .

Abbildung 7.2 zeigt das Original und das unter der Last verformte Fachwerk. Die Gesamtverschiebung an jedem Punkt ist farbig aufgetragen.

Die zugehörige Eingabedatei [Stab3D.in](#) befindet sich in [Abschnitt A.2](#).

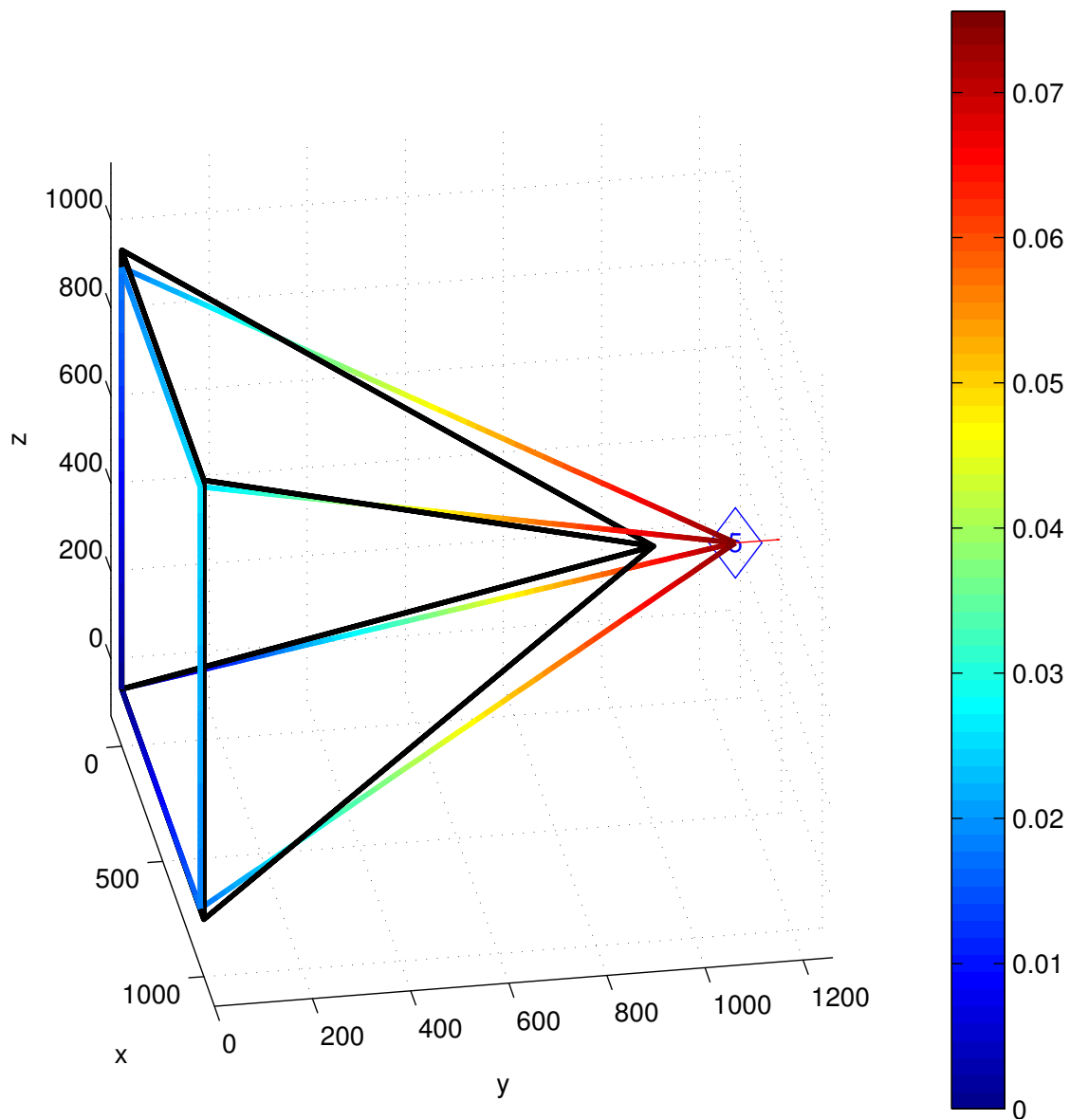


Abbildung 7.2: 3D-Fachwerk

### 7.3 Kragbalken aus Balkenelementen in 2D

Der zu untersuchende Balken ist an der einen Seite fest eingespannt und an der anderen Seite frei. Er besitzt einen quadratischen Querschnitt und die Geometrie- und Materialeigenschaften sind folgendend gegeben:

$$\begin{aligned}l &= 100mm \\b &= h = 10mm \\E &= 70000N/mm^2 \\ \nu &= 0,3\end{aligned}$$

Am freien Ende wird der Balken mit einem Moment  $M = 10000Nmm$  belastet.

Mit diesen Daten ergibt sich die analytische Lösung in y-Richtung zu:

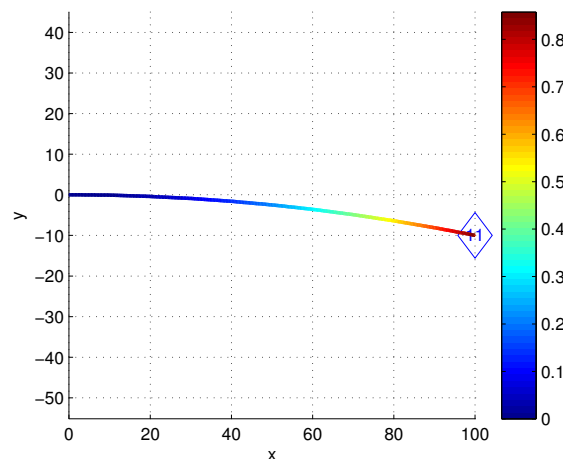
$$v = -\frac{Ml^2}{2EI_z} = -0,857mm$$

Für die FE-Analyse wird der Balken über seine Länge in 10 äquidistante Elemente aufgeteilt. Im 2D-Analyserraum entspricht dies 30 ungefesselten Freiheitsgraden.

Es ergibt sich eine Verschiebung in y-Richtung am freien Balkenende von  $v = -0,857mm$ , welche der analytischen Lösung entspricht.

Abbildung 7.3 zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Balken2D.in](#) befindet sich in [Abschnitt A.3](#).



**Abbildung 7.3:** Kragbalken aus 10 Balkenelementen in 2D

## 7.4 Kragbalken aus Balkenelementen in 3D

Der untersuchte Balken in diesem Beispiel ist identisch mit dem Balken aus [Abschnitt 7.3](#).

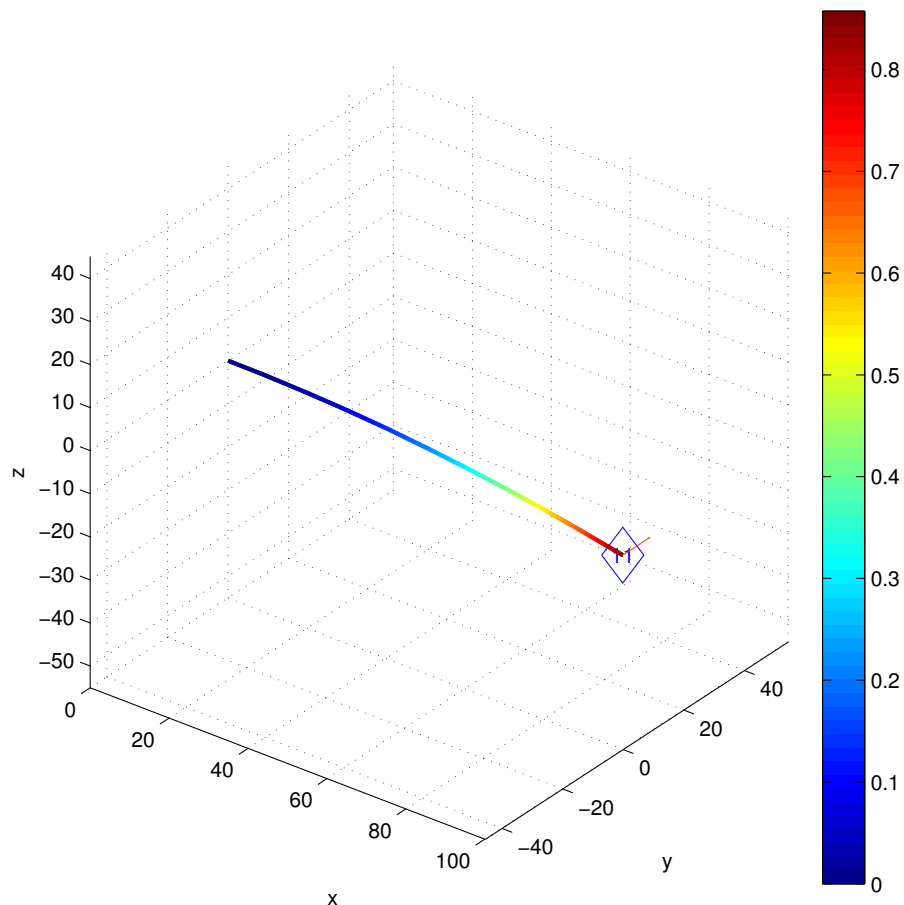
Für die FE-Analyse wird der Balken über seine Länge in 10 äquidistante Elemente aufgeteilt.

Im Gegensatz zu [Abschnitt 7.3](#) wird die Analyse allerdings im 3D-Raum ausgeführt, sodass die Anzahl der Freiheitsgrade auf 60 ansteigt.

Es ergibt sich eine Verschiebung in z-Richtung am freien Balkenende von  $w = -0,857\text{mm}$ , welche der analytischen Lösung entspricht.

[Abbildung 7.4](#) zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Balken3D.in](#) befindet sich in [Abschnitt A.4](#).



**Abbildung 7.4:** Kragbalken aus 10 Balkenelementen in 3D



## 7.5 Kragbalken aus 3-Knoten Scheibenelementen

Auch dieser untersuchte Balken ist identisch mit dem Balken aus [Abschnitt 7.3](#).

Für die FE-Analyse wird der Balken über seine Länge in 10 gleichlange, quadratische Flächen aufgeteilt, die je 2 Dreieckselemente beschreiben.

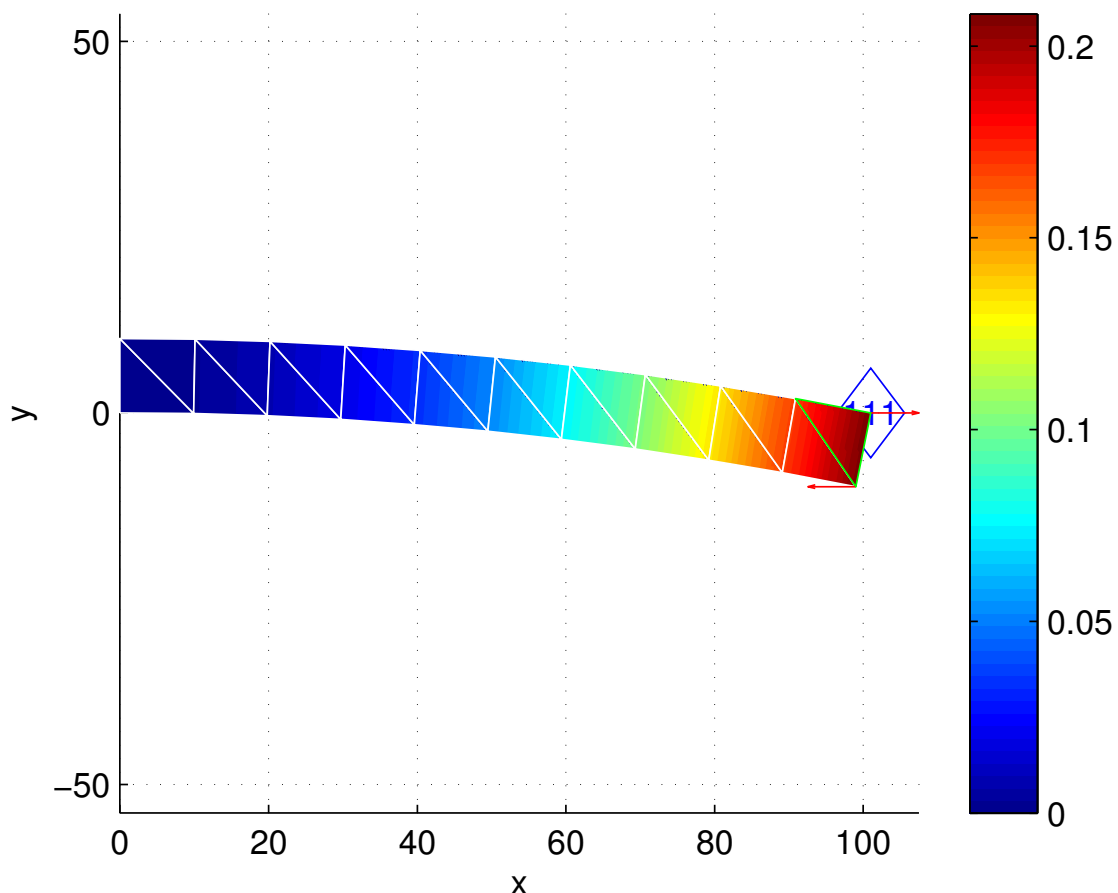
Im Gegensatz zu [Abschnitt 7.3](#) wurden zwar wegen der 2D-Elemente zusätzliche Knoten eingeführt, allerdings besitzen 2D-Kontinuumsselemente nur 2 Freiheitsgrade pro Knoten. Damit steigt die Zahl der unfesselten Freiheitsgrade auf 40.

Es ergibt sich eine gemittelte Verschiebung in y-Richtung am freien Balkenende von  $v = -0,207\text{mm}$ , welche um 75,8% von der analytischen Lösung abweicht.

Da Dreieckselemente nur in der Lage sind konstante Spannungsverläufe über dem Element darzustellen, werden deutlich mehr Elemente benötigt um das Problem näherungsweise zu beschreiben.

[Abbildung 7.5](#) zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Scheibe3K2D.in](#) befindet sich in [Abschnitt A.5](#).



**Abbildung 7.5:** Kragbalken aus 20 3-Knoten-Scheibenelementen

## 7.6 Kragbalken aus 4-Knoten Scheibenelementen

Für die FE-Analyse des Kragbalkens aus 4-Knoten Scheiben wird der Balken über seine Länge in 10 gleichlange, quadratische Elemente aufgeteilt.

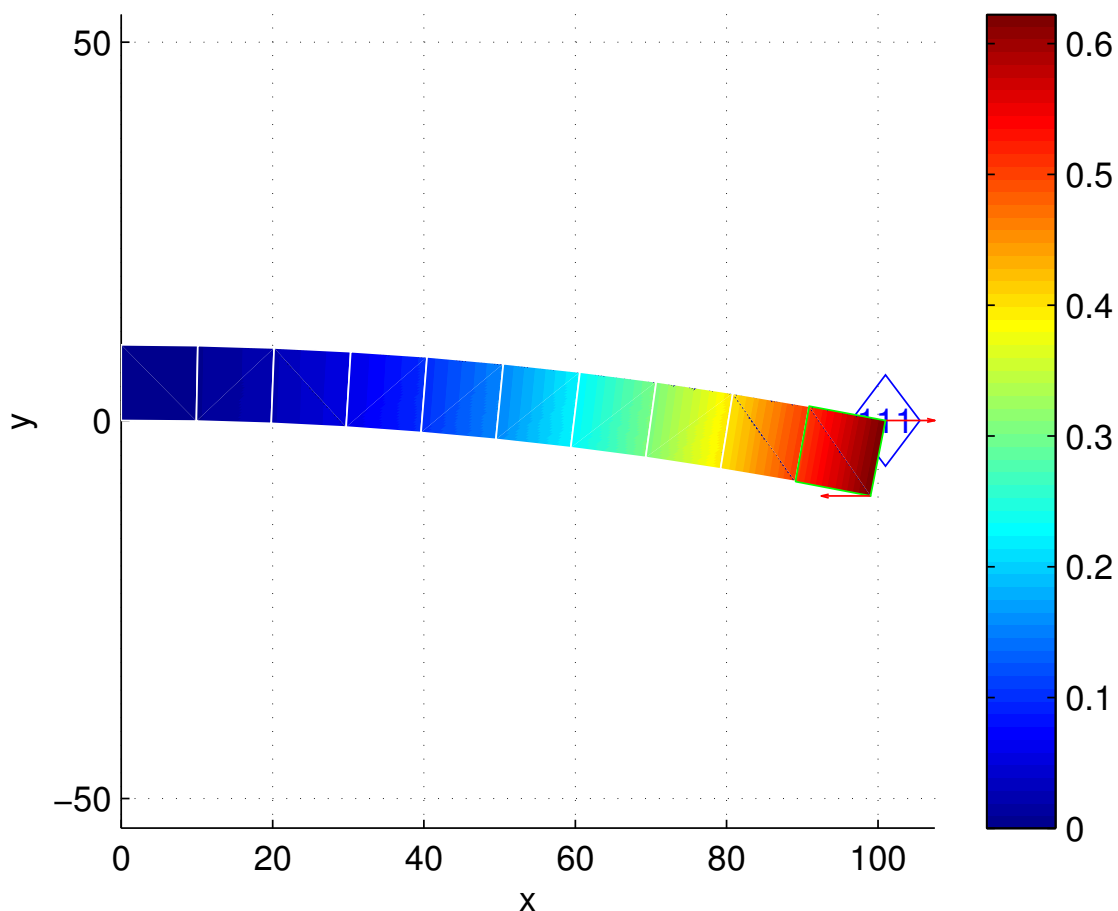
Im Vergleich zu [Abschnitt 7.5](#) wurden keine neuen Knoten eingeführt. Damit bleibt die Zahl der ungefesselten Freiheitsgrade bei 40.

Es ergibt sich eine gemittelte Verschiebung in y-Richtung am freien Balkenende von  $v = -0,619mm$ , welche um 27,8% von der analytischen Lösung abweicht.

Viereckselemente sind zwar wegen ihres linearen Spannungsverlaufes über dem Element genauer als Dreieckselemente, allerdings werden für eine gute Näherung mehr Elemente benötigt.

[Abbildung 7.6](#) zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Scheibe4K2D.in](#) befindet sich in [Abschnitt A.6](#).



**Abbildung 7.6:** Kragbalken aus 10 4-Knoten-Scheibenelementen

## 7.7 Kragbalken aus 8-Knoten Scheibenelementen

Der Balken aus 8-Knoten Scheiben wird über seine Länge in 10 gleichlange, quadratische Elemente aufgeteilt.

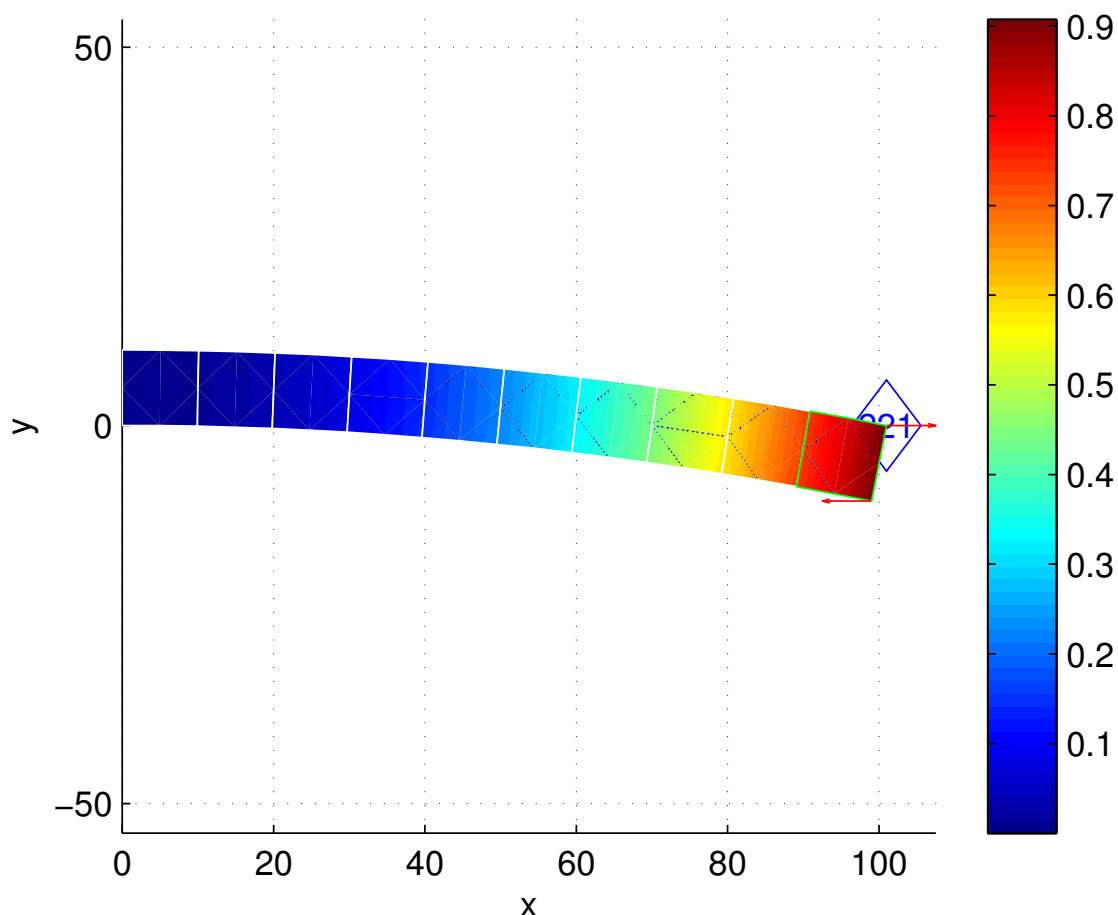
Im Vergleich zu [Abschnitt 7.5](#) und [Abschnitt 7.6](#) wurden an den Rändern der Elemente 30 neuen Knoten eingeführt. Damit erhöht sich die Anzahl der ungefesselten Freiheitsgrade auf 100.

Es ergibt sich eine gemittelte Verschiebung in y-Richtung am freien Balkenende von  $v = -0,902mm$ , welche um 5,3% von der analytischen Lösung abweicht.

8-Knoten-Elemente können wegen ihres quadratischen Spannungsverlaufes über dem Element schon mit wenigen Elementen die Verschiebung am freien Balkenende näherungsweise erfassen. Sie sind damit auf Kosten eines höheren Rechenaufwandes genauer als 3-Knoten und 4-Knoten-Elemente.

[Abbildung 7.7](#) zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Scheibe8K2D.in](#) befindet sich in [Abschnitt A.7](#).



**Abbildung 7.7:** Kragbalken aus 10 8-Knoten-Scheibenelementen

## 7.8 Kragbalken aus Volumenelementen

Der Balken aus Volumenelementen wird über seine Länge in 10 gleichlange, kubische Elemente aufgeteilt.

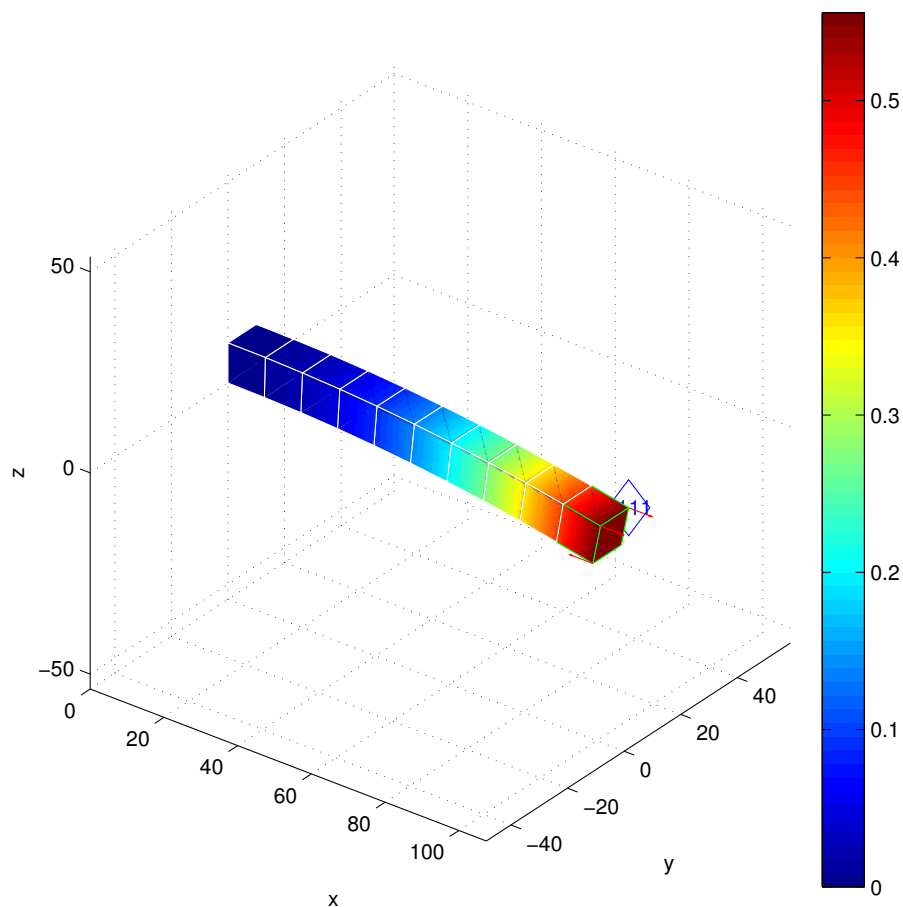
Durch die dreidimensionale Formulierung und den 3D-Analyseraum steigt die Zahl der ungefesselten Freiheitsgrade des Problems auf 120 an.

Am freien Balkenende ergibt sich eine gemittelte Verschiebung in z-Richtung von  $w = -0,553mm$ , welche um 35,4% vom Referenzwert abweicht.

Trotz der hohen Anzahl von Freiheitsgraden und damit hohem Rechenaufwand, ist das Ergebnis noch sehr ungenau. Mehr Elemente und damit gesteigerter Rechenaufwand sind notwendig.

Abbildung 7.8 zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Solid3D.in](#) befindet sich in [Abschnitt A.8](#).



**Abbildung 7.8:** Kragbalken aus 10 Volumenelementen

## 7.9 Kragbalken aus Schalenelementen

Der Kragbalken aus Schalenelementen unter reiner Biegung wird ebenfalls über 10 gleichlange, quadratische Elemente untersucht.

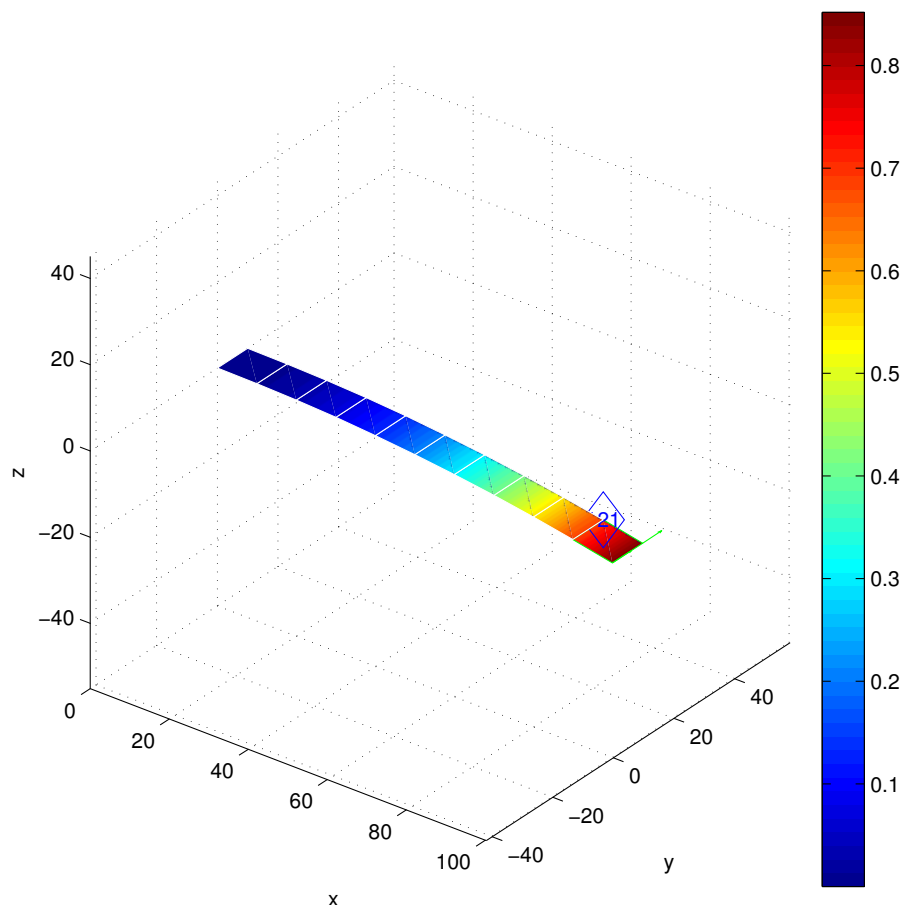
Da Schalenelemente 3D-Strukturelemente sind, besitzt jeder Knoten 6 Freiheitsgrade. Das Problem hat 120 ungefesselte Freiheitsgrade.

Die gemittelte Verschiebung in z-Richtung am freien Balkenende ergibt sich zu  $w = -0,852\text{mm}$ , welche um weniger als 0,01% von der analytischen Lösung abweicht.

Zwar kann die Struktur mit ihrer Dicke nicht mehr als dünnwandig bezeichnet werden, aber trotzdem liefern die Schalenelemente eine gute Lösung. Somit sind Schalenelemente den Volumenelementen aus [Abschnitt 7.8](#) an Effizienz überlegen. Allerdings wird die Dicke nur als Parameter berücksichtigt und nicht wie beim Volumenelement mit dargestellt.

[Abbildung 7.9](#) zeigt den skalierten, verformten Balken und die Gesamtverschiebung an jedem Punkt des Balkens.

Die zugehörige Eingabedatei [Schale3D.in](#) befindet sich in [Abschnitt A.9](#).



**Abbildung 7.9:** Kragbalken aus 10 Schalenelementen

## 7.10 An zwei Seiten eingespannte, ebene Schale

Bei diesem Beispiel handelt es sich um eine quadratische, ebene Schale aus 100 Schalenelementen. Zwei gegenüberliegende Seiten sind fest eingespannt, während die anderen beiden Seiten frei sind.

Die Geometrie und Materialeigenschaften sind im Folgenden gegeben. Zu beachten ist, dass bei dynamischen Analysen sämtliche Parameter im MKS-Einheitensystem eingegeben werden sollten.

$$L = B = 1m$$

$$t = 0.001m$$

$$E = 7 \cdot 10^{10} N/m^2$$

$$\nu = 0,3$$

$$\rho = 2700 kg/m^3$$

Die ersten drei Eigenfrequenzen und deren Moden können [Abbildung 7.10](#) entnommen werden.

Die zugehörige Eingabedatei [EFSchale3D.in](#) befindet sich in [Abschnitt A.10](#).

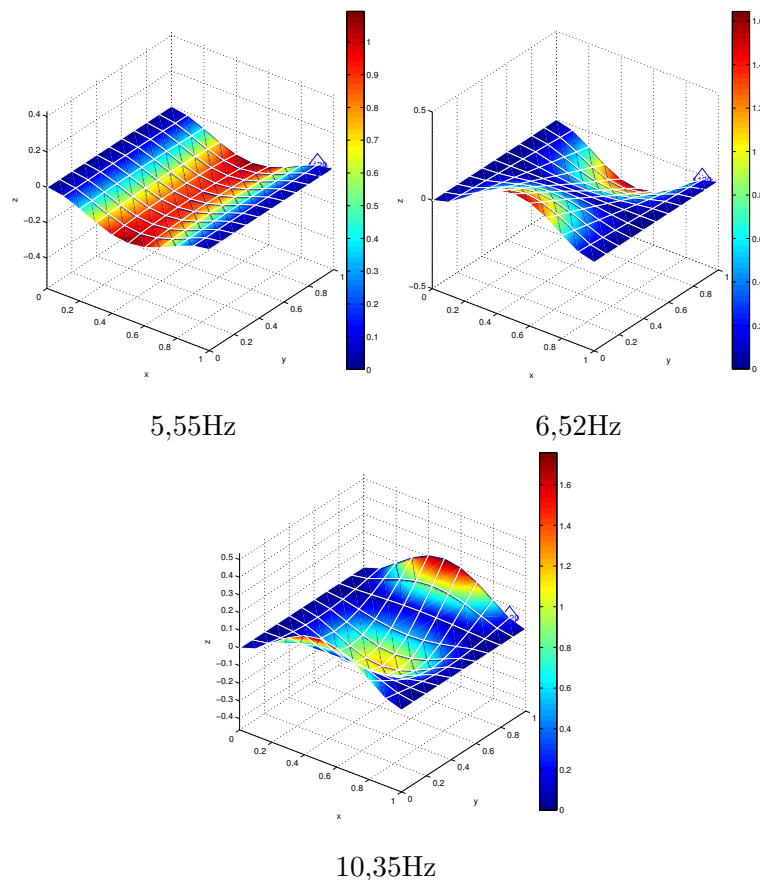


Abbildung 7.10: An zwei Seiten gelagerte, ebene Schale

---

# Teil II

## Entwicklerhandbuch

## 8. Funktionsbeschreibung

In diesem Abschnitt werden alle selbst entwickelten und überarbeiteten Funktionen beschrieben. Einige weitere verwendete Funktionen können im [Anhang C](#) gefunden werden.

### 8.1 Hauptprozedur

#### 8.1.1 main.m

##### Syntax

<code>[ fName ] = main( problemName )</code>
--

##### Beschreibung

Führt ilbFE aus um das ausgewählte Problem zu lösen.

##### Parameter

###### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
problemName	( <i>str</i> )	Name der Eingabedatei (mit Endung: .in)

###### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fName	( <i>str</i> )	Name der Ausgabedatei (mit Endung: .out)



## 8.2 Elemente

### 8.2.1 beam2s.m

#### Syntax

 $[eS] = \text{beam2s}(eX, eY, eP, eD)$ 

#### Beschreibung

Berechnet die maximalen Längsspannung eines Balkenelementes mit Stabanteil und 2 Knoten in 2D.

Die Biege- und Normalkraftanteile der Längsspannung werden so überlagert, dass der Maximalwert (bei  $\pm z_{max}$ ) für den Querschnitt ausgegeben wird.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eX	$\begin{bmatrix} x_1 & x_2 \end{bmatrix}$	X-Koordinate der Element-Knoten
eY	$\begin{bmatrix} y_1 & y_2 \end{bmatrix}$	Y-Koordinate der Element-Knoten
eP	$\begin{bmatrix} E & A & I & z_{max} \end{bmatrix}$	Vektor der derzeitigen Elementeigenschaften <i>E</i> : E-Modul <i>A</i> : Querschnittsfläche <i>I</i> : Flächenträgheitsmoment <i>z<sub>max</sub></i> : Randfaserabstand
eD	$\begin{bmatrix} u_1 & u_2 & \cdots & u_6 \end{bmatrix}$	Vektor der Element-Verschiebungen

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eS	$\begin{bmatrix} \sigma_{x1} \\ \sigma_{x2} \end{bmatrix}$	Vektor der Element-Spannungen

## 8.2.2 beam3s.m

### Syntax

 $[eS] = \text{beam3s}(eX, eY, eZ, eP, eD)$ 

### Beschreibung

Berechnet die maximalen Längsspannung eines Balkenelementes mit Stab- und Torsions-Stabanteil und mit 2 Knoten in 3D.

Die Biege- und Normalkraftanteile der Längsspannung werden so überlagert, dass der Maximalwert (bei  $(\pm y_{max} | \pm z_{max})$ ) für den Querschnitt ausgegeben wird.

### Parameter

#### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eX	$\begin{bmatrix} x_1 & x_2 \end{bmatrix}$	X-Koordinate der Element-Knoten
eY	$\begin{bmatrix} y_1 & y_2 \end{bmatrix}$	Y-Koordinate der Element-Knoten
eZ	$\begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix}$	Z-Koordinate der Element-Knoten
eP	$\begin{bmatrix} E & G & A & I_y & I_z & I_T & z_{max} & y_{max} \end{bmatrix}$	<p>Vektor der derzeitigen Elementeigenschaften</p> <p><math>E</math>: E-Modul</p> <p><math>G</math>: Schubmodul</p> <p><math>A</math>: Querschnittsfläche</p> <p><math>I_y</math>: Flächenträgheitsmoment um y-Achse</p> <p><math>I_z</math>: Flächenträgheitsmoment um z-Achse</p> <p><math>I_T</math>: Torsionsflächenträgheitsmoment</p> <p><math>z_{max}</math>: Randfaserabstand z-Richtung</p> <p><math>y_{max}</math>: Randfaserabstand y-Richtung</p>
e0	$\begin{bmatrix} x_z & y_z & z_z \end{bmatrix}$	Orientierung der lokalen Z-Achse (nur 3D-Balken-Elemente)
eD	$\begin{bmatrix} u_1 & u_2 & \cdots & u_{12} \end{bmatrix}$	Vektor der Element-Verschiebungen

# Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eS		Vektor der Element-Spannungen
	$\begin{bmatrix} \sigma_{x1} & 0 & 0 & 0 & \tau_{yz1} & 0 \\ \sigma_{x2} & 0 & 0 & 0 & \tau_{yz2} & 0 \end{bmatrix}$	

### 8.2.3 shelli4e.m

#### Syntax

```
[Ke] = shelli4e(eX, eY, eZ, eP, D, eQ)
[Ke, fe] = shelli4e(eX, eY, eZ, eP, D, eQ)
```

#### Beschreibung

Berechnet die Steifigkeitsmatrix eines isoparametrischen Schalenelementes mit 4 Knoten in 3D.

Die Integrationsordnung des Schubanteils ist reduziert um Schubsperrern zu verhindern.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eX	$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$	X-Koordinate der Element-Knoten
eY	$\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}$	Y-Koordinate der Element-Knoten
eZ	$\begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix}$	Z-Koordinate der Element-Knoten
eP	$\begin{bmatrix} t & ir \end{bmatrix}$	Vektor der derzeitigen Elementeigenschaften <i>t</i> : Dicke der Schale <i>ir</i> : Integrationsordnung
D	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix
eQ	$\begin{bmatrix} b_x & b_y \end{bmatrix}$	Verteilte Flächenlast

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
Ke	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{24 \times 24}$	Globale Steifigkeitsmatrix des Elements
fe	$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}_{24 \times 1}$	Äquivalenter Lastvektor

## 8.2.4 shelli4s.m

### Syntax

 $[eS, eT] = \text{shelli4s}(eX, eY, eZ, eP, D, eD)$ 

### Beschreibung

Berechnet die Normal- und Schubspannungen eines isoparametrischen Schalenelementes mit 4 Knoten in 3D.

Die Spannungen werden in den Eckpunkten berechnet.

### Parameter

#### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eX	$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$	X-Koordinate der Element-Knoten
eY	$\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}$	Y-Koordinate der Element-Knoten
eZ	$\begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix}$	Z-Koordinate der Element-Knoten
eP	$\begin{bmatrix} t & ir \end{bmatrix}$	Vektor der derzeitigen Elementeigenschaften <i>t</i> : Dicke der Schale <i>ir</i> : Integrationsordnung
D	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix
eD	$\begin{bmatrix} u_1 & u_2 & \cdots & u_{24} \end{bmatrix}$	Vektor der Element-Verschiebungen

#### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eS		Vektor der Element-Spannungen
	$\begin{bmatrix} \sigma_{x1} & \sigma_{y1} & (\sigma_{z1}) & \tau_{xy1} & \tau_{yz1} & \tau_{zx1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_{x4} & \sigma_{y4} & (\sigma_{z4}) & \tau_{xy4} & \tau_{yz4} & \tau_{zx4} \end{bmatrix}$	
eT		Vektor der Element-Ver"-schie"-bungen
	$\begin{bmatrix} u_1 & v_1 & w_1 & \varphi_{xy1} & \varphi_{yz1} & \varphi_{zx1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_4 & v_4 & w_4 & \varphi_{xy4} & \varphi_{yz4} & \varphi_{zx4} \end{bmatrix}$	

### 8.2.5 shelli4d.m

#### Syntax

```
[Ke, Me, Ce] = shelli4d(eX, eY, eZ, eP, D)
[Ke, Me] = shelli4d(eX, eY, eZ, eP, D)
```

#### Beschreibung

Berechnet die Steifigkeitsmatrix, Massenmatrix und die Dämpfungsmatrix eines isoparametrischen Schalenelementes mit 4 Knoten in 3D.

Die Integrationsordnung des Schubanteils ist reduziert um Schubsperrern zu verhindern.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eX	$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$	X-Koordinate der Element-Knoten
eY	$\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}$	Y-Koordinate der Element-Knoten
eZ	$\begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix}$	Z-Koordinate der Element-Knoten
eP	$\begin{bmatrix} t & ir & rho & a & b \end{bmatrix}$	Vektor der derzeitigen Elementeigenschaften <i>t</i> : Dicke der Schale <i>ir</i> : Integrationsordnung <i>rho</i> : Dichte <i>a, b</i> : Dämpfungsfaktoren $\mathbf{c}_e = a\mathbf{k}_e + b\mathbf{m}_e$
D	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
Ke	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{24 \times 24}$	Globale Steifigkeitsmatrix des Elements
Me	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{24 \times 24}$	Globale Massenmatrix des Elements
Ce	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{24 \times 24}$	Globale Dämpfungsmatrix des Elements

## 8.3 Materialgesetze

### 8.3.1 constitutiveTrans.m

#### Syntax

`[DPhi] = constitutiveTrans(D, phi)`

#### Beschreibung

Rotiert die Werkstoffmatrix um einen Winkel.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
D	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix
phi	(double)	Drehwinkel

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
DPhi	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix gedreht um den Winkel <i>phi</i>

### 8.3.2 iso3DConstitutive.m

#### Syntax

 $[D] = \text{iso3DConstitutive}(E, \nu)$ 

#### Beschreibung

Berechnet die lokale Werkstoffmatrix für ein isotropes Material in 3D.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
E	(double)	Elastizitätsmodul
nu	(double)	Querkontraktionszahl

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
D	$\begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix



### 8.3.3 orth3DConstitutive.m

#### Syntax

 $[D] = \text{orth3DConstitutive}(\text{Ep}, \text{Es}, \text{nue}, \text{Gq})$ 

#### Beschreibung

Berechnet die lokale Werkstoffmatrix für ein orthotropes Material in 3D.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
Ep	(double)	Elastizitätsmodul parallel zur Faserrichtung
Es	(double)	Elastizitätsmodul senkrecht zur Faserrichtung
nue	(double)	Querkontraktionszahl
Gq	(double)	Schubmodul quer zur Faserrichtung

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
D	$\begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix

### 8.3.4 rotMatrixStrain.m

#### Syntax

<code>[TEps] = rotMatrixStrain(phi)</code>
--

#### Beschreibung

Berechnet die Rotations-Matrix für Dehnungen für Faserverbund-Schichten.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
phi	(double)	Drehwinkel

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
TEps	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Dehnungs-Rotations-Matrix

### 8.3.5 rotMatrixStress.m

#### Syntax

<code>[TSig] = rotMatrixStress(phi)</code>
--

#### Beschreibung

Berechnet die Rotations-Matrix für Spannungen für Faserverbund-Schichten.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
phi	(double)	Drehwinkel

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
TSig	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Spannungs-Rotations-Matrix

## 8.4 Solver

### 8.4.1 solver1.m

#### Syntax

$$[fName] = \text{solver1}(\text{problemName}, \text{eDof}, K, f, \text{coord}, \text{dof}, \text{prop}, \text{bc}, \text{nen}, \text{eType}, \text{nID})$$

#### Beschreibung

Löst Finite Elemente Probleme in der linearen Statik.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eDof	$\begin{bmatrix} elem & dof_1 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Topologie-Matrix
K	$\begin{bmatrix} \cdot & \cdot \\ \vdots & \vdots \end{bmatrix}_{N \times N}$	Initialisierte Steifigkeitsmatrix
f	$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix}_{N \times 1}$	Lastvektor
coord	$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \end{bmatrix}$	Koordinaten aller Knoten
dof	$\begin{bmatrix} 1 & 2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Indizes der Freiheitsgrade jedes Knotens
prop	( <i>struct</i> )	Element-Eigenschaften
bc	$\begin{bmatrix} dof & U \\ \vdots & \vdots \end{bmatrix}$	Randbedingungen
nen	( <i>int</i> )	Maximale Anzahl von Knoten pro Element
eType	$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix}_{N_e \times 1}$	Vektor der Elementtypen

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fName	( <i>str</i> )	Name der Ausgabedatei (mit Endung: .out)

## 8.4.2 solver2.m

### Syntax

```
[fName] = solver2(problemName, eDof, K, M, coord, dof, prop, bc,  
nen, eType, nID, k)
```

### Beschreibung

Berechnet Eigenfrequenzen und modale Knotenverschiebungen für ein Finite Elemente Problem.

### Parameter

#### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eDof	$\begin{bmatrix} elem & dof_1 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Topologie-Matrix
K	$\begin{bmatrix} \ddots \end{bmatrix}_{N \times N}$	Initialisierte Steifigkeitsmatrix
M	$\begin{bmatrix} \ddots \end{bmatrix}_{N \times N}$	Initialisierte Massenmatrix
coord	$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \end{bmatrix}$	Koordinaten aller Knoten
dof	$\begin{bmatrix} 1 & 2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Indizes der Freiheitsgrade jedes Knotens
prop	<i>(struct)</i>	Element-Eigenschaften
bc	$\begin{bmatrix} dof & U \\ \vdots & \vdots \end{bmatrix}$	Randbedingungen
nen	<i>(int)</i>	Maximale Anzahl von Knoten pro Element
eType	$\begin{bmatrix} \vdots \end{bmatrix}_{N_e \times 1}$	Vektor der Elementtypen
k	<i>(int)</i>	Maximale Anzahl der Iterationsschritte

#### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fName	<i>(str)</i>	Name der Ausgabedatei (mit Endung: .out)

### 8.4.3 coordXtr.m

#### Syntax

```
[eX, eY, eZ] = coordXtr(eDof, coord, dof, nen)
[eX, eY] = coordXtr(eDof, coord, dof, nen)
```

#### Beschreibung

Extrahiert Daten der Knotenkoordinaten mehrerer Elemente aus der globalen Knotenkoordinaten-Matrix.

Die Anzahl der Knoten und Freiheitsgrade kann zwischen den Elementen variieren.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eDof	$\begin{bmatrix} elem & dof_1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Topologie-Matrix
coord	$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \end{bmatrix}$	Koordinaten aller Knoten
dof	$\begin{bmatrix} 1 & 2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Indizes der Freiheitsgrade jedes Knotens
nen	(int)	Maximale Anzahl von Knoten pro Element

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eX	$\begin{bmatrix} x_1 & x_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	X-Koordinate der Element-Knoten
eY	$\begin{bmatrix} y_1 & y_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Y-Koordinate der Element-Knoten
eZ	$\begin{bmatrix} z_1 & z_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Z-Koordinate der Element-Knoten

### 8.4.4 extractProperties.m

#### Syntax

```
[eP, D, e0] = extractProperties(eType, i, prop)
[eP, D] = extractProperties(eType, i, prop)
[eP, ~, e0] = extractProperties(eType, i, prop)
[eP] = extractProperties(eType, i, prop)
```

#### Beschreibung

Extrahiert die Daten der Eigenschaften aus *prop* und bereitet sie für die Element-Funktionen auf.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eType	( <i>int</i> )	Element-Typ
i	( <i>int</i> )	Index des derzeitigen Elementes
prop	( <i>struct</i> )	Element-Eigenschaften

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eP	$\begin{bmatrix} \dots \end{bmatrix}_{1 \times n}$	Vektor der derzeitigen Elementeigenschaften
D	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{6 \times 6}$	Werkstoff-Matrix
e0	$\begin{bmatrix} x_z & y_z & z_z \end{bmatrix}$	Orientierung der lokalen Z-Achse (nur 3D-Balken-Elemente)

### 8.4.5 solveig.m

#### Syntax

```
[ef, B] = solveig(K, M, bc)
[ef, B] = solveig(K, M, [])
[ef, B] = solveig(K, M, [], k)
[ef, B] = solveig(K, M, bc, k)
```

#### Beschreibung

Berechnet die Eigenfrequenzen und modalen (normierten) Knotenverschiebungen der zugehörigen Eigenfrequenzen.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
K	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{N \times N}$	Globale Steifigkeitsmatrix der Struktur
M	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{N \times N}$	Globale Massenmatrix der Struktur
bc	$\begin{bmatrix} dof & U \\ \vdots & \vdots \end{bmatrix}$	Randbedingungen
k	(int)	Gesuchte Anzahl an Eigenfrequenzen

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
ef	$\begin{bmatrix} \cdot \\ \vdots \end{bmatrix}_{k \times 1}$	Vektor der Eigenfrequenzen
B	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{N \times k}$	Eigenvektor / Modale Knotenverschiebung zu jeder Eigenfrequenz



## 8.5 Präprozessor

### 8.5.1 preprocess.m

#### Syntax

```
[eDof, K, M, C, f, coord, dof, prop, bc, nen, sType, eType, nID,  
k, err, title] = preprocess(problemName)
```

#### Beschreibung

Liest die Eingabe-Informationen aus einer Eingabedatei und bereitet die Daten für das Kern-Programm auf.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
problemName	(str)	Name der Eingabedatei (mit Endung: .in)

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eDof	$\begin{bmatrix} elem & dof_1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Topologie-Matrix
K	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{N \times N}$	Initialisierte Steifigkeitsmatrix
M	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{N \times N}$	Initialisierte Massenmatrix
C	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_{N \times N}$	Initialisierte Dämpfungsmatrix
f	$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix}_{N \times 1}$	Lastvektor
coord	$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \end{bmatrix}$	Koordinaten aller Knoten
dof	$\begin{bmatrix} 1 & 2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Indizes der Freiheitsgrade jedes Knotens
prop	(struct)	Element-Eigenschaften
bc	$\begin{bmatrix} dof & U \\ \vdots & \vdots \end{bmatrix}$	Randbedingungen
nen	(int)	Maximale Anzahl von Knoten pro Element

sType	( <i>int</i> )	Solver-Typ
		1 = Statische, lineare Analyse
		2 = Eigenfrequenz- und
		Modalanalyse
eType	$\begin{bmatrix} \vdots \end{bmatrix}_{N_e \times 1}$	Vektor der Elementtypen
nID	$\begin{bmatrix} \vdots \end{bmatrix}_{N_n \times 1}$	Vektor der Knoten-IDs
k	( <i>int</i> )	Maximale Anzahl der Iterations- schritte
err	( <i>double</i> )	Fehlerschranke
title	( <i>str</i> )	Titel des Problems

### 8.5.2 readInput.m

#### Syntax

`[input] = readInput(problemName)`

#### Beschreibung

Liest Daten aus der Eingabedatei ein und speichert diese in einer *struct* mit Feldern benannt nach den Spalten-Headern.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
problemName	( <i>str</i> )	Name der Eingabedatei (mit Endung: .in)

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
input	( <i>struct</i> )	Eingabe-Daten, gespeichert in Unter- <i>structs</i> mit Feldnamen passend zu Spalten-Headern in der Eingabe-Datei

### 8.5.3 prepareInput.m

#### Syntax

```
[eDof, K, M, C, f, coord, dof, prop, bc, nen, sType eType, nID, k
, err, title] = prepareInput(input)
```

#### Beschreibung

Bereitet die Eingabedaten für das Kern-Programm auf.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
input	( <i>struct</i> )	Eingabe-Daten, gespeichert in Unter- <i>structs</i> mit Feldnamen passend zu Spalten-Headern in der Eingabe-Datei

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
eDof	$\begin{bmatrix} elem & dof_1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Topologie-Matrix
K	$\begin{bmatrix} \cdot & \cdot \\ \vdots & \vdots \end{bmatrix}_{N \times N}$	Initialisierte Steifigkeitsmatrix
M	$\begin{bmatrix} \cdot & \cdot \\ \vdots & \vdots \end{bmatrix}_{N \times N}$	Initialisierte Massenmatrix
C	$\begin{bmatrix} \cdot & \cdot \\ \vdots & \vdots \end{bmatrix}_{N \times N}$	Initialisierte Dämpfungsmatrix
f	$\begin{bmatrix} \cdot \\ \vdots \end{bmatrix}_{N \times 1}$	Lastvektor
coord	$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \end{bmatrix}$	Koordinaten aller Knoten
dof	$\begin{bmatrix} 1 & 2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Indizes der Freiheitsgrade jedes Knotens
prop	( <i>struct</i> )	Element-Eigenschaften
bc	$\begin{bmatrix} dof & U \\ \vdots & \vdots \end{bmatrix}$	Randbedingungen
nen	( <i>int</i> )	Maximale Anzahl von Knoten pro Element

sType	( <i>int</i> )	Solver-Typ
		1 = Statische, lineare Analyse
		2 = Eigenfrequenz- und
		Modalanalyse
eType	$\begin{bmatrix} \vdots \end{bmatrix}_{N_e \times 1}$	Vektor der Elementtypen
nID	$\begin{bmatrix} \vdots \end{bmatrix}_{N_n \times 1}$	Vektor der Knoten-IDs
k	( <i>int</i> )	Maximale Anzahl der Iterations- schritte
err	( <i>double</i> )	Fehlerschranke
title	( <i>str</i> )	Titel des Problems

## 8.6 Postprozessor

### 8.6.1 postprocess.m

#### Syntax

`[fName] = postprocess(problemName, varargin)`

#### Beschreibung

Sammelt und bereitet die Ergebnisse auf, um sie in die Ausgabe-Datei zu schreiben.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
problemName	( <i>str</i> )	Name der Eingabedatei (mit Endung: .in)
(varargin)		Variable Anzahl von Parametern, die ausgegeben werden sollen (bekannte Matrix oder beliebige <i>struct</i> )

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fName	( <i>str</i> )	Name der Ausgabedatei (mit Endung: .out)

## 8.6.2 prepareOutput.m

### Syntax

<code>[output] = prepareOutput(output)</code>
---

### Beschreibung

Bereitet die Programm-Variablen für die Ausgabe vor.

### Parameter

#### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
output	( <i>struct</i> )	Ausgabe-Daten

#### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
output	( <i>struct</i> )	Ausgabe-Daten mit Unter- <i>struct Header</i> , die jedem Feld von <i>output</i> , das selbst keine <i>struct</i> ist, Spalten-Header zuordnet

### 8.6.3 writeOutput.m

#### Syntax

<code>[fName] = writeOutput(problemName, output)</code>
---

#### Beschreibung

Schreibt die Ausgabe-Daten in eine Text-Datei.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
problemName	( <i>str</i> )	Name der Eingabedatei (mit Endung: .in)
output	( <i>struct</i> )	Ausgabe-Daten mit Unter- <i>struct</i> <i>Header</i> , die jedem Feld von <i>output</i> , das selbst keine <i>struct</i> ist, Spalten- Header zuordnet

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fName	( <i>str</i> )	Name der Ausgabedatei (mit Endung: .out)



## 8.7 Sonstiges

### 8.7.1 checkFileName.m

#### Syntax

`[ fileName , isFileExist ] = checkFileName( fileName , extension )`

#### Beschreibung

Kontrolliert ob eine Datei im derzeitigen MATLAB-Pfad existiert und überprüft die Endung.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fileName	( <i>str</i> )	Original-Name der Datei (mit oder ohne Endung)
extension	( <i>str</i> )	Geforderte Endung

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
fileName	( <i>str</i> )	Name der Datei (mit geforderter Endung)
isFileExist	( <i>logical</i> )	Wahr, wenn die Datei existiert

### 8.7.2 ID2Index.m

#### Syntax

$$[\text{indexMat}] = \text{ID2Index}(\text{IDMat}, \text{IDs})$$

#### Beschreibung

Ersetzt jeden Wert von  $\text{IDMat}$  mit dem entsprechenden Zeilenindex dieses Wertes in  $\text{IDs}$  und gibt die resultierende Matrix als  $\text{IndexMat}$  aus.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
IDMat	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{n \times m}$	Matrix gefüllt mit IDs
IDs	$\begin{bmatrix} \cdot \\ \vdots \end{bmatrix}_{k \times 1}$	Vektor aller IDs

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
indexMat	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{n \times m}$	Matrix gefüllt mit den entsprechenden Indizes der IDs

### 8.7.3 index2ID.m

#### Syntax

$[IDMat] = \text{Index2ID}(\text{indexMat}, \text{IDs})$

#### Beschreibung

Ersetzt jeden Wert von *IndexMat* mit dem entsprechenden Wert am Zeilenindex von *IDs* und gibt die resultierende Matrix als *IDMat* aus.

#### Parameter

##### Eingabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
indexMat	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{n \times m}$	Matrix gefüllt mit den entsprechenden Indizes der IDs
IDs	$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}_{k \times 1}$	Vektor aller IDs

##### Ausgabe

<i>Name</i>	<i>Typ</i>	<i>Parameterbeschreibung</i>
IDMat	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}_{n \times m}$	Matrix gefüllt mit IDs

## 9. Einbindung Neuer Elemente und Solver

### 9.1 Neue Elemente

Die Implementierung neuer Elemente wird anhand des bereits implementierten Elements, eines [3D - 2 Knoten - Stab](#) mit der ID *123*, exemplarisch erklärt.

Die dazugehörigen Funktionen `bar3e()` und `bar3s()` sollten bereits existieren. Die Funktion `bar3e()` muss die globale Elementsteifigkeitsmatrix `Ke` aus den Zeilenvektoren der Elementkoordinaten `eX`, `eY`, `eZ` und dem Zeilenvektor der Elementigenschaften `eP` liefern. Mit der zusätzlichen Information der Elementverschiebung `eD` muss `bar3s()` dann die Elementspannung `eS` liefern.

#### 9.1.1 Anpassung des Präprozessors

Im Präprozessor `prepareInput()` muss zunächst die Anzahl der Freiheitsgrade pro Knoten auf das Element angepasst werden. So gibt es zum Beispiel bei 2D-Kontinuums Elementen 2 Freiheitsgrade pro Knoten und bei 3D-Struktur Elementen 6 Freiheitsgrade pro Knoten.

Die Matrix `dof` ist eine Matrix mit so vielen Zeilen, wie es Knoten gibt und so vielen Spalten, wie es Freiheitsgrade pro Knoten gibt. Die Einträge zählen Zeilenweise von rechts nach links von 1 hoch.

$$\text{dof} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ \vdots & \vdots & \vdots \end{bmatrix} \quad (9.1)$$

In Gleichung (9.1) ist ein Beispiel für `dof` bei einem 3D - Kontinuumselement, wie dem [3D - 2 Knoten - Stab](#), gegeben.

Zur Implementation wird Stabelement *123* in die `if` Abfrage (vgl. [Abbildung 9.1](#)) eingefügt.

```
%----- Degrees of Freedom -----
% How many Degrees of Freedom per direction (rotational DoF
if any(ismember([223 543], eType))
    % 3D - Structure Elements
    % 2 DoF per direction (translation, rotation) and all
        directions active
    dof = reshape((1 : 2*numel(coord))', [], size(coord, 1))';
    nDim = 3;
    [...]
end
```

**Abbildung 9.1:** Setzen der Freiheitsgrade im Präprozessor

Als nächstes muss die Anzahl der Knoten von Elementen des Typs *123* festgelegt werden. Das geschieht, indem `nens(i)` für alle *i* bei denen der Element-Typ-Vektor `eType` den Typ *123* hat.

Man erhält den Spaltenvektor `nens`, der für jedes Element die Anzahl der Knoten beinhaltet.

```
%----- number of nodes per element -----
nens = zeros(size(eType, 1), 1);
for i = 1 : size(eType, 1)
    switch eType(i)
        [...]
        case {122, 123, 222, 223} % 2 Node Elements
            nens(i) = 2;
        end
    end
end
nen = max(nens);
```

**Abbildung 9.2:** Setzen der Knotenzahl im Präprozessor

In der Topologie Matrix `eDof` müssen nun die Freiheitsgrade der Knoten jedes Elementes einsortiert werden.

Dabei steht jede Zeile für ein Element. Die erste Spalte gibt die Element-ID (`eID`) an und die folgenden Spalten die Nummern der Freiheitsgrade aller Elementknoten. Leere Freiheitsgrade werden mit Nullen aufgefüllt, die im Solver für jedes Element später wieder entfernt werden.

$$\mathbf{eDof} = \begin{bmatrix} eID & x1 & y1 & z1 & phiX1 & phiY1 & phiZ1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad (9.2)$$

Gleichung (9.2) zeigt den Aufbau von `eDof` für 3D-Elemente.

Im Gegensatz zu `dof` sind hier für jede Raumrichtung Verschiebungs und Rotationsfreiheitsgrade vorhanden, selbst wenn sie mit Nullen gefüllt sind und deaktiviert sind.

Für jeden zusätzlichen Knoten kommen also sechs (3D) beziehungsweise drei Spalten zur Matrix `eDof` hinzu.

```

%----- Topology Matrix eDof -----
[... ]
for i = 1 : size(eType, 1)
    [... ]
    switch eType(i)
        case {123} % bar, 2 Node, 3D
            eDof(i,[2 3 4 8 9 10]) = [dof(eIndices(:,1),:), dof(eIndices
                                   (:,2),:)] ;
        [... ]
    end
end
end

```

Abbildung 9.3: Setzen der Element-Topologie im Präprozessor

Die Indizes der Freiheitsgrade werden, wie in [Abbildung 9.3](#), für jeden Knoten  $j$  des Elements mit dem Befehl `dof(eIndices(:,j),:)` ausgegeben. Dazu müssen alle Freiheitsgrade der Element-Knoten in einem Zeilenvektor gesammelt werden (in eckigen Klammern).

Der Zeilenvektor `[2 3 4 8 9 10]` in `eDof(i,[2 3 4 8 9 10])` sortiert die Freiheitsgrade der Knoten an die entsprechenden Stellen in der Matrix `eDof` ein. Dabei muss der Zeilenvektor die gleiche Länge haben, wie die Anzahl der Freiheitsgrade des Elements.

Zuletzt müssen die Element-Eigenschaften aus der Eingabe *struct input* in die Datenbank *struct prop* übertragen werden.

```

%----- Element properties -----
for i = 1 : size(eType, 1)
    switch eType(i)
        case {122, 123} % bar elements
            matIndex = ID2Index(input.Elements.MatID(i), input.Materials.ID);
            propIndex = ID2Index(input.Elements.PropID(i), input.Properties.ID)
                        ;
            prop.Ep(i) = input.Materials.Ep(matIndex);
            prop.A(i) = input.Properties.A(propIndex);
        [... ]
    end
end
end

```

Abbildung 9.4: Setzen der Element-Eigenschaften im Präprozessor

In diesem Schritt ([Abbildung 9.4](#)) müssen für jedes Element (hier alle Elemente des Typ: *123*) der Material- und Geometrieindex ausgelesen werden (`matIndex`, `propIndex`) und dann die korrespondierenden Material- und Geometrie-Eigenschaften in die *struct prop* an die Stelle  $i$  des Elements geschrieben werden.

### 9.1.2 Anpassung der Elementeigenschaften-Ausleseprozedur

In der Elementeigenschaften-Ausleseprozedur `extractProperties()` müssen die Eigenschaften aus der Datenbank `Prop` ausgelesen, gegebenenfalls die Werkstoffmatrix `D` berechnet und alle weiteren Eigenschaften im Eigenschafts-Zeilenvektor `eP` gespeichert werden.

```
switch eType
[... ]
case {123} % bar, 2 Node, 3D
    eP = [prop.Ep(i) prop.A(i)];
[... ]
end
```

**Abbildung 9.5:** Anpassung der Elementeigenschaften-Ausleseprozedur

Der Eingabeparameter `i` der Funktion `extractProperties()` gibt dabei den index des derzeitigen Elementes an.

### 9.1.3 Anpassung des Löser

Im Löser müssen abschließend in den Vorlauf- und Nachlaufrechnungsschleifen die Elementeigenschaften-Ausleseprozedur `extractproperties()` und entsprechend die Steifigkeitsmatrix (`bar3e()`) oder die Spannungen (`bar3s()`) berechnet werden.

```
%———— Create element stiffness matrices Ke and assemble into K —
N = size(eType, 1);
for i = 1 : N
    switch eType(i)
    [...]
    case {123} % bar, 2 Node, 3D
        eP = extractProperties(eType(i), i, prop);
        Ke = bar3e(eX(i,:), eY(i,:), eZ(i,:), eP);
        K = assem(eDof(i,:), K, Ke);
    [...]
end
end
```

**Abbildung 9.6:** Anpassung der Vorlaufrechnung im Löser

Über die Funktion `assem()` wird die Matrix in die globale Steifigkeitsmatrix einsortiert. Dabei werden die Freiheitsgrade in `eDof(i,:)`, die Null sind gestrichen.

```
%----- Element stresses -----
[...]
```

```
for i = 1 : N
    switch eType(i)
        [...]
```

```
    case {123} % bar, 2 Node, 3D
        eP = extractProperties(eType(i), i, prop);
        bar3s(eX(i,:), eY(i,:), eZ(i,:), eP, eD(i,:));
        eStmp = bar3s(eX(i,:), eY(i,:), eZ(i,:), eP, eD(i,:));
        [~, j] = ismember([eDof(i,2) eDof(i,8)], dof); % find node indices
                of element nodes
        eS(nen*i-1 : nen*i, 1:7) = [eDof(i,1) eType(i) 1 nID(j(1)) eStmp 0 0;...
                                   eDof(i,1) eType(i) 2 nID(j(2)) eStmp 0 0];

    [...]
```

```
end
end
```

**Abbildung 9.7:** Anpassung der Nachlaufrechnung im Löser

Die Ausgabe `postprocess()` erwartet beim *Linearen Statik Löser* die Knotenverschiebungen `nDisp` und die Elementspannungen `eStress` und beim *Linearen Modal- und Eigenfrequenz Löser* die Eigenfrequenzen `mFreq` und die modalen Knotenverschiebungen `mDisp`.



## 9.2 Neue Solver

Neue Solver werden in der Hauptprozedur `main()` ([Abbildung 9.8](#)) eingebunden. Die Eingabekarten müssen gegebenenfalls um Variablen erweitert werden, die dann in `prepareInput()` weiterverarbeitet werden.

Die Ausgabe erwartet eine beliebige Anzahl von Parametern, die entweder vom Datentyp *struct* sind (und somit Spaltenüberschriften für Ausgabe-Header mitbringen) oder in `prepareOutput()` bekannt sind und dort eine Header-Zeile zugeordnet bekommen.

```
switch sType
case {1}
    % static linear analysis
    fName = solver1(problemName, eDof, K, f, coord, dof, prop, bc,
                    nen, eType, nID);
case {2}
    % eigen frequency and modal analysis
    fName = solver2(problemName, eDof, K, M, coord, dof, prop, bc,
                    nen, eType, nID, k);
case {3}
    % dynamic response through iteration in time domain
    fName = solver3(problemName, eDof, K, M, C, coord, dof, prop,
                    bc, nen, eType, nID, k, err);
end
```

**Abbildung 9.8:** Anpassung der Hauptprozedur

## 10. Zusammenfassung und Fazit

Zusammenfassend ist zu sagen, dass das Programm `ilbFE` ein gutes Beispiel für einen schlanken, aber leistungsfähigen FE-Code ist. Die Wahl von MATLAB als Programmiersprache erlaubt es die Elemente und Löser nah an mathematischen Formulierungen zu implementieren, wodurch der Übergang von der Theorie zur Implementierung verständlich ist.

Im Gegensatz zu kommerziellen FE-Programmen, die keinen tieferen Einblick in ihren Code erlauben und Probleme im Black-Box-Verfahren lösen, erhält der Nutzer ein eingehendes Verständnis des Themenbereichs durch den Einblick in die internen Abläufe eines FE-Programms.

Die Unterschiede zwischen Schritten, die auf Grund der Formulierung der Finite-Elemente-Methode notwendig sind, sowie numerisch oder programmiertechnisch sinnvollen Schritten werden aufgezeigt.

Mit der Beschreibung der Löser und Elemente wird dem Nutzer eine Übersicht über die Anwendung, Anforderungen und Leistungsfähigkeit von Finiten Elementen gegeben, die auf hoch entwickelte, kommerzielle Elemente und Löser übertragen werden können.

Auch wenn das Hauptaugenmerk auf dem Grundgerüst und der Modularität des Codes liegt, sind die implementierten Elemente leistungsfähig und ergeben in Tests gute Ergebnisse.

Die abschließende Dokumentation der Eingabe und Ausgabe im Benutzerhandbuch bietet viele Beispiele und Erläuterungen zum eigenständigen Erstellen von Eingabe- und Ausgabedateien und bietet einen Rahmen für die Weiterentwicklung.

Weiterführend würde dem Nutzer eine Anpassung und Dokumentation des getrennt entwickelten Prä- und Postprozessor GUI auf `ilbFE` Vereinfachungen in der Eingabe von Strukturen mit sich bringen. Die Unterstützung von verschiedenen Elementen und eine Erweiterung der Netzgenerierung müsste dazu entwickelt werden.

Zur schnelleren Struktureingabe wäre ein Kopplung einer **Structure** Karte mit dem Netzgenerator zur internen Erstellung einer Struktur aus aufeinander folgenden, gleichartigen Elementen ebenfalls sinnvoll.

Für Entwickler gibt die Funktionsbeschreibung auf dem Programmablauf aufbauend einen ausführlichen Überblick über die genutzten Funktionen und die Zusammenhänge im Programm.

Ein besonderes Augenmerk wurde auf die anschauliche Dokumentation der Parameter gelegt um zum Beispiel bei Matrizen die Bedeutung der Spalten kompakt und verständlich darzustellen.

Abschließend gibt das Entwicklerhandbuch eine Anleitung zur Einbindung von weiteren Elementen und Lösern. Für die Weiterentwicklung wurde im Programmcode darauf geachtet, dass am Grundgerüst möglich wenig geändert werden muss um neue Elemente und Solver zu integrieren.

Aus Sicht der Finite Elemente Methode wäre eine Weiterentwicklung von einigen Elementen bezüglich Unterstützung von Offsets vorstellbar. So unterstützt der Modulare Präprozessor das Einlesen solcher Elementeneigenschaften, allerdings müssten entsprechende Balkenelemente noch auf die Eingabedaten zugreifen.

Die Unterstützung von orthotropen Werkstoffen könnte von den Schalenelementen auf andere Elemente übertragen werden und eine Option für Laminare mit mehreren Schichten könnte hinzugefügt werden.

Passend hierzu könnte die Eingabe und Verarbeitung von Materialdaten zwischen den Elementen vereinheitlicht werden, sodass verschiedene Elemente nicht die gleichen Daten unter verschiedenen Eingabeparameterdaten anfordern.

Der große Themenbereich der iterativen Verfahren, in Hinblick auf die nichtlineare Rechnung und die Dynamik könnte über neue Solver und Elemente implementiert werden. Hierbei kann auf den Solver der Eigenwert- und Modalanalyse und die [3D - 4 Knoten - Schale](#) ([Unterabschnitt 5.6.1](#)) aufgebaut werden, welche die benötigten Parameter für dynamische Probleme mit Dämpfung bereits unterstützt.

Seitdem MATLAB neben Linearen Gleichungssystemen durch seine Kopplung mit MAPLE auch Analysis-Rechnungen beherrscht, wären auch generische Elemente mit Eingabe von Differential-Operatormatrix und Formfunktionen denkbar.

Vorbereitend darauf könnte die numerische Integration von der Formulierung der Elementsteifigkeitsmatrix getrennt werden, was für den Nicht-Analysis-Teil von MATLAB noch vorteilhaft war.

In jedem Fall bietet ilbFE, gerade durch seine Modularität und funktionale Trennung von Präprozessor, Löser, Elementen und Postprozessor einen soliden Grundstein für das Verständnis der Finiten Elemente Methode.

Die implementierten Elemente können viele Strukturen aus dem Leichtbau gut simulieren und die Ergebnisse stimmen mit den analytischen und kommerziellen FE-Rechnungen gut überein.

Durch die Programmierung und auch durch die Dokumentation konnte ich einen tieferen Einblick in die Hintergründe und Probleme der FEM gewinnen und letztere anhand der Vorlesung, weitergehender Literatur, meinem Betreuer und eigener Ideen lösen.

Mein Verständnis, aber auch meine Begeisterung für die Finite Elemente Analyse sind durch die Arbeit an ilbFE gewachsen und haben meinen akademischen Werdegang und meine fachliche Spezialisierung maßgeblich beeinflusst.

# Literatur

- [Aus+04] P.-E. AUSTRELL u. a. *CALFEM - A Finite Element Toolbox*. Lund: Division of Structural Mechanics, LTH, Sweden, 2004.
- [Bat02] K. J. BATHE. *Finite-Elemente-Methoden*. 2. Aufl. Berlin [u.a.]: Springer, 2002.
- [Rei+09] H.-G. REIMERDES u. a. *Finite Berechnungsmethoden im Leichtbau I*. Vorlesungsskript, Institut für Leichtbau, RWTH Aachen. Aachen, 2009.

# A. Eingebodateien zu den Beispielen

## A.1 Stab2D.in

Title Fachwerk aus 2D-Stabelementen

H Solver	Type
Solver	1

H Nodes	ID	X	Y
Nodes	1	0.0	0.0
Nodes	2	300.0	0.0
Nodes	3	150.0	260.0
Nodes	4	550.0	260.0

H Elements	ID	Type	MatID	PropID	N1	N2
Elements	1	122	1	1	1	2
Elements	2	122	1	1	1	3
Elements	3	122	1	1	2	3
Elements	4	122	1	1	3	4
Elements	5	122	1	1	2	4

H Materials	ID	Ep
Materials	1	70000.0

H Properties	ID	A
Properties	1	50.0

H BC	NodeID	XDir	YDir
BC	1	0	0
BC	2	i	0

H Loads	NodeID	ForceX	ForceY
Loads	4	0.0	-1000.0

## A.2 Stab3D.in

Title Fachwerk aus 3D-Stabelementen

H Solver      Type  
 Solver        1

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	2	1000	0	0
Nodes	3	0	0	1000
Nodes	4	1000	0	1000
Nodes	5	500	1000	500

H Elements	ID	Type	MatID	PropID	N1	N2
Elements	1	123	1	1	1	2
Elements	2	123	1	1	2	4
Elements	3	123	1	1	4	3
Elements	4	123	1	1	3	1
Elements	5	123	1	1	1	5
Elements	6	123	1	1	2	5
Elements	7	123	1	1	3	5
Elements	8	123	1	1	4	5

H Materials ID Ep  
 Materials 1 70000.0

H Properties ID A  
 Properties 1 100

H BC	NodeID	XDir	YDir	ZDir
BC	1	0	0	0
BC	2	i	0	i
BC	3	i	0	i
BC	4	i	0	i

H Loads	NodeID	ForceX	ForceY	ForceZ
Loads	5	0	1000	0

## A.3 Balken2D.in

Title Kragbalken aus 2D-Balkenelementen

H Solver	Type
Solver	1

H Nodes	ID	X	Y
Nodes	1	0.0	0.0
Nodes	2	10.0	0.0
Nodes	3	20.0	0.0
Nodes	4	30.0	0.0
Nodes	5	40.0	0.0
Nodes	6	50.0	0.0
Nodes	7	60.0	0.0
Nodes	8	70.0	0.0
Nodes	9	80.0	0.0
Nodes	10	90.0	0.0
Nodes	11	100.0	0.0

H Elements	ID	Type	MatID	PropID	N1	N2
Elements	1	222	1	1	1	2
Elements	2	222	1	1	2	3
Elements	3	222	1	1	3	4
Elements	4	222	1	1	4	5
Elements	5	222	1	1	5	6
Elements	6	222	1	1	6	7
Elements	7	222	1	1	7	8
Elements	8	222	1	1	8	9
Elements	9	222	1	1	9	10
Elements	10	222	1	1	10	11

H Materials	ID	Ep
Materials	1	70000.0

H Properties	ID	A	I	zMax
Properties	1	100.0	833.0	5.0

H BC	NodeID	XDir	YDir	rZDir
BC	1	0	0	0

H Loads	NodeID	ForceX	ForceY	MomentZ
Loads	11	0	0	-10000.0

## A.4 Balken3D.in

Title Kragbalken aus 3D-Balkenelementen

H Solver	Type
Solver	1

H Nodes	ID	X	Y
Nodes	1	0.0	0.0
Nodes	2	10.0	0.0
Nodes	3	20.0	0.0
Nodes	4	30.0	0.0
Nodes	5	40.0	0.0
Nodes	6	50.0	0.0
Nodes	7	60.0	0.0
Nodes	8	70.0	0.0
Nodes	9	80.0	0.0
Nodes	10	90.0	0.0
Nodes	11	100.0	0.0

H Elements	ID	Type	MatID	PropID	N1	N2
Elements	1	223	1	1	1	2
Elements	2	223	1	1	2	3
Elements	3	223	1	1	3	4
Elements	4	223	1	1	4	5
Elements	5	223	1	1	5	6
Elements	6	223	1	1	6	7
Elements	7	223	1	1	7	8
Elements	8	223	1	1	8	9
Elements	9	223	1	1	9	10
Elements	10	223	1	1	10	11

H Materials	ID	Ep	nue	Gq
Materials	1	70000.0	0.3	27000.0

H Properties	ID	A	Iy	Iz	Kv	xz	yz	zz	zMax	yMax
Properties	1	100	833	833	1406	0	-1	0	5	5

H BC	NodeID	XDir	YDir	ZDir	rXDir	rYDir	rZDir
BC	1	0	0	0	0	0	0

H Loads	NodeID	ForceX	ForceY	ForceZ	MomentX	MomentY	MomentZ
Loads	11	0	0	0	0	10000	0



## A.5 Scheibe3K2D.in

Title Kragbalken aus 3-Knoten Scheibenelementen

H Solver      Type  
Solver        1

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	2	10	0	0
Nodes	3	20	0	0
Nodes	4	30	0	0
Nodes	5	40	0	0
Nodes	6	50	0	0
Nodes	7	60	0	0
Nodes	8	70	0	0
Nodes	9	80	0	0
Nodes	10	90	0	0
Nodes	11	100	0	0
Nodes	101	0	10	0
Nodes	102	10	10	0
Nodes	103	20	10	0
Nodes	104	30	10	0
Nodes	105	40	10	0
Nodes	106	50	10	0
Nodes	107	60	10	0
Nodes	108	70	10	0
Nodes	109	80	10	0
Nodes	110	90	10	0
Nodes	111	100	10	0

H Elements	ID	Type	MatID	PropID	N1	N2	N3
Elements	1	332	1	1	1	2	101
Elements	2	332	1	1	2	102	101
Elements	3	332	1	1	2	3	102
Elements	4	332	1	1	3	103	102
Elements	5	332	1	1	3	4	103
Elements	6	332	1	1	4	104	103
Elements	7	332	1	1	4	5	104
Elements	8	332	1	1	5	105	104
Elements	9	332	1	1	5	6	105
Elements	10	332	1	1	6	106	105
Elements	11	332	1	1	6	7	106
Elements	12	332	1	1	7	107	106
Elements	13	332	1	1	7	8	107
Elements	14	332	1	1	8	108	107
Elements	15	332	1	1	8	9	108
Elements	16	332	1	1	9	109	108
Elements	17	332	1	1	9	10	109
Elements	18	332	1	1	10	110	109
Elements	19	332	1	1	10	11	110
Elements	20	332	1	1	11	111	110

H Materials	ID	Ep	Es	nue	Gq	phi
Materials	1	7.0e+004	7.0e+004	0.3	2.69e+004	0

H Properties	ID	t
Properties	1	10.0

H BC	NodeID	XDir	YDir
BC	1	0	0
BC	101	0	0

H Loads	NodeID	ForceX	ForceY
Loads	11	-1000	0
Loads	111	1000	0

## A.6 Scheibe4K2D.in

Title Kragbalken aus 4-Knoten Scheibenelementen

H Solver      Type  
Solver          1

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	2	10	0	0
Nodes	3	20	0	0
Nodes	4	30	0	0
Nodes	5	40	0	0
Nodes	6	50	0	0
Nodes	7	60	0	0
Nodes	8	70	0	0
Nodes	9	80	0	0
Nodes	10	90	0	0
Nodes	11	100	0	0
Nodes	101	0	10	0
Nodes	102	10	10	0
Nodes	103	20	10	0
Nodes	104	30	10	0
Nodes	105	40	10	0
Nodes	106	50	10	0
Nodes	107	60	10	0
Nodes	108	70	10	0
Nodes	109	80	10	0
Nodes	110	90	10	0
Nodes	111	100	10	0

H Elements	ID	Type	MatID	PropID	N1	N2	N3	N4
Elements	1	342	1	1	1	2	102	101
Elements	2	342	1	1	2	3	103	102
Elements	3	342	1	1	3	4	104	103
Elements	4	342	1	1	4	5	105	104
Elements	5	342	1	1	5	6	106	105
Elements	6	342	1	1	6	7	107	106
Elements	7	342	1	1	7	8	108	107
Elements	8	342	1	1	8	9	109	108
Elements	9	342	1	1	9	10	110	109
Elements	10	342	1	1	10	11	111	110

H Materials	ID	Ep	Es	nue	Gq	phi
Materials	1	7.0e+004	7.0e+004	0.3	2.69e+004	0

H Properties	ID	t
Properties	1	10.0

H BC	NodeID	XDir	YDir
BC	1	0	0
BC	101	0	0

H Loads	NodeID	ForceX	ForceY
Loads	11	−1000	0
Loads	111	1000	0

## A.7 Scheibe8K2D.in

Title Kragbalken aus 8-Knoten Scheibenelementen

H Solver      Type  
Solver        1

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	2	5	0	0
Nodes	3	10	0	0
Nodes	4	15	0	0
Nodes	5	20	0	0
Nodes	6	25	0	0
Nodes	7	30	0	0
Nodes	8	35	0	0
Nodes	9	40	0	0
Nodes	10	45	0	0
Nodes	11	50	0	0
Nodes	12	55	0	0
Nodes	13	60	0	0
Nodes	14	65	0	0
Nodes	15	70	0	0
Nodes	16	75	0	0
Nodes	17	80	0	0
Nodes	18	85	0	0
Nodes	19	90	0	0
Nodes	20	95	0	0
Nodes	21	100	0	0
Nodes	101	0	5	0
Nodes	102	5	5	0
Nodes	103	10	5	0
Nodes	104	15	5	0
Nodes	105	20	5	0
Nodes	106	25	5	0
Nodes	107	30	5	0
Nodes	108	35	5	0
Nodes	109	40	5	0
Nodes	110	45	5	0
Nodes	111	50	5	0
Nodes	112	55	5	0
Nodes	113	60	5	0
Nodes	114	65	5	0
Nodes	115	70	5	0
Nodes	116	75	5	0
Nodes	117	80	5	0
Nodes	118	85	5	0
Nodes	119	90	5	0
Nodes	120	95	5	0
Nodes	121	100	5	0
Nodes	201	0	10	0
Nodes	202	5	10	0

Nodes	203	10	10	0
Nodes	204	15	10	0
Nodes	205	20	10	0
Nodes	206	25	10	0
Nodes	207	30	10	0
Nodes	208	35	10	0
Nodes	209	40	10	0
Nodes	210	45	10	0
Nodes	211	50	10	0
Nodes	212	55	10	0
Nodes	213	60	10	0
Nodes	214	65	10	0
Nodes	215	70	10	0
Nodes	216	75	10	0
Nodes	217	80	10	0
Nodes	218	85	10	0
Nodes	219	90	10	0
Nodes	220	95	10	0
Nodes	221	100	10	0

H Elements	ID	Type	MatID	PropID	N1	N2	N3	N4	N5	N6	N7	N8
Elements	1	382	1	1	1	3	203	201	2	103	202	101
Elements	2	382	1	1	3	5	205	203	4	105	204	103
Elements	3	382	1	1	5	7	207	205	6	107	206	105
Elements	4	382	1	1	7	9	209	207	8	109	208	107
Elements	5	382	1	1	9	11	211	209	10	111	210	109
Elements	6	382	1	1	11	13	213	211	12	113	212	111
Elements	7	382	1	1	13	15	215	213	14	115	214	113
Elements	8	382	1	1	15	17	217	215	16	117	216	115
Elements	9	382	1	1	17	19	219	217	18	119	218	117
Elements	10	382	1	1	19	21	221	219	20	121	220	119

H Materials	ID	Ep	Es	nue	Gq	phi
Materials	1	7.0e+004	7.0e+004	0.3	2.69e+004	0

H Properties	ID	t
Properties	1	10.0

H BC	NodeID	XDir	YDir
BC	1	0	0
BC	101	0	0
BC	201	0	0

H Loads	NodeID	ForceX	ForceY
Loads	21	-1000	0
Loads	221	1000	0

## A.8 Solid3D.in

Title Kragbalken aus Volumenelementen

H Solver      Type  
Solver        1

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	2	10	0	0
Nodes	3	20	0	0
Nodes	4	30	0	0
Nodes	5	40	0	0
Nodes	6	50	0	0
Nodes	7	60	0	0
Nodes	8	70	0	0
Nodes	9	80	0	0
Nodes	10	90	0	0
Nodes	11	100	0	0
Nodes	101	0	10	0
Nodes	102	10	10	0
Nodes	103	20	10	0
Nodes	104	30	10	0
Nodes	105	40	10	0
Nodes	106	50	10	0
Nodes	107	60	10	0
Nodes	108	70	10	0
Nodes	109	80	10	0
Nodes	110	90	10	0
Nodes	111	100	10	0
Nodes	1001	0	0	10
Nodes	1002	10	0	10
Nodes	1003	20	0	10
Nodes	1004	30	0	10
Nodes	1005	40	0	10
Nodes	1006	50	0	10
Nodes	1007	60	0	10
Nodes	1008	70	0	10
Nodes	1009	80	0	10
Nodes	1010	90	0	10
Nodes	1011	100	0	10
Nodes	1101	0	10	10
Nodes	1102	10	10	10
Nodes	1103	20	10	10
Nodes	1104	30	10	10
Nodes	1105	40	10	10
Nodes	1106	50	10	10
Nodes	1107	60	10	10
Nodes	1108	70	10	10
Nodes	1109	80	10	10
Nodes	1110	90	10	10
Nodes	1111	100	10	10

H Elements	ID	Type	MatID	PropID	N1	N2	N3	N4	N5	N6	N7	N8
Elements	1	683	1	1	1	2	102	101	1001	1002	1102	1101
Elements	2	683	1	1	2	3	103	102	1002	1003	1103	1102
Elements	3	683	1	1	3	4	104	103	1003	1004	1104	1103
Elements	4	683	1	1	4	5	105	104	1004	1005	1105	1104
Elements	5	683	1	1	5	6	106	105	1005	1006	1106	1105
Elements	6	683	1	1	6	7	107	106	1006	1007	1107	1106
Elements	7	683	1	1	7	8	108	107	1007	1008	1108	1107
Elements	8	683	1	1	8	9	109	108	1008	1009	1109	1108
Elements	9	683	1	1	9	10	110	109	1009	1010	1110	1109
Elements	10	683	1	1	10	11	111	110	1010	1011	1111	1110

H Materials	ID	Ep	Es	nue	Gq	phi
Materials	1	70000	70000	0.3	27000	0

H Properties	ID
Properties	1

H BC	NodeID	XDir	YDir	ZDir
BC	1	0	0	0
BC	101	0	0	0
BC	1001	0	0	0
BC	1101	0	0	0

H Loads	NodeID	ForceX	ForceY	ForceZ
Loads	11	−500	0	0
Loads	111	−500	0	0
Loads	1011	500	0	0
Loads	1111	500	0	0



## A.9 Schale3D.in

Title Kragbalken aus Schalenelementen

H Solver      Type  
Solver          1

H Nodes	ID	X	Y	Z
Nodes	1	0	0	0
Nodes	2	100	0	0
Nodes	3	0	10	0
Nodes	4	100	10	0
Nodes	102	10	0	0
Nodes	103	20	0	0
Nodes	104	30	0	0
Nodes	105	40	0	0
Nodes	106	50	0	0
Nodes	107	60	0	0
Nodes	108	70	0	0
Nodes	109	80	0	0
Nodes	110	90	0	0
Nodes	113	10	10	0
Nodes	114	20	10	0
Nodes	115	30	10	0
Nodes	116	40	10	0
Nodes	117	50	10	0
Nodes	118	60	10	0
Nodes	119	70	10	0
Nodes	120	80	10	0
Nodes	121	90	10	0

H Elements	ID	Type	MatID	PropID	N1	N2	N3	N4
Elements	1	543	1	1	1	102	113	3
Elements	2	543	1	1	102	103	114	113
Elements	3	543	1	1	103	104	115	114
Elements	4	543	1	1	104	105	116	115
Elements	5	543	1	1	105	106	117	116
Elements	6	543	1	1	106	107	118	117
Elements	7	543	1	1	107	108	119	118
Elements	8	543	1	1	108	109	120	119
Elements	9	543	1	1	109	110	121	120
Elements	10	543	1	1	110	2	4	121

H Materials	ID	Ep	Es	nue	Gq	phi	rho	a	b
Materials	1	7.0 e4	7.0 e4	0.3	2.7 e4	0	1	0.1	0.05

H Properties	ID	t
Properties	1	10.0

H BC	NodeID	XDir	YDir	ZDir	rXDir	rYDir	rZDir
BC	1	0	0	0	0	0	0
BC	3	0	0	0	0	0	0

H Loads	NodeID	ForceX	ForceY	ForceZ	MomentX	MomentY	MomentZ
Loads	2	0	0	0	0	5000	0
Loads	4	0	0	0	0	5000	0

## A.10 EFSchale3D.in

Title An zwei Seiten eingespannte , ebene Schale

H Solver	Type	Steps			
Solver	2	10			
H Nodes	ID	X	Y	Z	
Nodes	1	0	0	0	
Nodes	2	1	0	0	
Nodes	3	0	1	0	
Nodes	4	1	1	0	
Nodes	1002	0.1	0	0	
Nodes	1003	0.2	0	0	
Nodes	1004	0.3	0	0	
Nodes	1005	0.4	0	0	
Nodes	1006	0.5	0	0	
Nodes	1007	0.6	0	0	
Nodes	1008	0.7	0	0	
Nodes	1009	0.8	0	0	
Nodes	1010	0.9	0	0	
Nodes	1012	0	0.1	0	
Nodes	1013	0.1	0.1	0	
Nodes	1014	0.2	0.1	0	
Nodes	1015	0.3	0.1	0	
Nodes	1016	0.4	0.1	0	
Nodes	1017	0.5	0.1	0	
Nodes	1018	0.6	0.1	0	
Nodes	1019	0.7	0.1	0	
Nodes	1020	0.8	0.1	0	
Nodes	1021	0.9	0.1	0	
Nodes	1022	1	0.1	0	
Nodes	1023	0	0.2	0	
Nodes	1024	0.1	0.2	0	
Nodes	1025	0.2	0.2	0	
Nodes	1026	0.3	0.2	0	
Nodes	1027	0.4	0.2	0	
Nodes	1028	0.5	0.2	0	
Nodes	1029	0.6	0.2	0	
Nodes	1030	0.7	0.2	0	
Nodes	1031	0.8	0.2	0	
Nodes	1032	0.9	0.2	0	
Nodes	1033	1	0.2	0	
Nodes	1034	0	0.3	0	
Nodes	1035	0.1	0.3	0	
Nodes	1036	0.2	0.3	0	
Nodes	1037	0.3	0.3	0	
Nodes	1038	0.4	0.3	0	
Nodes	1039	0.5	0.3	0	
Nodes	1040	0.6	0.3	0	
Nodes	1041	0.7	0.3	0	
Nodes	1042	0.8	0.3	0	

Nodes	1043	0.9	0.3	0
Nodes	1044	1	0.3	0
Nodes	1045	0	0.4	0
Nodes	1046	0.1	0.4	0
Nodes	1047	0.2	0.4	0
Nodes	1048	0.3	0.4	0
Nodes	1049	0.4	0.4	0
Nodes	1050	0.5	0.4	0
Nodes	1051	0.6	0.4	0
Nodes	1052	0.7	0.4	0
Nodes	1053	0.8	0.4	0
Nodes	1054	0.9	0.4	0
Nodes	1055	1	0.4	0
Nodes	1056	0	0.5	0
Nodes	1057	0.1	0.5	0
Nodes	1058	0.2	0.5	0
Nodes	1059	0.3	0.5	0
Nodes	1060	0.4	0.5	0
Nodes	1061	0.5	0.5	0
Nodes	1062	0.6	0.5	0
Nodes	1063	0.7	0.5	0
Nodes	1064	0.8	0.5	0
Nodes	1065	0.9	0.5	0
Nodes	1066	1	0.5	0
Nodes	1067	0	0.6	0
Nodes	1068	0.1	0.6	0
Nodes	1069	0.2	0.6	0
Nodes	1070	0.3	0.6	0
Nodes	1071	0.4	0.6	0
Nodes	1072	0.5	0.6	0
Nodes	1073	0.6	0.6	0
Nodes	1074	0.7	0.6	0
Nodes	1075	0.8	0.6	0
Nodes	1076	0.9	0.6	0
Nodes	1077	1	0.6	0
Nodes	1078	0	0.7	0
Nodes	1079	0.1	0.7	0
Nodes	1080	0.2	0.7	0
Nodes	1081	0.3	0.7	0
Nodes	1082	0.4	0.7	0
Nodes	1083	0.5	0.7	0
Nodes	1084	0.6	0.7	0
Nodes	1085	0.7	0.7	0
Nodes	1086	0.8	0.7	0
Nodes	1087	0.9	0.7	0
Nodes	1088	1	0.7	0
Nodes	1089	0	0.8	0
Nodes	1090	0.1	0.8	0
Nodes	1091	0.2	0.8	0
Nodes	1092	0.3	0.8	0
Nodes	1093	0.4	0.8	0
Nodes	1094	0.5	0.8	0

Nodes	1095	0.6	0.8	0
Nodes	1096	0.7	0.8	0
Nodes	1097	0.8	0.8	0
Nodes	1098	0.9	0.8	0
Nodes	1099	1	0.8	0
Nodes	1100	0	0.9	0
Nodes	1101	0.1	0.9	0
Nodes	1102	0.2	0.9	0
Nodes	1103	0.3	0.9	0
Nodes	1104	0.4	0.9	0
Nodes	1105	0.5	0.9	0
Nodes	1106	0.6	0.9	0
Nodes	1107	0.7	0.9	0
Nodes	1108	0.8	0.9	0
Nodes	1109	0.9	0.9	0
Nodes	1110	1	0.9	0
Nodes	1112	0.1	1	0
Nodes	1113	0.2	1	0
Nodes	1114	0.3	1	0
Nodes	1115	0.4	1	0
Nodes	1116	0.5	1	0
Nodes	1117	0.6	1	0
Nodes	1118	0.7	1	0
Nodes	1119	0.8	1	0
Nodes	1120	0.9	1	0

H Elements	ID	Type	MatID	PropID	N1	N2	N3	N4
Elements	1	543	1	1	1	1002	1013	1012
Elements	2	543	1	1	1002	1003	1014	1013
Elements	3	543	1	1	1003	1004	1015	1014
Elements	4	543	1	1	1004	1005	1016	1015
Elements	5	543	1	1	1005	1006	1017	1016
Elements	6	543	1	1	1006	1007	1018	1017
Elements	7	543	1	1	1007	1008	1019	1018
Elements	8	543	1	1	1008	1009	1020	1019
Elements	9	543	1	1	1009	1010	1021	1020
Elements	10	543	1	1	1010	2	1022	1021
Elements	11	543	1	1	1012	1013	1024	1023
Elements	12	543	1	1	1013	1014	1025	1024
Elements	13	543	1	1	1014	1015	1026	1025
Elements	14	543	1	1	1015	1016	1027	1026
Elements	15	543	1	1	1016	1017	1028	1027
Elements	16	543	1	1	1017	1018	1029	1028
Elements	17	543	1	1	1018	1019	1030	1029
Elements	18	543	1	1	1019	1020	1031	1030
Elements	19	543	1	1	1020	1021	1032	1031
Elements	20	543	1	1	1021	1022	1033	1032
Elements	21	543	1	1	1023	1024	1035	1034
Elements	22	543	1	1	1024	1025	1036	1035
Elements	23	543	1	1	1025	1026	1037	1036
Elements	24	543	1	1	1026	1027	1038	1037
Elements	25	543	1	1	1027	1028	1039	1038

Elements	26	543	1	1	1028	1029	1040	1039
Elements	27	543	1	1	1029	1030	1041	1040
Elements	28	543	1	1	1030	1031	1042	1041
Elements	29	543	1	1	1031	1032	1043	1042
Elements	30	543	1	1	1032	1033	1044	1043
Elements	31	543	1	1	1034	1035	1046	1045
Elements	32	543	1	1	1035	1036	1047	1046
Elements	33	543	1	1	1036	1037	1048	1047
Elements	34	543	1	1	1037	1038	1049	1048
Elements	35	543	1	1	1038	1039	1050	1049
Elements	36	543	1	1	1039	1040	1051	1050
Elements	37	543	1	1	1040	1041	1052	1051
Elements	38	543	1	1	1041	1042	1053	1052
Elements	39	543	1	1	1042	1043	1054	1053
Elements	40	543	1	1	1043	1044	1055	1054
Elements	41	543	1	1	1045	1046	1057	1056
Elements	42	543	1	1	1046	1047	1058	1057
Elements	43	543	1	1	1047	1048	1059	1058
Elements	44	543	1	1	1048	1049	1060	1059
Elements	45	543	1	1	1049	1050	1061	1060
Elements	46	543	1	1	1050	1051	1062	1061
Elements	47	543	1	1	1051	1052	1063	1062
Elements	48	543	1	1	1052	1053	1064	1063
Elements	49	543	1	1	1053	1054	1065	1064
Elements	50	543	1	1	1054	1055	1066	1065
Elements	51	543	1	1	1056	1057	1068	1067
Elements	52	543	1	1	1057	1058	1069	1068
Elements	53	543	1	1	1058	1059	1070	1069
Elements	54	543	1	1	1059	1060	1071	1070
Elements	55	543	1	1	1060	1061	1072	1071
Elements	56	543	1	1	1061	1062	1073	1072
Elements	57	543	1	1	1062	1063	1074	1073
Elements	58	543	1	1	1063	1064	1075	1074
Elements	59	543	1	1	1064	1065	1076	1075
Elements	60	543	1	1	1065	1066	1077	1076
Elements	61	543	1	1	1067	1068	1079	1078
Elements	62	543	1	1	1068	1069	1080	1079
Elements	63	543	1	1	1069	1070	1081	1080
Elements	64	543	1	1	1070	1071	1082	1081
Elements	65	543	1	1	1071	1072	1083	1082
Elements	66	543	1	1	1072	1073	1084	1083
Elements	67	543	1	1	1073	1074	1085	1084
Elements	68	543	1	1	1074	1075	1086	1085
Elements	69	543	1	1	1075	1076	1087	1086
Elements	70	543	1	1	1076	1077	1088	1087
Elements	71	543	1	1	1078	1079	1090	1089
Elements	72	543	1	1	1079	1080	1091	1090
Elements	73	543	1	1	1080	1081	1092	1091
Elements	74	543	1	1	1081	1082	1093	1092
Elements	75	543	1	1	1082	1083	1094	1093
Elements	76	543	1	1	1083	1084	1095	1094
Elements	77	543	1	1	1084	1085	1096	1095

Elements	78	543	1	1	1085	1086	1097	1096
Elements	79	543	1	1	1086	1087	1098	1097
Elements	80	543	1	1	1087	1088	1099	1098
Elements	81	543	1	1	1089	1090	1101	1100
Elements	82	543	1	1	1090	1091	1102	1101
Elements	83	543	1	1	1091	1092	1103	1102
Elements	84	543	1	1	1092	1093	1104	1103
Elements	85	543	1	1	1093	1094	1105	1104
Elements	86	543	1	1	1094	1095	1106	1105
Elements	87	543	1	1	1095	1096	1107	1106
Elements	88	543	1	1	1096	1097	1108	1107
Elements	89	543	1	1	1097	1098	1109	1108
Elements	90	543	1	1	1098	1099	1110	1109
Elements	91	543	1	1	1100	1101	1112	3
Elements	92	543	1	1	1101	1102	1113	1112
Elements	93	543	1	1	1102	1103	1114	1113
Elements	94	543	1	1	1103	1104	1115	1114
Elements	95	543	1	1	1104	1105	1116	1115
Elements	96	543	1	1	1105	1106	1117	1116
Elements	97	543	1	1	1106	1107	1118	1117
Elements	98	543	1	1	1107	1108	1119	1118
Elements	99	543	1	1	1108	1109	1120	1119
Elements	100	543	1	1	1109	1110	4	1120

H Materials	ID	Ep	Es	nue	Gq	phi	rho
Materials	1	7e+10	7e+10	0.3	2.7e+10	0	2700.0

H Properties	ID	t
Properties	1	0.001

H BC	NodeID	XDir	YDir	ZDir	rXDir	rYDir	rZDir
BC	1	0	0	0	0	0	0
BC	1012	0	0	0	0	0	0
BC	1023	0	0	0	0	0	0
BC	1034	0	0	0	0	0	0
BC	1045	0	0	0	0	0	0
BC	1056	0	0	0	0	0	0
BC	1067	0	0	0	0	0	0
BC	1078	0	0	0	0	0	0
BC	1089	0	0	0	0	0	0
BC	1100	0	0	0	0	0	0
BC	3	0	0	0	0	0	0
BC	2	0	0	0	0	0	0
BC	1022	0	0	0	0	0	0
BC	1033	0	0	0	0	0	0
BC	1044	0	0	0	0	0	0
BC	1055	0	0	0	0	0	0
BC	1066	0	0	0	0	0	0
BC	1077	0	0	0	0	0	0
BC	1088	0	0	0	0	0	0
BC	1099	0	0	0	0	0	0
BC	1110	0	0	0	0	0	0

BC	4	0	0	0	0	0	0
----	---	---	---	---	---	---	---



---

INSTITUT FÜR LEICHTBAU  
Univ.-Prof. Dr.-Ing. H.-G. Reimerdes  
– Fakultät für Maschinenwesen –

Hilfe zu  
MATLAB

Dieses Dokument enthält 7 Seiten  
Aachen, den 3. September 2012

## 1 Rechen- und Matrixoperationen

### 1.1 Matrix Erstellen

$$M = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
>> M = [1 3 5; 2 4 6; 7 8 9]
```

#### 1.1.1 Matrix Initialisieren

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
>> M = zeros(3,4)
```

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

```
>> M = ones(3,3)
```

#### 1.1.2 Einheitsmatrix

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
>> M = eye(3)
```

### 1.2 Lineare Vektoren Erstellen

Lineare Vektoren werden in MATLAB in der Regel als Zeilenvektoren erstellt. Durch Transponieren (1.3.4) können diese Vektoren in Spaltenvektoren umgewandelt werden.

#### 1.2.1 Vektoren mit Linearer Schrittweite

$$\vec{V} = ( 1 \ 2 \ 3 \ 4 \ 5 )$$

```
>> V = 1 : 5
```

$$\vec{V} = \begin{pmatrix} 0.0 \\ 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \\ 1.0 \end{pmatrix}$$

```
>> V = (0 : 0.2 : 1)'
```

### 1.2.2 Vektoren mit vorgegebener Zahl von Elementen

$$\vec{V} = ( 0 \quad 0.3333 \quad 0.6667 \quad 1 )$$

```
>> V = linspace(0, 1, 4)
```

## 1.3 Rechenoperationen

### 1.3.1 Matrizen Addieren

$$A = B + C$$

```
>> A = B + C
```

### 1.3.2 Matrizen Multiplizieren

$$A = B * C$$

```
>> A = B * C
```

### 1.3.3 Matrizen Potenzieren

$$A = B^2$$

```
>> A = B^2
```

### 1.3.4 Matrizen Transponieren

$$A = B^T$$

```
>> A = B'
```

### 1.3.5 Matrizen Invertieren

$$A = B^{-1}$$

```
>> A = inv(B)
```

### 1.3.6 Lineares Gleichungssystem Lösen

Das Lösen des Linearen Gleichungssystems  $A \cdot x = b$  erfolgt am effektivsten mit dem Befehl

```
>> x = A \ b
```

## 1.4 Komponentenweise Rechenoperationen

Komponentenweise Rechenoperationen werden durch einen `.` vor dem Operator (`*`, `/`, `^`) beschrieben. Dadurch können sehr schnell Komponenten von zwei gleichgroßen Matrizen miteinander multipliziert, dividiert oder potenziert werden. Komponentenweise Rechenoperationen folgen somit **nicht** den normalen Rechenregeln für Matrizen.

### 1.4.1 Matrizen Komponentenweise Multiplizieren

$$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$C = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$$

$$A = B .* C = \begin{pmatrix} 2 & 8 \\ 18 & 32 \end{pmatrix}$$

```
>> B = [1 2; 3 4]
```

```
>> C = [2 4; 6 8]
```

```
>> A = B .* C
```

### 1.4.2 Matrizen Komponentenweise Dividieren

$$B = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$A = B ./ C = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

```
>> B = [2 4; 6 8]
```

```
>> C = [1 2; 3 4]
```

```
>> A = B ./ C
```

### 1.4.3 Matrizen Komponentenweise Potenzieren

$$B = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$A = B.^C = \begin{pmatrix} 2 & 16 \\ 216 & 4096 \end{pmatrix}$$

```
>> B = [2 4; 6 8]
```

```
>> C = [1 2; 3 4]
```

```
>> A = B.^C
```

## 1.5 Größe einer Matrix

$$\vec{V} = \begin{pmatrix} 1.1 \\ 2.2 \\ 3.3 \\ 4.4 \\ 5.5 \\ 6.6 \\ 7.7 \end{pmatrix}$$

Die Größe von  $\vec{V}$  ist  $7 \times 1$  ( $a=7, b=1$ )

```
>> V = [1.1; 2.2; 3.3; 4.4; 5.5; 6.6; 7.7]
>> a = size(V,1)
>> b = size(V,2)
```

## 1.6 Elemente und Submatrizen

Über die Indizes einer Matrix kann man auf einzelne Elemente oder ganze Submatrizen einer Matrix zugreifen. Dabei steht der **erste Index** für die **Zeile(n)** und der **zweite Index** für die **Spalte(n)**.

### 1.6.1 Ganze Spalte / Zeile einer Matrix

Durch den **:** Operator kann auf eine ganze Zeile oder Spalte zugegriffen werden. Er entspricht dem Vektor **1 : end**.

$$M = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$
$$A = \begin{pmatrix} 3 \\ 4 \\ 8 \end{pmatrix}$$

```
>> M = [1 3 5; 2 4 6; 7 8 9]
>> A = M(:,2)
```

### 1.6.2 Submatrizen

Vektoren als Indizes erlauben das gezielte Abgreifen von mehreren Elementen zu einer Submatrix.

$$M = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$
$$B = \begin{pmatrix} 3 & 5 \\ 4 & 6 \end{pmatrix}$$

```
>> M = [1 3 5; 2 4 6; 7 8 9]
```

```
>> B = M(1:2,2:3)
```

$$\vec{V} = \begin{pmatrix} 1.1 \\ 2.2 \\ 3.3 \\ 3.4 \\ 5.5 \\ 6.6 \\ 7.7 \end{pmatrix}$$

$$\vec{U} = \begin{pmatrix} 1.1 \\ 7.7 \\ 3.3 \end{pmatrix}$$

```
>> V = [1.1; 2.2; 3.3; 4.4; 5.5; 6.6; 7.7]
```

```
>> U = V([1 7 3])
```

## 2 Funktionen

### 2.1 Erstellen von Funktionen

Erstelle eine neue .m Datei im aktuellen Verzeichnis.

```
function [output1,output2,...] = NameDerFunktion(input1,input2,...)
% NAMEDERFUNKTION Kurzzusammenfassung der Funktion
% Weitere Zusammenfassungen, Erklärungen, etc

output1 = f(input1, input2); % Semikolon unterdrückt Ausgabe
output2 = ... ; % Weitere Kommentare

end
```

### 2.2 Aufrufen von Funktionen

Die entsprechende .m Datei muss sich im aktuellen Verzeichnis oder einem eingebundenen Unterordner (addpath('Unterordner')) befinden.

```
[output1, output2, ...] = NameDerFunktion(input1, input2, ...)
```

### 2.3 Wichtige Funktionen

#### 2.3.1 disp(['String1', 'String2'])

*disp()* gibt einen Zeilenvektor aus mehreren Strings als Text im Command Window aus.

#### 2.4 figure

*figure* erstellt ein Grafik Fenster und gibt dessen handle zurück.

#### 2.4.1 `plot(x, y)`

`plot(x, y)` zeichnet die Punkte definiert durch die Zeilen- oder Spaltenvektoren `x` und `y` in ein Koordinatensystem des zuletzt aktiven Grafikfensters.

#### 2.4.2 `clear all`

`clear all` löscht alle im Speicher befindlichen Daten.

#### 2.4.3 `close all`

`close all` schließt alle Grafikfenster. Bei einem *figure handle* statt dem Parameter `all` wird nur dieses Fenster geschlossen.

#### 2.4.4 `clc`

`clc` löscht sämtlichen im Command Window ausgegebenen Text.

#### 2.4.5 `for` - Schleife

```
for i = 1 : n
    v(i) = myfun(i);
end
```

#### 2.4.6 `if` - Bedingung

```
if a > 0
    b = 1;
else
    b = 0;
end
```

### 3 Arrays

#### 3.1 Matrix - Array

Matrix - Arrays sind der Standard Array von Matlab. Sie können als skalare oder auch beliebig dimensionale Tensoren definiert und berechnet werden.

Die verschiedenen Werte eines Matrix - Arrays müssen den gleichen Datentyp (z.B. double, float, integer, ...) haben.

- **Initialisierung:** `A = zeros(2, 3);`
- **Zuweisung:** `A(1, 2:3) = [1, -5];`
- **Auslesen:** `A(2,2)`

### 3.2 String - Array

Ein String ist ein Array aus mehreren Zeichen. Sie stellen einen Sonderfall der Matrix - Arrays dar.

- **Zuweisung:** `str = 'asdf';`
- **Auslesen:** `str(2:3)`

### 3.3 Cell - Array

Anders als ein Matrix - Array können Cell - Arrays beliebige, verschiedene Datentypen oder auch andere Arrays als Werte enthalten.

Sie eignen sich besonders zur Übergabe von Strings unterschiedlicher Länge oder zur sortierten Übergabe von einer variablen Anzahl von Variablen.

- **Initialisierung:** `C = cell(3,4);`
- **Zuweisung:** `C{1,1} = TestMatrix1; C(1,2) = {TestMatrix2}; C(2,2) = {'Test'};`
- **Auslesen:** *Inhalt:* `C{1,2}` *Unter - Cell - Array:* `C(1,1)`

### 3.4 Struct - Array

Ein Struct - Array ist eine (auch mehrdimensionale) Sammlung von mehreren Untervariablen in einer geordneten Struktur.

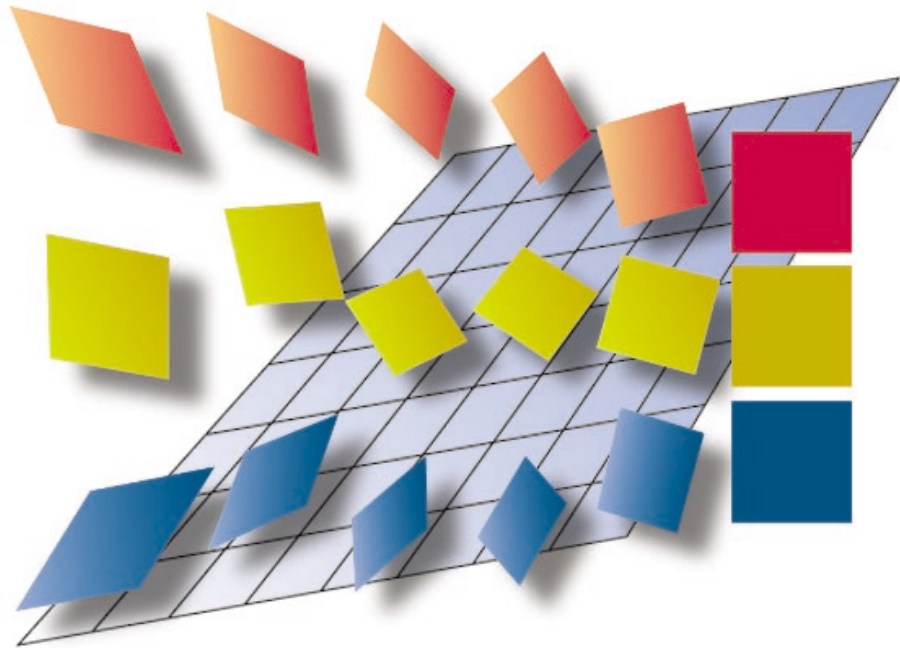
Die Untervariablen können beliebige und verschiedene Daten- und Arraytypen (auch weitere Struct - Arrays) sein.

Struct - Arrays sind besonders nützlich zur übersichtlichen Übergabe von Werten, die zwar inhaltlich zusammengehören, aber mathematisch oder typenmäßig nicht zusammenpassen.

Die Größe des Struct - Arrays wird dynamisch angepasst, was zu Speicherengpässen und Verlangsamung des Programmes führen kann. Nichtdefinierte Werte werden mit leeren Matrix - Arrays gefüllt.

- **Initialisierung:** `S = struct([]);`
- **Zuweisung:** `S(1, 5).f1 = 'TestFeld'; S(1, 5).f2 = 123;`
- **Auslesen:** *Unter - Struct - Array:* `S(1,5)` *Feldwert:* `S(1, 5).f1` *Mehrere:* `[S(1, 3:5)]`





# **C A L F E M**

## **A FINITE ELEMENT TOOLBOX**

### ***Version 3.4***

P-E AUSTRELL, O DAHLBLOM, J LINDEMANN,  
A OLSSON, K-G OLSSON, K PERSSON, H PETERSSON,  
M RISTINMAA, G SANDBERG, P-A WERNBERG

C A L F E M  
A FINITE ELEMENT TOOLBOX  
*Version 3.4*

P-E AUSTRELL, O DAHLBLOM, J LINDEMANN,  
A OLSSON, K-G OLSSON, K PERSSON, H PETERSSON,  
M RISTINMAA, G SANDBERG, P-A WERNBERG

ISBN: 91-8855823-1

Copyright © 2004 by Structural Mechanics, LTH, Sweden.

The software described in this document is furnished under a license agreement. The software may be used or copied only under terms in the license agreement.

No part of this manual may be photocopied or reproduced in any form without the prior written consent by the Division of Structural Mechanics.

© Copyright 1992–2004 by the Division of Structural Mechanics at Lund University. All rights reserved.

CALFEM is the trademark of the Division of Structural Mechanics, Lund University.  
MATLAB is the trademark of The MathWorks, Inc.

E-mail address:

`calfem@byggmek.lth.se`

Homepage:

`http://www.byggmek.lth.se/Calfem`

Contacts:

The Division of Structural Mechanics  
Lund University  
PO Box 118  
SE-221 00 Lund  
SWEDEN  
Phone: +46 46 222 0000  
Fax: +46 46 222 4420

## **Preface**

CALFEM® is an interactive computer program for teaching the finite element method (FEM). The name CALFEM is an abbreviation of "Computer Aided Learning of the Finite Element Method". The program can be used for different types of structural mechanics problems and field problems.

CALFEM, the program and its built-in philosophy have been developed at the Division of Structural Mechanics, Lund University, starting in the late 70's. Many coworkers, former and present, have been engaged in the development at different stages.

This release represents the latest development of CALFEM. The functions for finite element applications are all MATLAB functions (.m-files) as described in the MATLAB manual. We believe that this environment increases the versatility and handling of the program and, above all, the ease of teaching the finite element method. CALFEM also works with Octave, presently with exception for some graphical functions.

Lund, October 7, 2004

The authors



## Contents

<b>1</b>	<b>Introduction</b>	<b>1 – 1</b>
<b>2</b>	<b>General purpose functions</b>	<b>2 – 1</b>
<b>3</b>	<b>Matrix functions</b>	<b>3 – 1</b>
<b>4</b>	<b>Material functions</b>	<b>4 – 1</b>
<b>5</b>	<b>Element functions</b>	<b>5.1 – 1</b>
5.1	Introduction . . . . .	5.1 – 1
5.2	Spring element . . . . .	5.2 – 1
5.3	Bar elements . . . . .	5.3 – 1
5.4	Heat flow elements . . . . .	5.4 – 1
5.5	Solid elements . . . . .	5.5 – 1
5.6	Beam elements . . . . .	5.6 – 1
5.7	Plate element . . . . .	5.7 – 1
<b>6</b>	<b>System functions</b>	<b>6.1 – 1</b>
6.1	Introduction . . . . .	6.1 – 1
6.2	Static system functions . . . . .	6.2 – 1
6.3	Dynamic system functions . . . . .	6.3 – 1
<b>7</b>	<b>Statements and macros</b>	<b>7 – 1</b>
<b>8</b>	<b>Graphics functions</b>	<b>8 – 1</b>
<b>9</b>	<b>User's Manual, examples</b>	<b>9.1 – 1</b>
9.1	Introduction . . . . .	9.1 – 1
9.2	MATLAB introduction . . . . .	9.2 – 1
9.3	Static analysis . . . . .	9.3 – 1
9.4	Dynamic analysis . . . . .	9.4 – 1
9.5	Nonlinear analysis . . . . .	9.5 – 1



## 1 Introduction

The computer program CALFEM is a MATLAB toolbox for finite element applications. This manual concerns mainly the finite element functions, but it also contains descriptions of some often used MATLAB functions.

The finite element analysis can be carried out either interactively or in a batch oriented fashion. In the interactive mode the functions are evaluated one by one in the MATLAB command window. In the batch oriented mode a sequence of functions are written in a file named .m-file, and evaluated by writing the file name in the command window. The batch oriented mode is a more flexible way of performing finite element analysis because the .m-file can be written in an ordinary editor. This way of using CALFEM is recommended because it gives a structured organization of the functions. Changes and reruns are also easily executed in the batch oriented mode.

A command line consists typically of functions for vector and matrix operations, calls to functions in the CALFEM finite element library or commands for workspace operations. An example of a command line for a matrix operation is

$$C = A + B'$$

where two matrices  $A$  and  $B'$  are added together and the result is stored in matrix  $C$ . The matrix  $B'$  is the transpose of  $B$ . An example of a call to the element library is

$$Ke = \text{bar1e}(k)$$

where the two-by-two element stiffness matrix  $K^e$  is computed for a spring element with spring stiffness  $k$ , and is stored in the variable  $Ke$ . The input argument is given within parentheses ( ) after the name of the function. Some functions have multiple input arguments and/or multiple output arguments. For example

$$[\text{lambda}, X] = \text{eigen}(K, M)$$

computes the eigenvalues and eigenvectors to a pair of matrices  $K$  and  $M$ . The output variables - the eigenvalues stored in the vector **lambda** and the corresponding eigenvectors stored in the matrix **X** - are surrounded by brackets [ ] and separated by commas. The input arguments are given inside the parentheses and also separated by commas.

The statement

`help function`

provides information about purpose and syntax for the specified function.



The available functions are organized in groups as follows. Each group is described in a separate chapter.

<b>Groups of functions</b>	
<b>General purpose commands</b>	for managing variables, workspace, output etc
<b>Matrix functions</b>	for matrix handling
<b>Material functions</b>	for computing material matrices
<b>Element functions</b>	for computing element matrices and element forces
<b>System functions</b>	for setting up and solving systems of equations
<b>Statement functions</b>	for algorithm definitions
<b>Graphics functions</b>	for plotting

## 2 General purpose functions

The general purpose functions are used for managing variables and workspace, control of output etc. The functions listed here are a subset of the general purpose functions described in the MATLAB manual. The functions can be divided into the following groups

Managing commands and functions	
help	Online documentation
type	List .m-file
what	Directory listing of .m-, .mat- and .mex-files
...	Continuation
%	Write a comment line

Managing variables and the workspace	
clear	Remove variables from workspace
disp	Display variables in workspace on display screen
load	Retrieve variable from disk and load in workspace
save	Save matrix bank variable on disk
who,	List directory of variables in workspace
whos	

Working with files and controlling the command window	
diary	Save session in a named file
echo	Control output on the display screen
format	Control the output display format
quit	Stop execution and exit from the CALFEM program

## **clear**

---

### **Purpose:**

Remove variables from workspace.

### **Syntax:**

```
clear  
clear name1 name2 name3 ...
```

### **Description:**

`clear` removes all variables from workspace.

`clear name1 name2 name3 ...` removes specified variables from workspace.

### **Note:**

This is a MATLAB built-in function. For more information about the `clear` function, type `help clear`.

---

**diary**

---

**Purpose:**

Save session in a disk file.

**Syntax:**

`diary filename`  
`diary off`  
`diary on`

**Description:**

`diary filename` writes a copy of all subsequent keyboard input and most of the resulting output (but not graphs) on the named file. If the file *filename* already exists, the output is appended to the end of that file.

`diary off` stops storage of the output.

`diary on` turns it back on again, using the current filename or default filename `diary` if none has yet been specified.

The `diary` function may be used to store the current session for later runs. To make this possible, finish each command line with semicolon `;` to avoid the storage of intermediate results on the named diary file.

**Note:**

This is a MATLAB built-in function. For more information about the `diary` function, type `help diary`.

## **disp**

---

### **Purpose:**

Display a variable in matrix bank on display screen.

### **Syntax:**

`disp(A)`

### **Description:**

`disp(A)` displays the matrix *A* on the display screen.

### **Note:**

This is a MATLAB built-in function. For more information about the `disp` function, type `help disp`.

---

**echo**

---

**Purpose:**

Control output on the display screen.

**Syntax:**

echo on  
echo off  
echo

**Description:**

echo on turns on echoing of commands inside Script-files.

echo off turns off echoing.

echo by itself, toggles the echo state.

**Note:**

This is a MATLAB built-in function. For more information about the `echo` function, type `help echo`.

## **format**

---

### **Purpose:**

Control the output display format.

### **Syntax:**

See the listing below.

### **Description:**

**format** controls the output format. By default, MATLAB displays numbers in a short format with five decimal digits.

Command	Result	Example
<b>format short</b>	5 digit scaled fixed point	3.1416
<b>format long</b>	15 digit scaled fixed point	3.14159265358979
<b>format short e</b>	5 digit floating point	3.1416e+000
<b>format long e</b>	16 digit floating point	3.141592653589793e+000

### **Note:**

This is a MATLAB built-in function. For more information about the **format** function, type **help format**.

---

**help**

---

**Purpose:**

Display a description of purpose and syntax for a specific function.

**Syntax:**

`help function name`

**Description:**

`help` provides an online documentation for the specified function.

**Example:**

Typing

```
>> help bar1e
```

yields

```
Ke=bar1e(ep)
```

---

PURPOSE

Compute element stiffness matrix  
for spring (analog) element.

INPUT: `ep = [k]`; spring stiffness or analog quantity.

OUTPUT: `Ke` : stiffness matrix, `dim(Ke)= 2 x 2`

---

**Note:**

This is a MATLAB built-in function. For more information about the `help` function, type `help help`.



## load

---

### Purpose:

Retrieve variable from disk and load in workspace.

### Syntax:

```
load filename
load filename.ext
```

### Description:

`load filename` retrieves the variables from the binary file *filename.mat*.

`load filename.ext` reads the ASCII file *filename.ext* with numeric data arranged in *m* rows and *n* columns. The result is an *m*-by-*n* matrix residing in workspace with the name *filename*, i.e. with the extension stripped.

### Note:

This is a MATLAB built-in function. For more information about the `load` function, type `help load`.

---

**quit**

---

**Purpose:**

Terminate CALFEM session.

**Syntax:**

quit

**Description:**

quit *filename* terminates the CALFEM without saving the workspace.

**Note:**

This is a MATLAB built-in function. For more information about the quit function, type help quit.

## **save**

---

### **Purpose:**

Save workspace variables on disk.

### **Syntax:**

```
save filename  
save filename variables  
save filename variables -ascii
```

### **Description:**

`save filename` writes all variables residing in workspace in a binary file named *filename.mat*

`save filename variables` writes named variables, separated by blanks, in a binary file named *filename.mat*

`save filename variables -ascii` writes named variables in an ASCII file named *filename*.

### **Note:**

This is a MATLAB built-in function. For more information about the `save` function, type `help save`.

---

**type**

---

**Purpose:**

List file.

**Syntax:**

`type filename`

**Description:**

`type filename` lists the specified file. Use path names in the usual way for your operating system. If a filename extension is not given, .m is added by default. This makes it convenient to list the contents of .m-files on the screen.

**Note:**

This is a MATLAB built-in function. For more information about the `type` function, type `help type`.

## **what**

---

### **Purpose:**

Directory listing of .m-files, .mat-files and .mex-files.

### **Syntax:**

`what`  
`what dirname`

### **Description:**

`what` lists the .m-files, .mat-files and .mex-files in the current directory.

`what dirname` lists the files in directory *dirname* in the MATLAB search path. The syntax of the path depends on your operating system.

### **Note:**

This is a MATLAB built-in function. For more information about the `what` function, type `help what`.

**Purpose:**

List directory of variables in matrix bank.

**Syntax:**

who  
whos

**Description:**

who lists the variables currently in memory.

whos lists the current variables and their size.

**Examples:**

who

Your variables are:

A   B   C  
K   M   X  
k   lambda

whos

name	size	elements	bytes	density	complex
A	3-by-3	9	72	Full	No
B	3-by-3	9	72	Full	No
C	3-by-3	9	72	Full	No
K	20-by-20	400	3200	Full	No
M	20-by-20	400	3200	Full	No
X	20-by-20	400	3200	Full	No
k	1-by-1	1	8	Full	No
lambda	20-by-1	20	160	Full	No

Grand total is 1248 elements using 9984 bytes

**Note:**

These are MATLAB built-in functions. For more information about the functions, type `help who` or `help whos`.

...

---

**Purpose:**

Continuation.

**Syntax:**

...

**Description:**

An expression can be continued on the next line by using ... .

**Note:**

This is a MATLAB built-in function.

%

---

**Purpose:**

Write a comment line.

**Syntax:**

% *arbitrary text*

**Description:**

An arbitrary text can be written after the symbol %.

**Note:**

This is a MATLAB built-in character.





### 3 Matrix functions

The group of matrix functions comprises functions for vector and matrix operations and also functions for sparse matrix handling. MATLAB has two storage modes, full and sparse. Only nonzero entries and their indices are stored for sparse matrices. Sparse matrices are not created automatically. But once initiated, sparsity propagates. Operations on sparse matrices produce sparse matrices and operations on a mixture of sparse and full matrices also normally produce sparse matrices.

The following functions are described in this chapter:

Vector and matrix operations	
<code>[] ( ) =</code>	Special characters
<code>' . , ;</code>	Special characters
<code>:</code>	Create vectors and do matrix subscripting
<code>+ - * /</code>	Matrix arithmetic
<code>abs</code>	Absolute value
<code>det</code>	Matrix determinant
<code>diag</code>	Diagonal matrices and diagonals of a matrix
<code>inv</code>	Matrix inverse
<code>length</code>	Vector length
<code>max</code>	Maximum element(s) of a matrix
<code>min</code>	Minimum element(s) of a matrix
<code>ones</code>	Generate a matrix of all ones
<code>red</code>	Reduce the size of a square matrix
<code>size</code>	Matrix dimensions
<code>sqrt</code>	Square root
<code>sum</code>	Sum of the elements of a matrix
<code>zeros</code>	Generate a zero matrix

Sparse matrix handling	
<code>full</code>	Convert sparse matrix to full matrix
<code>sparse</code>	Create sparse matrix
<code>spy</code>	Visualize sparsity structure

[ ] ( ) = ' . , ;

---

**Purpose:**

Special characters.

**Syntax:**

[ ] ( ) = ' . , ;

**Description:**

- [ ] Brackets are used to form vectors and matrices.
- ( ) Parentheses are used to indicate precedence in arithmetic expressions and to specify an element of a matrix.
- = Used in assignment statements.
- ' Matrix transpose.  $X'$  is the transpose of  $X$ . If  $X$  is complex, the apostrophe sign performs complex conjugate as well. Do  $X'$  if only the transpose of the complex matrix is desired
- . Decimal point. 314/100, 3.14 and 0.314e1 are all the same.
- , Comma. Used to separate matrix subscripts and function arguments.
- ; Semicolon. Used inside brackets to end rows. Used after an expression to suppress printing or to separate statements.

**Examples:**

By the statement

$$a = 2$$

the scalar  $a$  is assigned a value of 2. An element in a matrix may be assigned a value according to

$$A(2,5) = 3$$

The statement

$$D = [ 1 \ 2 ; 3 \ 4 ]$$

results in matrix

$$D = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

stored in the matrix bank. To copy the contents of the matrix  $D$  to a matrix  $E$ , use

$$E = D$$

The character  $'$  is used in the following statement to store the transpose of the matrix  $A$  in a new matrix  $F$

$$F = A'$$

**Note:**

These are MATLAB built-in characters.

:

**Purpose:**

Create vectors and do matrix subscripting.

**Description:**

The colon operator uses the following rules to create regularly spaced vectors:

$j : k$  is the same as  $[j, j + 1, \dots, k]$

$j : i : k$  is the same as  $[j, j + i, j + 2i, \dots, k]$

The colon notation may also be used to pick out selected rows, columns, and elements of vectors and matrices:

$A(:, j)$  is the  $j$ :th column of  $A$

$A(i, :)$  is the  $i$ :th row of  $A$

**Examples:**

The colon ':' used with integers

$$d = 1 : 4$$

results in a row vector

$$d = [1 \ 2 \ 3 \ 4]$$

stored in the workspace.

The colon notation may be used to display selected rows and columns of a matrix on the terminal. For example, if we have created a 3-times-4 matrix  $D$  by the statement

$$D = [d; 2 * d; 3 * d]$$

resulting in

$$D = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

columns three and four are displayed by entering

$$D(:, 3 : 4)$$

resulting in

$$D(:, 3 : 4) = \begin{bmatrix} 3 & 4 \\ 6 & 8 \\ 9 & 12 \end{bmatrix}$$

:

---

In order to copy parts of the D matrix into another matrix the colon notation is used as

$$E(3 : 4, 2 : 3) = D(1 : 2, 3 : 4)$$

Assuming the matrix E was a zero matrix before the statement is executed, the result will be

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 6 & 8 & 0 \end{bmatrix}$$

**Note:**

This is a MATLAB built-in character.

**Purpose:**

Matrix arithmetic.

**Syntax:**

$A + B$

$A - B$

$A * B$

$A/s$

**Description:**

Matrix operations are defined by the rules of linear algebra.

**Examples:**

An example of a sequence of matrix-to-matrix operations is

$$D = A + B - C$$

A matrix-to-vector multiplication followed by a vector-to-vector subtraction may be defined by the statement

$$b = c - A * x$$

and finally, to scale a matrix by a scalar  $s$  we may use

$$B = A/s$$

**Note:**

These are MATLAB built-in operators.

## abs

---

### Purpose:

Absolute value.

### Syntax:

`B=abs(A)`

### Description:

`B=abs(A)` computes the absolute values of the elements of matrix `A` and stores them in matrix `B`.

### Examples:

Assume the matrix

$$C = \begin{bmatrix} -7 & 4 \\ -3 & -8 \end{bmatrix}$$

The statement `D=abs(C)` results in a matrix

$$D = \begin{bmatrix} 7 & 4 \\ 3 & 8 \end{bmatrix}$$

stored in the workspace.

### Note:

This is a MATLAB built-in function. For more information about the `abs` function, type `help abs`.

---

**det**

---

**Purpose:**

Matrix determinant.

**Syntax:**

`a=det(A)`

**Description:**

`a=det(A)` computes the determinant of the matrix **A** and stores it in the scalar **a**.

**Note:**

This is a MATLAB built-in function. For more information about the **det** function, type `help det`.



## diag

---

### Purpose:

Diagonal matrices and diagonals of a matrix.

### Syntax:

$\mathbf{M}=\text{diag}(\mathbf{v})$   
 $\mathbf{v}=\text{diag}(\mathbf{M})$

### Description:

For a vector  $\mathbf{v}$  with  $n$  components, the statement  $\mathbf{M}=\text{diag}(\mathbf{v})$  results in an  $n \times n$  matrix  $\mathbf{M}$  with the elements of  $\mathbf{v}$  as the main diagonal.

For a  $n \times n$  matrix  $\mathbf{M}$ , the statement  $\mathbf{v}=\text{diag}(\mathbf{M})$  results in a column vector  $\mathbf{v}$  with  $n$  components formed by the main diagonal in  $\mathbf{M}$ .

### Note:

This is a MATLAB built-in function. For more information about the **diag** function, type `help diag`.

**Purpose:**

Convert sparse matrices to full storage class.

**Syntax:**

A=full(S)

**Description:**

A=full(S) converts the storage of a matrix from sparse to full. If A is already full, full(A) returns A.

**Note:**

This is a MATLAB built-in function. For more information about the full function, type `help full`.

## **inv**

---

### **Purpose:**

Matrix inverse.

### **Syntax:**

$B = \text{inv}(A)$

### **Description:**

$B = \text{inv}(A)$  computes the inverse of the square matrix  $A$  and stores the result in the matrix  $B$ .

### **Note:**

This is a MATLAB built-in function. For more information about the `inv` function, type `help inv`.

---

**length**

---

**Purpose:**

Vector length.

**Syntax:**

`n=length(x)`

**Description:**

`n=length(x)` returns the dimension of the vector `x`.

**Note:**

This is a MATLAB built-in function. For more information about the **length** function, type `help length`.

## **max**

---

### **Purpose:**

Maximum element(s) of a matrix.

### **Syntax:**

$\mathbf{b}=\text{max}(\mathbf{A})$

### **Description:**

For a vector  $\mathbf{a}$ , the statement  $\mathbf{b}=\text{max}(\mathbf{a})$  assigns the scalar  $\mathbf{b}$  the maximum element of the vector  $\mathbf{a}$ .

For a matrix  $\mathbf{A}$ , the statement  $\mathbf{b}=\text{max}(\mathbf{A})$  returns a row vector  $\mathbf{b}$  containing the maximum elements found in each column vector in  $\mathbf{A}$ .

The maximum element found in a matrix may thus be determined by  $\mathbf{c}=\text{max}(\text{max}(\mathbf{A}))$ .

### **Examples:**

Assume the matrix  $\mathbf{B}$  is defined as

$$\mathbf{B} = \begin{bmatrix} -7 & 4 \\ -3 & -8 \end{bmatrix}$$

The statement  $\mathbf{d}=\text{max}(\mathbf{B})$  results in a row vector

$$\mathbf{d} = \begin{bmatrix} -3 & 4 \end{bmatrix}$$

The maximum element in the matrix  $\mathbf{B}$  may be found by  $\mathbf{e}=\text{max}(\mathbf{d})$  which results in the scalar  $\mathbf{e} = 4$ .

### **Note:**

This is a MATLAB built-in function. For more information about the **max** function, type **help max**.

**Purpose:**

Minimum element(s) of a matrix.

**Syntax:**

`b=min(A)`

**Description:**

For a vector **a**, the statement `b=min(a)` assigns the scalar **b** the minimum element of the vector **a**.

For a matrix **A**, the statement `b=min(A)` returns a row vector **b** containing the minimum elements found in each column vector in **A**.

The minimum element found in a matrix may thus be determined by `c=min(min(A))`.

**Examples:**

Assume the matrix **B** is defined as

$$\mathbf{B} = \begin{bmatrix} -7 & 4 \\ -3 & -8 \end{bmatrix}$$

The statement `d=min(B)` results in a row vector

$$\mathbf{d} = \begin{bmatrix} -7 & -8 \end{bmatrix}$$

The minimum element in the matrix **B** is then found by `e=min(d)`, which results in the scalar **e** = -8.

**Note:**

This is a MATLAB built-in function. For more information about the `min` function, type `help min`.

## **ones**

---

### **Purpose:**

Generate a matrix of all ones.

### **Syntax:**

`A=ones(m,n)`

### **Description:**

`A=ones(m,n)` results in an `m`-times-`n` matrix `A` with all ones.

### **Note:**

This is a MATLAB built-in function. For more information about the `ones` function, type `help ones`.

**Purpose:**

Reduce the size of a square matrix by omitting rows and columns.

**Syntax:**

$B = \text{red}(A, b)$

**Description:**

$B = \text{red}(A, b)$  reduces the square matrix  $A$  to a smaller matrix  $B$  by omitting rows and columns of  $A$ . The indices for rows and columns to be omitted are specified by the column vector  $b$ .

**Examples:**

Assume that the matrix  $A$  is defined as

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

and  $b$  as

$$b = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

The statement  $B = \text{red}(A, b)$  results in the matrix

$$B = \begin{bmatrix} 1 & 3 \\ 9 & 11 \end{bmatrix}$$



## size

---

### Purpose:

Matrix dimensions.

### Syntax:

```
d=size(A)
[m,n]=size(A)
```

### Description:

`d=size(A)` returns a vector with two integer components, `d=[m,n]`, from the matrix `A` with dimensions `m` times `n`.

`[m,n]=size(A)` returns the dimensions `m` and `n` of the  $m \times n$  matrix `A`.

### Note:

This is a MATLAB built-in function. For more information about the `size` function, type `help size`.

---

**sparse**

---

**Purpose:**

Create sparse matrices.

**Syntax:**

S=sparse(A)  
S=sparse(m,n)

**Description:**

S=sparse(A) converts a full matrix to sparse form by extracting all nonzero matrix elements. If S is already sparse, **sparse(S)** returns S.

S=sparse(m,n) generates an m-times-n sparse zero matrix.

**Note:**

This is a MATLAB built-in function. For more information about the **sparse** function, type **help sparse**.

## **spy**

---

### **Purpose:**

Visualize matrix sparsity structure.

### **Syntax:**

`spy(S)`

### **Description:**

`spy(S)` plots the sparsity structure of any matrix **S**. **S** is usually a sparse matrix, but the function also accepts full matrices and the nonzero matrix elements are plotted.

### **Note:**

This is a MATLAB built-in function. For more information about the **spy** function, type `help spy`.

---

**sqrt**

---

**Purpose:**

Square root.

**Syntax:**

B=sqrt(A)

**Description:**

B=sqrt(A) computes the square root of the elements in matrix A and stores the result in matrix B.

**Note:**

This is a MATLAB built-in function. For more information about the **sqrt** function, type **help sqrt**.

## **sum**

---

### **Purpose:**

Sum of the elements of a matrix.

### **Syntax:**

`b=sum(A)`

### **Description:**

For a vector **a**, the statement `b=sum(a)` results in a scalar **a** containing the sum of all elements of **a**.

For a matrix **A**, the statement `b=sum(A)` returns a row vector **b** containing the sum of the elements found in each column vector of **A**.

The sum of all elements of a matrix is determined by `c=sum(sum(A))`.

### **Note:**

This is a MATLAB built-in function. For more information about the `sum` function, type `help sum`.

**Purpose:**

Generate a zero matrix.

**Syntax:**

`A=zeros(m,n)`

**Description:**

`A=zeros(m,n)` results in an `m`-times-`n` matrix `A` of zeros.

**Note:**

This is a MATLAB built-in function. For more information about the **zeros** function, type `help zeros`.



---

## 4 Material functions

The group of material functions comprises functions for constitutive models. The available models can treat linear elastic and isotropic hardening von Mises material. These material models are defined by the functions:

Material property functions	
hooke	Form linear elastic constitutive matrix
mises	Compute stresses and plastic strains for isotropic hardening von Mises material
dmises	Form elasto-plastic continuum matrix for isotropic hardening von Mises material



## hooke

---

### Purpose:

Compute material matrix for a linear elastic and isotropic material.

### Syntax:

$D = \text{hooke}(\text{ptype}, E, \nu)$

### Description:

`hooke` computes the material matrix  $D$  for a linear elastic and isotropic material.

The variable `ptype` is used to define the type of analysis.

$$\text{ptype} = \begin{cases} 1 & \text{plane stress.} \\ 2 & \text{plane strain.} \\ 3 & \text{axisymmetry.} \\ 4 & \text{three dimensional analysis.} \end{cases}$$

The material parameters  $E$  and  $\nu$  define the modulus of elasticity  $E$  and the Poisson's ratio  $\nu$ , respectively.

For plane stress, `ptype=1`,  $D$  is formed as

$$D = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1 - \nu}{2} \end{bmatrix}$$

For plane strain, `ptype=2` and axisymmetry, `ptype=3`,  $D$  is formed as

$$D = \frac{E}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} 1 - \nu & \nu & \nu & 0 \\ \nu & 1 - \nu & \nu & 0 \\ \nu & \nu & 1 - \nu & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) \end{bmatrix}$$

For the three dimensional case, `ptype=4`,  $D$  is formed as

$$D = \frac{E}{(1 + \nu)(1 - 2\nu)} \begin{bmatrix} 1 - \nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 - \nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 - \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1 - 2\nu) \end{bmatrix}$$

**Purpose:**

Compute stresses and plastic strains for an elasto-plastic isotropic hardening von Mises material.

**Syntax:**

`[es,deps,st]=mises(ptype,mp,est,st)`

**Description:**

**mises** computes updated stresses **es**, plastic strain increments **deps**, and state variables **st** for an elasto-plastic isotropic hardening von Mises material.

The input variable **ptype** is used to define the type of analysis, cf. **hooke**. The vector **mp** contains the material constants

$$\mathbf{mp} = [ E \ \nu \ h ]$$

where  $E$  is the modulus of elasticity,  $\nu$  is the Poisson's ratio, and  $h$  is the plastic modulus. The input matrix **est** contains trial stresses obtained by using the elastic material matrix **D** in **plants** or some similar **s**-function, and the input vector **st** contains the state parameters

$$\mathbf{st} = [ yi \ \sigma_y \ \epsilon_{eff}^p ]$$

at the beginning of the step. The scalar  $yi$  states whether the material behaviour is elasto-plastic ( $yi=1$ ), or elastic ( $yi=0$ ). The current yield stress is denoted by  $\sigma_y$  and the effective plastic strain by  $\epsilon_{eff}^p$ .

The output variables **es** and **st** contain updated values of **es** and **st** obtained by integration of the constitutive equations over the actual displacement step. The increments of the plastic strains are stored in the vector **deps**.

If **es** and **st** contain more than one row, then every row will be treated by the command.

**Note:**

It is not necessary to check whether the material behaviour is elastic or elasto-plastic, this test is done by the function. The computation is based on an Euler-Backward method, i.e. the radial return method.

Only the cases **ptype**=2, 3 and 4, are implemented.

## dmises

---

### Purpose:

Form the elasto-plastic continuum matrix for an isotropic hardening von Mises material.

### Syntax:

$D = \text{dmises}(\text{ptype}, \text{mp}, \text{es}, \text{st})$

### Description:

**dmises** forms the elasto-plastic continuum matrix for an isotropic hardening von Mises material.

The input variable **ptype** is used to define the type of analysis, cf. **hooke**. The vector **mp** contains the material constants

$$\text{mp} = [ E \ \nu \ h ]$$

where  $E$  is the modulus of elasticity,  $\nu$  is the Poisson's ratio, and  $h$  is the plastic modulus. The matrix **es** contains current stresses obtained from **plants** or some similar **s**-function, and the vector **st** contains the current state parameters

$$\text{st} = [ yi \ \sigma_y \ \epsilon_{eff}^p ]$$

where  $yi=1$  if the material behaviour is elasto-plastic, and  $yi=0$  if the material behaviour is elastic. The current yield stress is denoted by  $\sigma_y$ , and the current effective plastic strain by  $\epsilon_{eff}^p$ .

### Note:

Only the case **ptype**=2 is implemented.

## 5 Element functions

### 5.1 Introduction

The group of element functions contains functions for computation of element matrices and element forces for different element types. The element functions have been divided into the following groups

<b>Spring element</b>
<b>Bar elements</b>
<b>Heat flow elements</b>
<b>Solid elements</b>
<b>Beam elements</b>
<b>Plate element</b>

For each element type there is a function for computation of the element stiffness matrix  $\mathbf{K}^e$ . For most of the elements, an element load vector  $\mathbf{f}^e$  can also be computed. These functions are identified by their last letter -e.

Using the function `assem`, the element stiffness matrices and element load vectors are assembled into a global stiffness matrix  $\mathbf{K}$  and a load vector  $\mathbf{f}$ . Unknown nodal values of temperatures or displacements  $\mathbf{a}$  are computed by solving the system of equations  $\mathbf{K}\mathbf{a} = \mathbf{f}$  using the function `solveq`. A vector of nodal values of temperatures or displacements for a specific element is formed by the function `extract`.

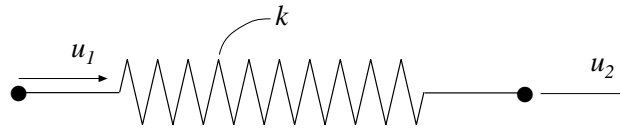
When the element nodal values have been computed, the element flux or element stresses can be calculated using functions specific to the element type concerned. These functions are identified by their last letter -s.

For some elements, a function for computing the internal force vector is also available. These functions are identified by their last letter -f.



## 5.2 Spring element

The spring element, shown below, can be used for the analysis of one-dimensional spring systems and for a variety of analogous physical problems.



Quantities corresponding to the variables of the spring are listed in Table 1.

Problem type	Spring stiffness	Nodal displacement	Element force	Spring force
Spring	$k$	$u$	$P$	$N$
Bar	$\frac{EA}{L}$	$u$	$P$	$N$
Thermal conduction	$\frac{\lambda A}{L}$	$T$	$\bar{H}$	$H$
Electrical circuit	$\frac{1}{R}$	$U$	$\bar{I}$	$I$
Groundwater flow	$\frac{kA}{L}$	$\phi$	$\bar{H}$	$H$
Pipe network	$\frac{\pi D^4}{128\mu L}$	$p$	$\bar{H}$	$H$

Table 1: *Analogous quantities*

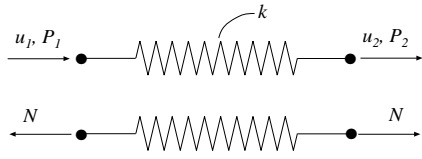
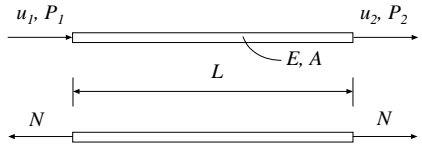
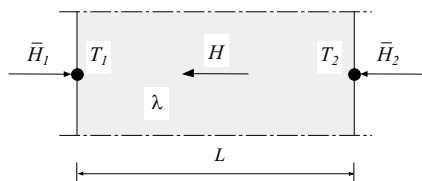
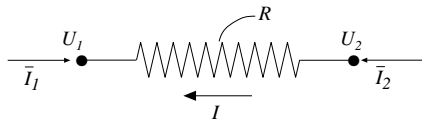
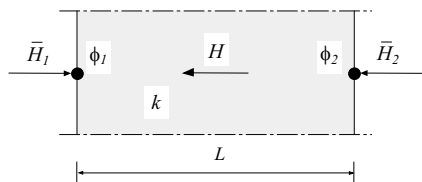
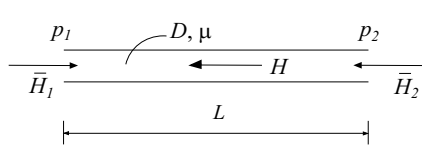
Interpretations of the spring element			
Problem type	Quantities	Designations	
Spring		$k$ $u$ $P$ $N$	spring stiffness displacement element force spring force
Bar		$L$ $E$ $A$ $u$ $P$ $N$	length modulus of elasticity area of cross section displacement element force normal force
Thermal conduction		$L$ $\lambda$ $T$ $\bar{H}$ $H$	length thermal conductivity temperature element heat flow internal heat flow
Electrical circuit		$R$ $U$ $\bar{I}$ $I$	resistance potential element current internal current
Ground-water flow		$L$ $k$ $\phi$ $\bar{H}$ $H$	length permeability piezometric head element water flow internal water flow
Pipe network (laminar flow)		$L$ $D$ $\mu$ $p$ $\bar{H}$ $H$	length pipe diameter viscosity pressure element fluid flow internal fluid flow

Table 2: Quantities used in different types of problems

---

The following functions are available for the spring element:

<b>Spring functions</b>	
<b>spring1e</b>	Compute element matrix
<b>spring1s</b>	Compute spring force

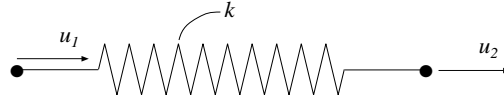


**spring1e****Spring element**

---

**Purpose:**

Compute element stiffness matrix for a spring element.

**Syntax:**

`Ke=spring1e(ep)`

**Description:**

`spring1e` provides the element stiffness matrix **Ke** for a spring element.

The input variable

$$\mathbf{ep} = [ k ]$$

supplies the spring stiffness  $k$  or the analog quantity defined in Table 1.

**Theory:**

The element stiffness matrix  $\mathbf{K}^e$ , stored in **Ke**, is computed according to

$$\mathbf{K}^e = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix}$$

where  $k$  is defined by **ep**.

**Spring element****spring1s**

---

**Purpose:**

Compute spring force in a spring element.

**Syntax:**

```
es=spring1s(ep,ed)
```

**Description:**

`spring1s` computes the spring force `es` in a spring element.

The input variable `ep` is defined in `spring1e` and the element nodal displacements `ed` are obtained by the function `extract`.

The output variable

$$\mathbf{es} = [ N ]$$

contains the spring force  $N$ , or the analog quantity.

**Theory:**

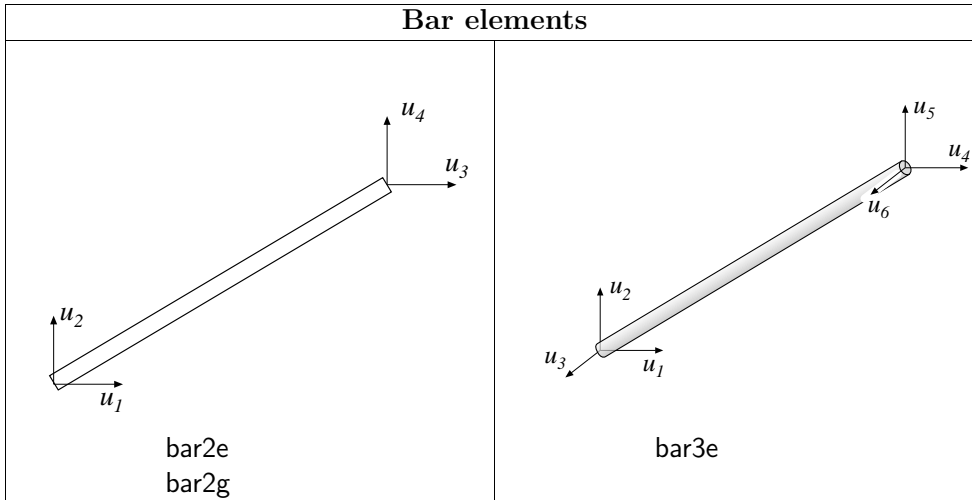
The spring force  $N$ , or analog quantity, is computed according to

$$N = k [ u_2 - u_1 ]$$



### 5.3 Bar elements

Bar elements are available for one, two, and three dimensional analysis. For the one dimensional element, see the spring element.



Two dimensional bar functions	
bar2e	Compute element matrix
bar2g	Compute element matrix for geometric nonlinear element
bar2s	Compute normal force

Three dimensional bar functions	
bar3e	Compute element matrix
bar3s	Compute normal force

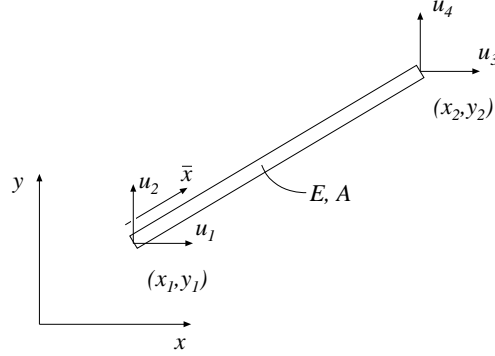
## bar2e

## Two dimensional bar element

---

### Purpose:

Compute element stiffness matrix for a two dimensional bar element.



### Syntax:

$\mathbf{K}e = \text{bar2e}(ex, ey, ep)$

### Description:

bar2e provides the global element stiffness matrix  $\mathbf{K}e$  for a two dimensional bar element.

The input variables

$$\begin{aligned} ex &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} & ep &= \begin{bmatrix} E & A \end{bmatrix} \\ ey &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \end{aligned}$$

supply the element nodal coordinates  $x_1, y_1, x_2$ , and  $y_2$ , the modulus of elasticity  $E$ , and the cross section area  $A$ .

### Theory:

The element stiffness matrix  $\mathbf{K}^e$ , stored in  $\mathbf{K}e$ , is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & 0 & 0 \\ 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} \end{bmatrix}$$

The transformation matrix  $\mathbf{G}$  contains the direction cosines

$$n_{x\bar{x}} = \frac{x_2 - x_1}{L} \quad n_{y\bar{x}} = \frac{y_2 - y_1}{L}$$

where the length

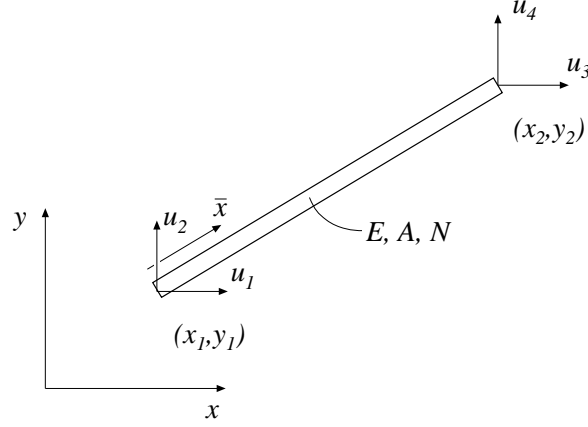
$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

## Two dimensional bar element

bar2g

### Purpose:

Compute element stiffness matrix for a two dimensional geometric nonlinear bar.



### Syntax:

$\mathbf{K}_e = \text{bar2g}(\text{ex}, \text{ey}, \text{ep}, N)$

### Description:

bar2g provides the element stiffness matrix  $\mathbf{K}_e$  for a two dimensional geometric non-linear bar element.

The input variables **ex**, **ey** and **ep** are described in **bar2e**. The input variable

$$N = [ N ]$$

contains the value of the normal force, which is positive in tension.

### Theory:

The global element stiffness matrix  $\mathbf{K}^e$ , stored in  $\mathbf{K}_e$ , is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \frac{EA}{L} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \frac{N}{L} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & 0 & 0 \\ n_{x\bar{y}} & n_{y\bar{y}} & 0 & 0 \\ 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} \\ 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} \end{bmatrix}$$

**bar2g****Two dimensional bar element**

---

The transformation matrix **G** contains the direction cosines

$$n_{x\bar{x}} = n_{y\bar{y}} = \frac{x_2 - x_1}{L} \quad n_{y\bar{x}} = -n_{x\bar{y}} = \frac{y_2 - y_1}{L}$$

where the length

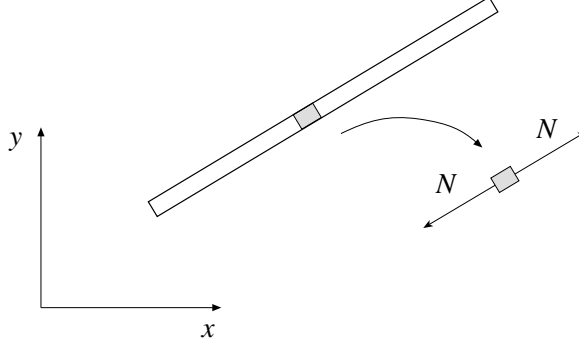
$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

## Two dimensional bar element

bar2s

### Purpose:

Compute normal force in a two dimensional bar element.



### Syntax:

```
es=bar2s(ex,ey,ep,ed)
```

### Description:

**bar2s** computes the normal force in the two dimensional bar elements **bar2e** and **bar2g**.

The input variables **ex**, **ey**, and **ep** are defined in **bar2e** and the element nodal displacements, stored in **ed**, are obtained by the function **extract**.

The output variable

$$\mathbf{es} = [ N ]$$

contains the normal force  $N$ .

### Theory:

The normal force  $N$  is computed from

$$N = \frac{EA}{L} [ -1 \quad 1 ] \mathbf{G} \mathbf{a}^e$$

where  $E$ ,  $A$ ,  $L$ , and the transformation matrix  $\mathbf{G}$  are defined in **bar2e**. The nodal displacements in global coordinates

$$\mathbf{a}^e = [ u_1 \quad u_2 \quad u_3 \quad u_4 ]^T$$

are also shown in **bar2e**. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.

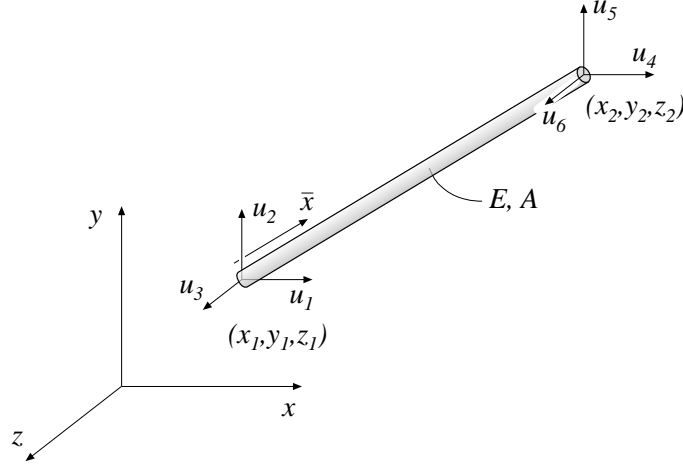


bar3e

Three dimensional bar element

**Purpose:**

Compute element stiffness matrix for a three dimensional bar element.



**Syntax:**

`Ke=bar3e(ex,ey,ez,ep)`

**Description:**

bar3e provides the element stiffness matrix  $\mathbf{K}^e$  for a three dimensional bar element.

The input variables

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \\ \mathbf{ez} &= \begin{bmatrix} z_1 & z_2 \end{bmatrix} \end{aligned} \quad \mathbf{ep} = \begin{bmatrix} E & A \end{bmatrix}$$

supply the element nodal coordinates  $x_1, y_1, z_1, x_2$  etc, the modulus of elasticity  $E$ , and the cross section area  $A$ .

**Theory:**

The global element stiffness matrix  $\mathbf{K}^e$  is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} \end{bmatrix}$$

The transformation matrix  $\mathbf{G}$  contains the direction cosines

$$n_{x\bar{x}} = \frac{x_2 - x_1}{L} \quad n_{y\bar{x}} = \frac{y_2 - y_1}{L} \quad n_{z\bar{x}} = \frac{z_2 - z_1}{L}$$

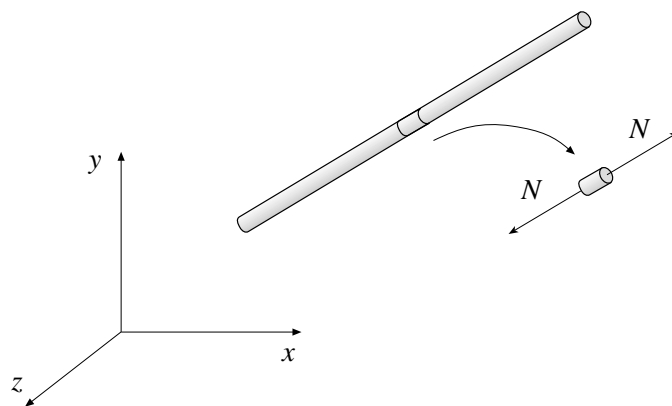
where the length  $L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ .

## Three dimensional bar element

bar3s

### Purpose:

Compute normal force in a three dimensional bar element.



### Syntax:

```
es=bar3s(ex,ey,ez,ep,ed)
```

### Description:

**bar3s** computes the normal force in a three dimensional bar element.

The input variables **ex**, **ey**, **ez**, and **ep** are defined in **bar3e**, and the element nodal displacements, stored in **ed**, are obtained by the function **extract**.

The output variable

$$\mathbf{es} = [ N ]$$

contains the normal force  $N$  of the bar.

### Theory:

The normal force  $N$  is computed from

$$N = \frac{EA}{L} [ -1 \quad 1 ] \mathbf{G} \mathbf{a}^e$$

where  $E$ ,  $A$ ,  $L$ , and the transformation matrix  $\mathbf{G}$  are defined in **bar3e**. The nodal displacements in global coordinates

$$\mathbf{a}^e = [ u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 ]^T$$

are also shown in **bar3e**. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.



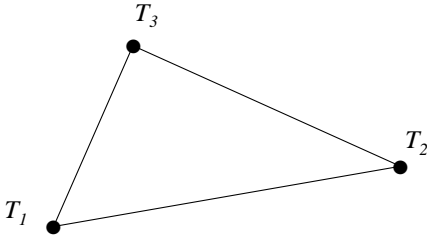
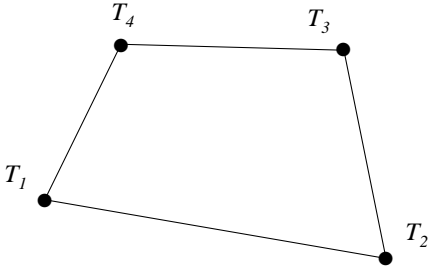
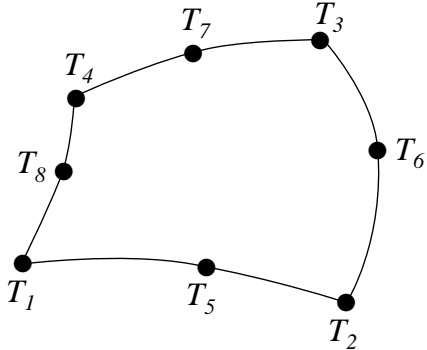
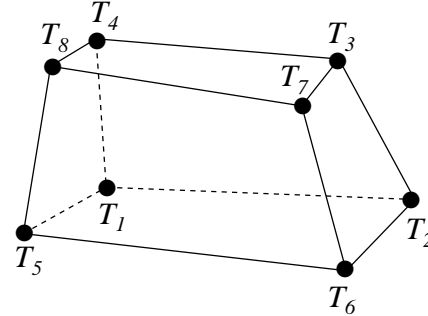
## 5.4 Heat flow elements

Heat flow elements are available for one, two, and three dimensional analysis. For one dimensional heat flow the spring element **spring1** is used.

A variety of important physical phenomena are described by the same differential equation as the heat flow problem. The heat flow element is thus applicable in modelling different physical applications. Table 3 below shows the relation between the primary variable **a**, the constitutive matrix **D**, and the load vector **f<sub>l</sub>** for a chosen set of two dimensional physical problems.

Problem type	<b>a</b>	<b>D</b>	<b>f<sub>l</sub></b>	Designation
Heat flow	$T$	$\lambda_x, \lambda_y$	$Q$	$T$ = temperature $\lambda_x, \lambda_y$ = thermal conductivity $Q$ = heat supply
Groundwater flow	$\phi$	$k_x, k_y$	$Q$	$\phi$ = piezometric head $k_x, k_y$ = permeabilities $Q$ = fluid supply
St. Venant torsion	$\phi$	$\frac{1}{G_{zy}}, \frac{1}{G_{zx}}$	$2\Theta$	$\phi$ = stress function $G_{zy}, G_{zx}$ = shear moduli $\Theta$ = angle of torsion per unit length

Table 3: *Problem dependent parameters*

Heat flow elements	
 <p>flw2te</p>	 <p>flw2qe flw2i4e</p>
 <p>flw2i8e</p>	 <p>flw3i8e</p>

2D heat flow functions	
flw2te	Compute element matrices for a triangular element
flw2ts	Compute temperature gradients and flux
flw2qe	Compute element matrices for a quadrilateral element
flw2qs	Compute temperature gradients and flux
flw2i4e	Compute element matrices, 4 node isoparametric element
flw2i4s	Compute temperature gradients and flux
flw2i8e	Compute element matrices, 8 node isoparametric element
flw2i8s	Compute temperature gradients and flux

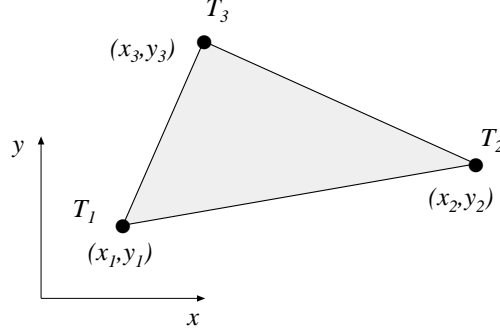
3D heat flow functions	
flw3i8e	Compute element matrices, 8 node isoparametric element
flw3i8s	Compute temperature gradients and flux

## Two dimensional heat flow elements

flw2te

### Purpose:

Compute element stiffness matrix for a triangular heat flow element.



### Syntax:

```
Ke=flw2te(ex,ey,ep,D)
[Ke,fe]=flw2te(ex,ey,ep,D,eq)
```

### Description:

flw2te provides the element stiffness (conductivity) matrix **Ke** and the element load vector **fe** for a triangular heat flow element.

The element nodal coordinates  $x_1, y_1, x_2$  etc, are supplied to the function by **ex** and **ey**, the element thickness  $t$  is supplied by **ep** and the thermal conductivities (or corresponding quantities)  $k_{xx}, k_{xy}$  etc are supplied by **D**.

$$\begin{aligned} \mathbf{ex} &= [x_1 \ x_2 \ x_3] \\ \mathbf{ey} &= [y_1 \ y_2 \ y_3] \end{aligned} \quad \mathbf{ep} = [t] \quad \mathbf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable **eq** is given in the function, the element load vector **fe** is computed, using

$$\mathbf{eq} = [Q]$$

where  $Q$  is the heat supply per unit volume.

### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in **Ke** and **fe**, respectively, are computed according to

$$\begin{aligned} \mathbf{K}^e &= (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \mathbf{D} \bar{\mathbf{B}} t dA \ \mathbf{C}^{-1} \\ \mathbf{f}_l^e &= (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{N}}^T Q t dA \end{aligned}$$

with the constitutive matrix **D** defined by **D**.

**flw2te**

**Two dimensional heat flow elements**

The evaluation of the integrals for the triangular element is based on the linear temperature approximation  $T(x, y)$  and is expressed in terms of the nodal variables  $T_1$ ,  $T_2$  and  $T_3$  as

$$T(x, y) = \mathbf{N}^e \mathbf{a}^e = \bar{\mathbf{N}} \mathbf{C}^{-1} \mathbf{a}^e$$

where

$$\bar{\mathbf{N}} = [1 \ x \ y] \quad \mathbf{C} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

and hence it follows that

$$\bar{\mathbf{B}} = \nabla \bar{\mathbf{N}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix}$$

Evaluation of the integrals for the triangular element yields

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \bar{\mathbf{B}}^T \mathbf{D} \bar{\mathbf{B}} \mathbf{C}^{-1} t A$$

$$\mathbf{f}_l^e = \frac{QAt}{3} [1 \ 1 \ 1]^T$$

where the element area  $A$  is determined as

$$A = \frac{1}{2} \det \mathbf{C}$$

**Two dimensional heat flow elements****flw2ts****Purpose:**

Compute heat flux and temperature gradients in a triangular heat flow element.

**Syntax:**

`[es,et]=flw2ts(ex,ey,D,ed)`

**Description:**

`flw2ts` computes the heat flux vector `es` and the temperature gradient `et` (or corresponding quantities) in a triangular heat flow element.

The input variables `ex`, `ey` and the matrix `D` are defined in `flw2te`. The vector `ed` contains the nodal temperatures  $\mathbf{a}^e$  of the element and is obtained by the function `extract` as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [T_1 \ T_2 \ T_3]$$

The output variables

$$\mathbf{es} = \mathbf{q}^T = [q_x \ q_y]$$

$$\mathbf{et} = (\nabla T)^T = \left[ \frac{\partial T}{\partial x} \ \frac{\partial T}{\partial y} \right]$$

contain the components of the heat flux and the temperature gradient computed in the directions of the coordinate axis.

**Theory:**

The temperature gradient and the heat flux are computed according to

$$\nabla T = \bar{\mathbf{B}} \mathbf{C}^{-1} \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D} \nabla T$$

where the matrices `D`, `B̄`, and `C` are described in `flw2te`. Note that both the temperature gradient and the heat flux are constant in the element.



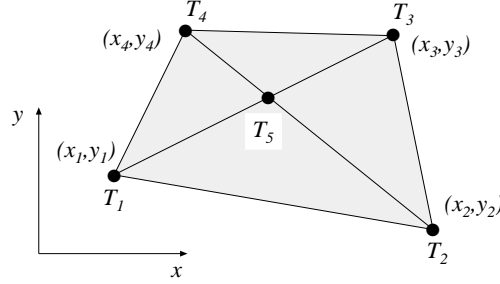
**flw2qe**

**Two dimensional heat flow elements**

---

**Purpose:**

Compute element stiffness matrix for a quadrilateral heat flow element.



**Syntax:**

```
Ke=flw2qe(ex,ey,ep,D)
[Ke,fe]=flw2qe(ex,ey,ep,D,eq)
```

**Description:**

flw2qe provides the element stiffness (conductivity) matrix **Ke** and the element load vector **fe** for a quadrilateral heat flow element.

The element nodal coordinates  $x_1, y_1, x_2$  etc, are supplied to the function by **ex** and **ey**, the element thickness  $t$  is supplied by **ep** and the thermal conductivities (or corresponding quantities)  $k_{xx}, k_{xy}$  etc are supplied by **D**.

$$\begin{aligned} \mathbf{ex} &= [x_1 \ x_2 \ x_3 \ x_4] \\ \mathbf{ey} &= [y_1 \ y_2 \ y_3 \ y_4] \end{aligned} \quad \mathbf{ep} = [t] \quad \mathbf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable **eq** is given in the function, the element load vector **fe** is computed, using

$$\mathbf{eq} = [Q]$$

where  $Q$  is the heat supply per unit volume.

**Theory:**

In computing the element matrices, a fifth degree of freedom is introduced. The location of this extra degree of freedom is defined by the mean value of the coordinates in the corner points. Four sets of element matrices are calculated using **flw2te**. These matrices are then assembled and the fifth degree of freedom is eliminated by static condensation.

**Two dimensional heat flow elements****flw2qs****Purpose:**

Compute heat flux and temperature gradients in a quadrilateral heat flow element.

**Syntax:**

```
[es,et]=flw2qs(ex,ey,ep,D,ed)
[es,et]=flw2qs(ex,ey,ep,D,ed,eq)
```

**Description:**

flw2qs computes the heat flux vector **es** and the temperature gradient **et** (or corresponding quantities) in a quadrilateral heat flow element.

The input variables **ex**, **ey**, **eq** and the matrix **D** are defined in flw2qe. The vector **ed** contains the nodal temperatures  $\mathbf{a}^e$  of the element and is obtained by the function extract as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [T_1 \ T_2 \ T_3 \ T_4]$$

The output variables

$$\mathbf{es} = \mathbf{q}^T = [q_x \ q_y]$$

$$\mathbf{et} = (\nabla T)^T = \begin{bmatrix} \frac{\partial T}{\partial x} & \frac{\partial T}{\partial y} \end{bmatrix}$$

contain the components of the heat flux and the temperature gradient computed in the directions of the coordinate axis.

**Theory:**

By assembling four triangular elements as described in flw2te a system of equations containing 5 degrees of freedom is obtained. From this system of equations the unknown temperature at the center of the element is computed. Then according to the description in flw2ts the temperature gradient and the heat flux in each of the four triangular elements are produced. Finally the temperature gradient and the heat flux of the quadrilateral element are computed as area weighted mean values from the values of the four triangular elements. If heat is supplied to the element, the element load vector **eq** is needed for the calculations.

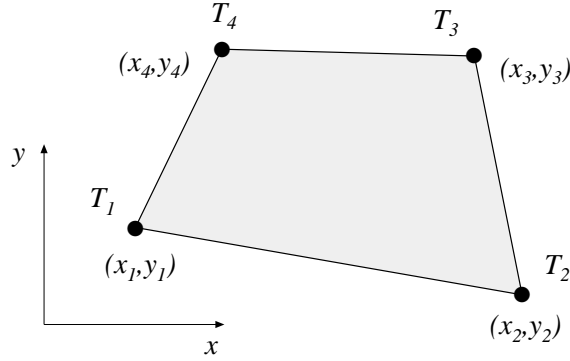
## flw2i4e

## Two dimensional heat flow elements

---

### Purpose:

Compute element stiffness matrix for a 4 node isoparametric heat flow element.



### Syntax:

```
Ke=flw2i4e(ex,ey,ep,D)
[Ke,fe]=flw2i4e(ex,ey,ep,D,eq)
```

### Description:

flw2i4e provides the element stiffness (conductivity) matrix **Ke** and the element load vector **fe** for a 4 node isoparametric heat flow element.

The element nodal coordinates  $x_1, y_1, x_2$  etc, are supplied to the function by **ex** and **ey**. The element thickness  $t$  and the number of Gauss points  $n$

$(n \times n)$  integration points,  $n = 1, 2, 3$

are supplied to the function by **ep** and the thermal conductivities (or corresponding quantities)  $k_{xx}, k_{xy}$  etc are supplied by **D**.

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} \end{aligned} \quad \mathbf{ep} = \begin{bmatrix} t & n \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable **eq** is given in the function, the element load vector **fe** is computed, using

$$\mathbf{eq} = \begin{bmatrix} Q \end{bmatrix}$$

where  $Q$  is the heat supply per unit volume.

**Two dimensional heat flow elements****flw2i4e****Theory:**

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{Ke}$  and  $\mathbf{fe}$ , respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e t dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} Q t dA$$

with the constitutive matrix  $\mathbf{D}$  defined by  $D$ .

The evaluation of the integrals for the isoparametric 4 node element is based on a temperature approximation  $T(\xi, \eta)$ , expressed in a local coordinates system in terms of the nodal variables  $T_1, T_2, T_3$  and  $T_4$  as

$$T(\xi, \eta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{N}^e = [N_1^e \ N_2^e \ N_3^e \ N_4^e] \quad \mathbf{a}^e = [T_1 \ T_2 \ T_3 \ T_4]^T$$

The element shape functions are given by

$$N_1^e = \frac{1}{4}(1 - \xi)(1 - \eta) \quad N_2^e = \frac{1}{4}(1 + \xi)(1 - \eta)$$

$$N_3^e = \frac{1}{4}(1 + \xi)(1 + \eta) \quad N_4^e = \frac{1}{4}(1 - \xi)(1 + \eta)$$

The  $\mathbf{B}^e$ -matrix is given by

$$\mathbf{B}^e = \nabla \mathbf{N}^e = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \mathbf{N}^e = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \mathbf{N}^e$$

where  $\mathbf{J}$  is the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

**flw2i4s**
**Two dimensional heat flow elements**


---

**Purpose:**

Compute heat flux and temperature gradients in a 4 node isoparametric heat flow element.

**Syntax:**

`[es,et,eci]=flw2i4s(ex,ey,ep,D,ed)`

**Description:**

`flw2i4s` computes the heat flux vector `es` and the temperature gradient `et` (or corresponding quantities) in a 4 node isoparametric heat flow element.

The input variables `ex`, `ey`, `ep` and the matrix `D` are defined in `flw2i4e`. The vector `ed` contains the nodal temperatures  $\mathbf{a}^e$  of the element and is obtained by `extract` as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [T_1 \ T_2 \ T_3 \ T_4]$$

The output variables

$$\mathbf{es} = \bar{\mathbf{q}}^T = \begin{bmatrix} q_x^1 & q_y^1 \\ q_x^2 & q_y^2 \\ \vdots & \vdots \\ q_x^{n^2} & q_y^{n^2} \end{bmatrix}$$

$$\mathbf{et} = (\bar{\nabla}T)^T = \begin{bmatrix} \frac{\partial T^1}{\partial x} & \frac{\partial T^1}{\partial y} \\ \frac{\partial T^2}{\partial x} & \frac{\partial T^2}{\partial y} \\ \vdots & \vdots \\ \frac{\partial T^{n^2}}{\partial x} & \frac{\partial T^{n^2}}{\partial y} \end{bmatrix} \quad \mathbf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the heat flux, the temperature gradient, and the coordinates of the integration points. The index  $n$  denotes the number of integration points used within the element, cf. `flw2i4e`.

**Theory:**

The temperature gradient and the heat flux are computed according to

$$\nabla T = \mathbf{B}^e \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D} \nabla T$$

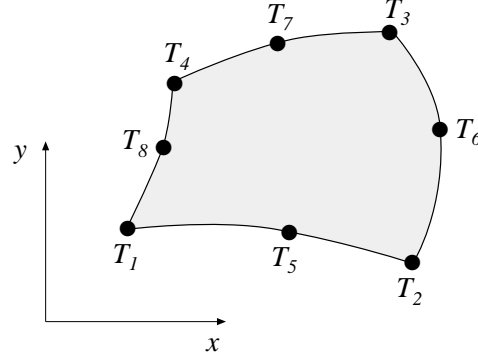
where the matrices  $\mathbf{D}$ ,  $\mathbf{B}^e$ , and  $\mathbf{a}^e$  are described in `flw2i4e`, and where the integration points are chosen as evaluation points.

## Two dimensional heat flow elements

flw2i8e

### Purpose:

Compute element stiffness matrix for an 8 node isoparametric heat flow element.



### Syntax:

```
Ke=flw2i8e(ex,ey,ep,D)
[Ke,fe]=flw2i8e(ex,ey,ep,D,eq)
```

### Description:

flw2i8e provides the element stiffness (conductivity) matrix **Ke** and the element load vector **fe** for an 8 node isoparametric heat flow element.

The element nodal coordinates  $x_1, y_1, x_2$  etc, are supplied to the function by **ex** and **ey**. The element thickness  $t$  and the number of Gauss points  $n$

$(n \times n)$  integration points,  $n = 1, 2, 3$

are supplied to the function by **ep** and the thermal conductivities (or corresponding quantities)  $k_{xx}, k_{xy}$  etc are supplied by **D**.

$$\begin{aligned} \mathbf{ex} &= [x_1 \ x_2 \ x_3 \ \dots \ x_8] \\ \mathbf{ey} &= [y_1 \ y_2 \ y_3 \ \dots \ y_8] \end{aligned} \quad \mathbf{ep} = [t \ n] \quad \mathbf{D} = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix}$$

If the scalar variable **eq** is given in the function, the vector **fe** is computed, using

$$\mathbf{eq} = [Q]$$

where  $Q$  is the heat supply per unit volume.

**Theory:**

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{Ke}$  and  $\mathbf{fe}$ , respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e t dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} Q t dA$$

with the constitutive matrix  $\mathbf{D}$  defined by  $\mathbf{D}$ .

The evaluation of the integrals for the 2D isoparametric 8 node element is based on a temperature approximation  $T(\xi, \eta)$ , expressed in a local coordinates system in terms of the nodal variables  $T_1$  to  $T_8$  as

$$T(\xi, \eta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{N}^e = [ N_1^e \ N_2^e \ N_3^e \ \dots \ N_8^e ] \quad \mathbf{a}^e = [ T_1 \ T_2 \ T_3 \ \dots \ T_8 ]^T$$

The element shape functions are given by

$$\begin{aligned} N_1^e &= -\frac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta) & N_5^e &= \frac{1}{2}(1-\xi^2)(1-\eta) \\ N_2^e &= -\frac{1}{4}(1+\xi)(1-\eta)(1-\xi+\eta) & N_6^e &= \frac{1}{2}(1+\xi)(1-\eta^2) \\ N_3^e &= -\frac{1}{4}(1+\xi)(1+\eta)(1-\xi-\eta) & N_7^e &= \frac{1}{2}(1-\xi^2)(1+\eta) \\ N_4^e &= -\frac{1}{4}(1-\xi)(1+\eta)(1+\xi-\eta) & N_8^e &= \frac{1}{2}(1-\xi)(1-\eta^2) \end{aligned}$$

The  $\mathbf{B}^e$ -matrix is given by

$$\mathbf{B}^e = \nabla \mathbf{N}^e = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \mathbf{N}^e = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \mathbf{N}^e$$

where  $\mathbf{J}$  is the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

---

Two dimensional heat flow elements

flw2i8s

---

**Purpose:**

Compute heat flux and temperature gradients in an 8 node isoparametric heat flow element.

**Syntax:**

[es,et,eci]=flw2i8s(ex,ey,ep,D,ed)

**Description:**

flw2i8s computes the heat flux vector **es** and the temperature gradient **et** (or corresponding quantities) in an 8 node isoparametric heat flow element.

The input variables **ex**, **ey**, **ep** and the matrix **D** are defined in flw2i8e. The vector **ed** contains the nodal temperatures  $\mathbf{a}^e$  of the element and is obtained by the function extract as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [T_1 \ T_2 \ T_3 \ \dots \ T_8]$$

The output variables

$$\mathbf{es} = \bar{\mathbf{q}}^T = \begin{bmatrix} q_x^1 & q_y^1 \\ q_x^2 & q_y^2 \\ \vdots & \vdots \\ q_x^{n^2} & q_y^{n^2} \end{bmatrix}$$

$$\mathbf{et} = (\bar{\nabla}T)^T = \begin{bmatrix} \frac{\partial T^1}{\partial x} & \frac{\partial T^1}{\partial y} \\ \frac{\partial T^2}{\partial x} & \frac{\partial T^2}{\partial y} \\ \vdots & \vdots \\ \frac{\partial T^{n^2}}{\partial x} & \frac{\partial T^{n^2}}{\partial y} \end{bmatrix}$$

$$\mathbf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the heat flux, the temperature gradient, and the coordinates of the integration points. The index  $n$  denotes the number of integration points used within the element, cf. flw2i8e.

**Theory:**

The temperature gradient and the heat flux are computed according to

$$\nabla T = \mathbf{B}^e \mathbf{a}^e$$

$$\mathbf{q} = -\mathbf{D}\nabla T$$

where the matrices **D**, **B<sup>e</sup>**, and **a<sup>e</sup>** are described in flw2i8e, and where the integration points are chosen as evaluation points.

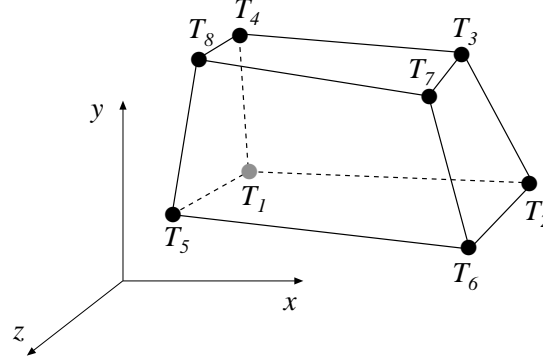


**flw3i8e**
**Three dimensional heat flow elements**


---

**Purpose:**

Compute element stiffness matrix for an 8 node isoparametric element.


**Syntax:**

```
Ke=flw3i8e(ex,ey,ez,ep,D)
[Ke,fe]=flw3i8e(ex,ey,ez,ep,D,eq)
```

**Description:**

flw3i8e provides the element stiffness (conductivity) matrix **Ke** and the element load vector **fe** for an 8 node isoparametric heat flow element.

The element nodal coordinates  $x_1, y_1, z_1, x_2$  etc, are supplied to the function by **ex**, **ey** and **ez**. The number of Gauss points  $n$

$(n \times n \times n)$  integration points,  $n = 1, 2, 3$

are supplied to the function by **ep** and the thermal conductivities (or corresponding quantities)  $k_{xx}, k_{xy}$  etc are supplied by **D**.

$$\begin{aligned} \mathbf{ex} &= [x_1 \ x_2 \ x_3 \ \dots \ x_8] \\ \mathbf{ey} &= [y_1 \ y_2 \ y_3 \ \dots \ y_8] \\ \mathbf{ez} &= [z_1 \ z_2 \ z_3 \ \dots \ z_8] \end{aligned} \quad \mathbf{ep} = [n] \quad \mathbf{D} = \begin{bmatrix} k_{xx} & k_{xy} & k_{xz} \\ k_{yx} & k_{yy} & k_{yz} \\ k_{zx} & k_{zy} & k_{zz} \end{bmatrix}$$

If the scalar variable **eq** is given in the function, the element load vector **fe** is computed, using

$$\mathbf{eq} = [Q]$$

where  $Q$  is the heat supply per unit volume.

### Three dimensional heat flow elements

flw3i8e

#### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{Ke}$  and  $\mathbf{fe}$ , respectively, are computed according to

$$\mathbf{K}^e = \int_V \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e dV$$

$$\mathbf{f}_l^e = \int_V \mathbf{N}^{eT} Q dV$$

with the constitutive matrix  $\mathbf{D}$  defined by  $\mathbf{D}$ .

The evaluation of the integrals for the 3D isoparametric 8 node element is based on a temperature approximation  $T(\xi, \eta, \zeta)$ , expressed in a local coordinates system in terms of the nodal variables  $T_1$  to  $T_8$  as

$$T(\xi, \eta, \zeta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{N}^e = [ N_1^e \ N_2^e \ N_3^e \ \dots \ N_8^e ] \quad \mathbf{a}^e = [ T_1 \ T_2 \ T_3 \ \dots \ T_8 ]^T$$

The element shape functions are given by

$$\begin{aligned} N_1^e &= \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta) & N_2^e &= \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta) \\ N_3^e &= \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta) & N_4^e &= \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta) \\ N_5^e &= \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta) & N_6^e &= \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta) \\ N_7^e &= \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta) & N_8^e &= \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta) \end{aligned}$$

The  $\mathbf{B}^e$ -matrix is given by

$$\mathbf{B}^e = \nabla \mathbf{N}^e = \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{bmatrix} \mathbf{N}^e = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{bmatrix} \mathbf{N}^e$$

where  $\mathbf{J}$  is the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}$$

Evaluation of the integrals is done by Gauss integration.

**flw3i8s**
**Three dimensional heat flow elements**


---

**Purpose:**

Compute heat flux and temperature gradients in an 8 node isoparametric heat flow element.

**Syntax:**

[es,et,eci]=flw3i8s(ex,ey,ez,ep,D,ed)

**Description:**

flw3i8s computes the heat flux vector **es** and the temperature gradient **et** (or corresponding quantities) in an 8 node isoparametric heat flow element.

The input variables **ex**, **ey**, **ez**, **ep** and the matrix **D** are defined in flw3i8e. The vector **ed** contains the nodal temperatures  $\mathbf{a}^e$  of the element and is obtained by the function extract as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [T_1 \ T_2 \ T_3 \ \dots \ T_8]$$

The output variables

$$\mathbf{es} = \bar{\mathbf{q}}^T = \begin{bmatrix} q_x^1 & q_y^1 & q_z^1 \\ q_x^2 & q_y^2 & q_z^2 \\ \vdots & \vdots & \vdots \\ q_x^{n^3} & q_y^{n^3} & q_z^{n^3} \end{bmatrix}$$

$$\mathbf{et} = (\bar{\nabla}T)^T = \begin{bmatrix} \frac{\partial T^1}{\partial x} & \frac{\partial T^1}{\partial y} & \frac{\partial T^1}{\partial z} \\ \frac{\partial T^2}{\partial x} & \frac{\partial T^2}{\partial y} & \frac{\partial T^2}{\partial z} \\ \vdots & \vdots & \vdots \\ \frac{\partial T^{n^3}}{\partial x} & \frac{\partial T^{n^3}}{\partial y} & \frac{\partial T^{n^3}}{\partial z} \end{bmatrix}$$

$$\mathbf{eci} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{n^3} & y_{n^3} & z_{n^3} \end{bmatrix}$$

contain the heat flux, the temperature gradient, and the coordinates of the integration points. The index  $n$  denotes the number of integration points used within the element, cf. flw3i8e.

**Theory:**

The temperature gradient and the heat flux are computed according to

$$\nabla T = \mathbf{B}^e \mathbf{a}^e$$

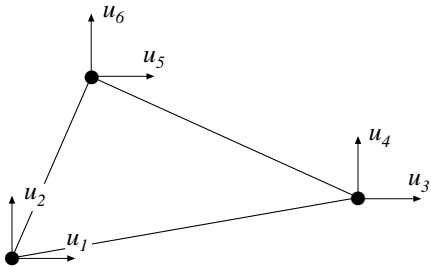
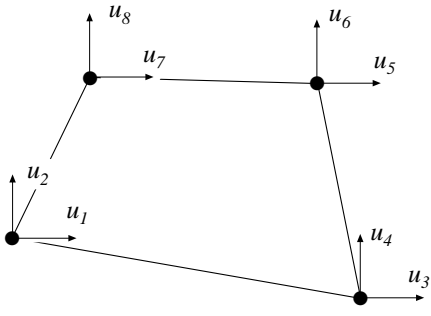
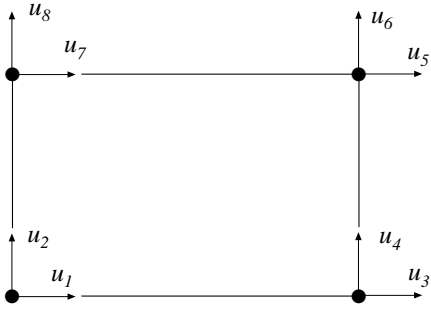
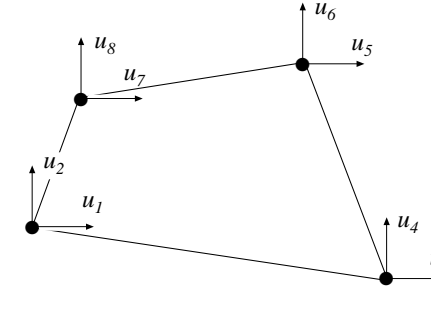
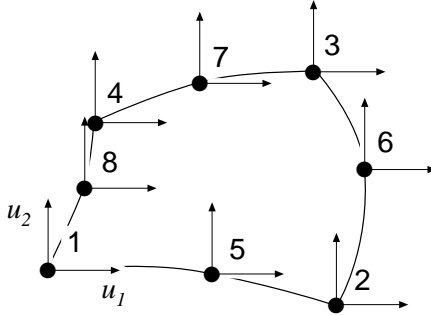
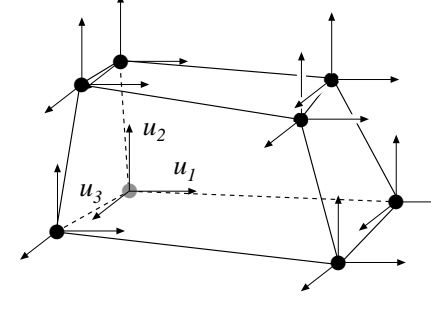
$$\mathbf{q} = -\mathbf{D}\nabla T$$

where the matrices **D**, **B<sup>e</sup>**, and **a<sup>e</sup>** are described in flw3i8e, and where the integration points are chosen as evaluation points.

## 5.5 Solid elements

Solid elements are available for two dimensional analysis in plane stress (panels) and plane strain, and for general three dimensional analysis. In the two dimensional case there are a triangular three node element, a quadrilateral four node element, two rectangular four node elements, and quadrilateral isoparametric four and eight node elements. For three dimensional analysis there is an eight node isoparametric element.

The elements are able to deal with both isotropic and anisotropic materials. The triangular element and the three isoparametric elements can also be used together with a nonlinear material model. The material properties are specified by supplying the constitutive matrix **D** as an input variable to the element functions. This matrix can be formed by the functions described in Section 4.

Solid elements	
 <p>plante</p>	 <p>planqe</p>
 <p>planre plantce</p>	 <p>plani4e</p>
 <p>plani8e</p>	 <p>soli8e</p>

<b>2D solid functions</b>	
plante	Compute element matrices for a triangular element
plants	Compute stresses and strains
plantf	Compute internal element forces
planqe	Compute element matrices for a quadrilateral element
planqs	Compute stresses and strains
planre	Compute element matrices for a rectangular Melosh element
planrs	Compute stresses and strains
plantce	Compute element matrices for a rectangular Turner-Clough element
plantcs	Compute stresses and strains
plani4e	Compute element matrices, 4 node isoparametric element
plani4s	Compute stresses and strains
plani4f	Compute internal element forces
plani8e	Compute element matrices, 8 node isoparametric element
plani8s	Compute stresses and strains
plani8f	Compute internal element forces

<b>3D solid functions</b>	
solli8e	Compute element matrices, 8 node isoparametric element
solli8s	Compute stresses and strains
solli8f	Compute internal element forces

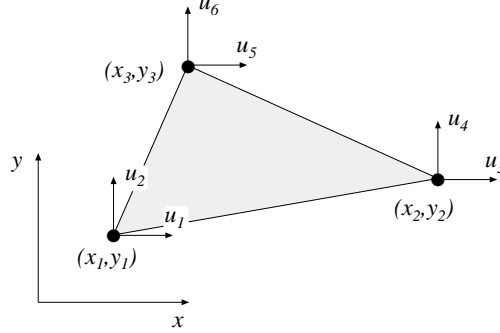
**plante**

**Two dimensional solid elements**

---

**Purpose:**

Compute element matrices for a triangular element in plane strain or plane stress.



**Syntax:**

```
Ke=plante(ex,ey,ep,D)
[Ke,fe]=plante(ex,ey,ep,D,eq)
```

**Description:**

**plante** provides an element stiffness matrix **Ke** and an element load vector **fe** for a triangular element in plane strain or plane stress.

The element nodal coordinates  $x_1, y_1, x_2$  etc. are supplied to the function by **ex** and **ey**. The type of analysis *p*type and the element thickness *t* are supplied by **ep**,

*p*type = 1 plane stress  
*p*type = 2 plane strain

and the material properties are supplied by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions from  $(3 \times 3)$  to  $(6 \times 6)$  may be given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \end{aligned} \quad \mathbf{ep} = [\textit{p}\textit{type} \ t]$$

$$\mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \quad \text{or} \quad \mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

## Two dimensional solid elements

plante

If uniformly distributed loads are applied to the element, the element load vector  $\mathbf{f}_e$  is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume,  $b_x$  and  $b_y$ , is then given.

### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{K}_e$  and  $\mathbf{f}_e$ , respectively, are computed according to

$$\begin{aligned} \mathbf{K}^e &= (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \mathbf{D} \bar{\mathbf{B}} t dA \mathbf{C}^{-1} \\ \mathbf{f}_l^e &= (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{N}}^T \mathbf{b} t dA \end{aligned}$$

with the constitutive matrix  $\mathbf{D}$  defined by  $\mathbf{D}$ , and the body force vector  $\mathbf{b}$  defined by  $\mathbf{eq}$ .

The evaluation of the integrals for the triangular element is based on a linear displacement approximation  $\mathbf{u}(x, y)$  and is expressed in terms of the nodal variables  $u_1, u_2, \dots, u_6$  as

$$\mathbf{u}(x, y) = \mathbf{N}^e \mathbf{a}^e = \bar{\mathbf{N}} \mathbf{C}^{-1} \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \bar{\mathbf{N}} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix}$$

The matrix  $\bar{\mathbf{B}}$  is obtained as

$$\bar{\mathbf{B}} = \tilde{\mathbf{V}} \bar{\mathbf{N}}_c \quad \text{where} \quad \tilde{\mathbf{V}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$



---

**plane**


---

**Two dimensional solid elements**


---

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane stress ( $ptype = 1$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix by static condensation using  $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$ . These stress components are connected with the rows 3, 5 and 6 in the  $\mathbf{D}$ -matrix respectively.

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane strain ( $ptype = 2$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix using  $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$ . This implies that a  $(3 \times 3)$   $\mathbf{D}$ -matrix is created by the rows and the columns 1, 2 and 4 from the original  $\mathbf{D}$ -matrix.

Evaluation of the integrals for the triangular element yields

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \bar{\mathbf{B}}^T \mathbf{D} \bar{\mathbf{B}} \mathbf{C}^{-1} t A$$

$$\mathbf{f}_l^e = \frac{A t}{3} [b_x \ b_y \ b_x \ b_y \ b_x \ b_y]^T$$

where the element area  $A$  is determined as

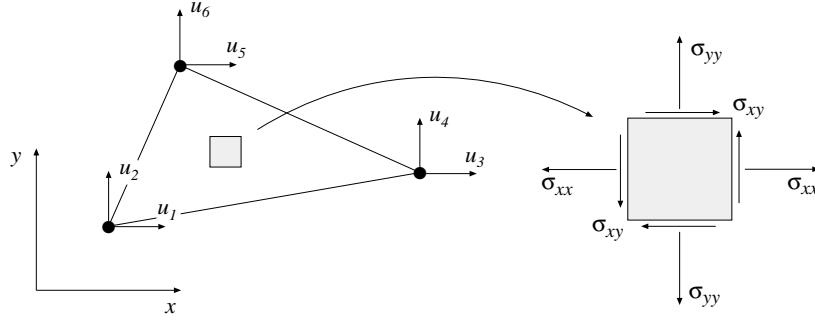
$$A = \frac{1}{2} \det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}$$

## Two dimensional solid elements

## plants

### Purpose:

Compute stresses and strains in a triangular element in plane strain or plane stress.



### Syntax:

`[es,et]=plants(ex,ey,ep,D,ed)`

### Description:

`plants` computes the stresses `es` and the strains `et` in a triangular element in plane strain or plane stress.

The input variables `ex`, `ey`, `ep` and `D` are defined in `plante`. The vector `ed` contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function `extract` as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_6]$$

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = [\sigma_{xx} \ \sigma_{yy} \ [\sigma_{zz}] \ \sigma_{xy} \ [\sigma_{xz}] \ [\sigma_{yz}]]$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = [\varepsilon_{xx} \ \varepsilon_{yy} \ [\varepsilon_{zz}] \ \gamma_{xy} \ [\gamma_{xz}] \ [\gamma_{yz}]]$$

contain the stress and strain components. The size of `es` and `et` follows the size of `D`. Note that for plane stress  $\varepsilon_{zz} \neq 0$ , and for plane strain  $\sigma_{zz} \neq 0$ .

### Theory:

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \bar{\mathbf{B}} \mathbf{C}^{-1} \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}$$

where the matrices  $\mathbf{D}$ ,  $\bar{\mathbf{B}}$ ,  $\mathbf{C}$  and  $\mathbf{a}^e$  are described in `plante`. Note that both the strains and the stresses are constant in the element.

**plantf**

**Two dimensional solid elements**

---

**Purpose:**

Compute internal element force vector in a triangular element in plane strain or plane stress.

**Syntax:**

`ef=plantf(ex,ey,ep,es)`

**Description:**

`plantf` computes the internal element forces `ef` in a triangular element in plane strain or plane stress.

The input variables `ex`, `ey` and `ep` are defined in `plante`, and the input variable `es` is defined in `plants`.

The output variable

$$\mathbf{ef} = \mathbf{f}_i^{eT} = [ f_{i1} \ f_{i2} \ \dots \ f_{i6} ]$$

contains the components of the internal force vector.

**Theory:**

The internal force vector is computed according to

$$\mathbf{f}_i^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \boldsymbol{\sigma} t \, dA$$

where the matrices  $\bar{\mathbf{B}}$  and  $\mathbf{C}$  are defined in `plante` and  $\boldsymbol{\sigma}$  is defined in `plants`.

Evaluation of the integral for the triangular element yields

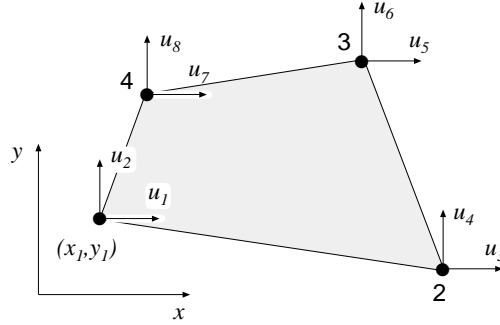
$$\mathbf{f}_i^e = (\mathbf{C}^{-1})^T \bar{\mathbf{B}}^T \boldsymbol{\sigma} t A$$

## Two dimensional solid elements

planqe

### Purpose:

Compute element matrices for a quadrilateral element in plane strain or plane stress.



### Syntax:

```
Ke=planqe(ex,ey,ep,D)
[Ke,fe]=planqe(ex,ey,ep,D,eq)
```

### Description:

planqe provides an element stiffness matrix **Ke** and an element load vector **fe** for a quadrilateral element in plane strain or plane stress.

The element nodal coordinates  $x_1, y_1, x_2$  etc. are supplied to the function by **ex** and **ey**. The type of analysis *ptype* and the element thickness  $t$  are supplied by **ep**,

```
ptype = 1  plane stress
ptype = 2  plane strain
```

and the material properties are supplied by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions from  $(3 \times 3)$  to  $(6 \times 6)$  may be given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

```
ex = [ x1 x2 x3 x4 ]
ey = [ y1 y2 y3 y4 ]      ep = [ ptype t ]
```

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \quad \text{or} \quad D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

**plane****Two dimensional solid elements**

---

If uniformly distributed loads are applied on the element, the element load vector **fe** is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume,  $b_x$  and  $b_y$ , is then given.

**Theory:**

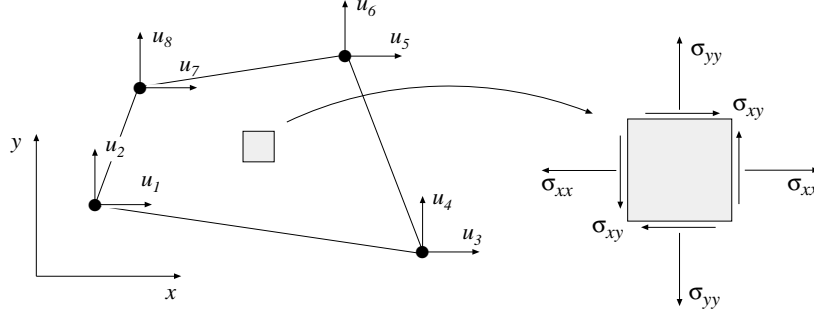
In computing the element matrices, two more degrees of freedom are introduced. The location of these two degrees of freedom is defined by the mean value of the coordinates at the corner points. Four sets of element matrices are calculated using **plane**. These matrices are then assembled and the two extra degrees of freedom are eliminated by static condensation.

## Two dimensional solid elements

## planqs

### Purpose:

Compute stresses and strains in a quadrilateral element in plane strain or plane stress.



### Syntax:

```
[es,et]=planqs(ex,ey,ep,D,ed)
[es,et]=planqs(ex,ey,ep,D,ed,eq)
```

### Description:

**planqs** computes the stresses **es** and the strains **et** in a quadrilateral element in plane strain or plane stress.

The input variables **ex**, **ey**, **ep**, **D** and **eq** are defined in **planqe**. The vector **ed** contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function **extract** as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_8]$$

If body forces are applied to the element the variable **eq** must be included.

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = [\sigma_{xx} \ \sigma_{yy} \ [\sigma_{zz}] \ \sigma_{xy} \ [\sigma_{xz}] \ [\sigma_{yz}]]$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = [\varepsilon_{xx} \ \varepsilon_{yy} \ [\varepsilon_{zz}] \ \gamma_{xy} \ [\gamma_{xz}] \ [\gamma_{yz}]]$$

contain the stress and strain components. The size of **es** and **et** follows the size of **D**. Note that for plane stress  $\varepsilon_{zz} \neq 0$ , and for plane strain  $\sigma_{zz} \neq 0$ .

### Theory:

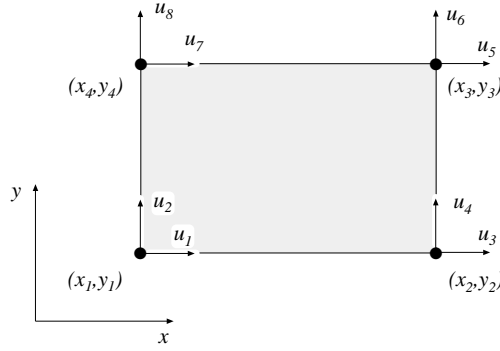
By assembling triangular elements as described in **planqe** a system of equations containing 10 degrees of freedom is obtained. From this system of equations the two unknown displacements at the center of the element are computed. Then according to the description in **plants** the strain and stress components in each of the four triangular elements are produced. Finally the quadrilateral element strains and stresses are computed as area weighted mean values from the values of the four triangular elements. If uniformly distributed loads are applied on the element, the element load vector **eq** is needed for the calculations.

## planre

## Two dimensional solid elements

### Purpose:

Compute element matrices for a rectangular (Melosh) element in plane strain or plane stress.



### Syntax:

```
Ke=planre(ex,ey,ep,D)
[Ke,fe]=planre(ex,ey,ep,D,eq)
```

### Description:

planre provides an element stiffness matrix **Ke** and an element load vector **fe** for a rectangular (Melosh) element in plane strain or plane stress. This element can only be used if the element edges are parallel to the coordinate axis.

The element nodal coordinates  $(x_1, y_1)$  and  $(x_3, y_3)$  are supplied to the function by **ex** and **ey**. The type of analysis *ptype* and the element thickness *t* are supplied by **ep**,

```
ptype = 1  plane stress
ptype = 2  plane strain
```

and the material properties are supplied by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions from  $(3 \times 3)$  to  $(6 \times 6)$  may be given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

```
ex = [ x1  x3 ]
ey = [ y1  y3 ]      ep = [ ptype  t ]
```

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \quad \text{or} \quad D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

## Two dimensional solid elements

planre

If uniformly distributed loads are applied on the element, the element load vector  $\mathbf{f}_e$  is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume,  $b_x$  and  $b_y$ , is then given.

### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{Ke}$  and  $\mathbf{fe}$ , respectively, are computed according to

$$\begin{aligned} \mathbf{K}^e &= \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e t dA \\ \mathbf{f}_l^e &= \int_A \mathbf{N}^{eT} \mathbf{b} t dA \end{aligned}$$

with the constitutive matrix  $\mathbf{D}$  defined by  $\mathbf{D}$ , and the body force vector  $\mathbf{b}$  defined by  $\mathbf{eq}$ .

The evaluation of the integrals for the rectangular element is based on a bilinear displacement approximation  $\mathbf{u}(x, y)$  and is expressed in terms of the nodal variables  $u_1, u_2, \dots, u_8$  as

$$\mathbf{u}(x, y) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e & 0 \\ 0 & N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_8 \end{bmatrix}$$

With a local coordinate system located at the center of the element, the element shape functions  $N_1^e - N_4^e$  are obtained as

$$\begin{aligned} N_1^e &= \frac{1}{4ab}(x - x_2)(y - y_4) \\ N_2^e &= -\frac{1}{4ab}(x - x_1)(y - y_3) \\ N_3^e &= \frac{1}{4ab}(x - x_4)(y - y_2) \\ N_4^e &= -\frac{1}{4ab}(x - x_3)(y - y_1) \end{aligned}$$

where

$$a = \frac{1}{2}(x_3 - x_1) \quad \text{and} \quad b = \frac{1}{2}(y_3 - y_1)$$



The matrix  $\mathbf{B}^e$  is obtained as

$$\mathbf{B}^e = \tilde{\mathbf{V}} \mathbf{N}^e \quad \text{where} \quad \tilde{\mathbf{V}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane stress ( $ptype = 1$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix by static condensation using  $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$ . These stress components are connected with the rows 3, 5 and 6 in the  $\mathbf{D}$ -matrix respectively.

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane strain ( $ptype = 2$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix using  $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$ . This implies that a  $(3 \times 3)$   $\mathbf{D}$ -matrix is created by the rows and the columns 1, 2 and 4 from the original  $\mathbf{D}$ -matrix.

Evaluation of the integrals for the rectangular element can be done either analytically or numerically by use of a  $2 \times 2$  point Gauss integration. The element load vector  $\mathbf{f}_l^e$  yields

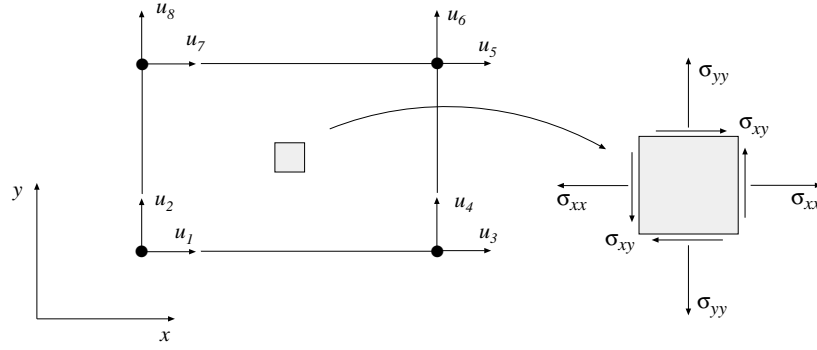
$$\mathbf{f}_l^e = abt \begin{bmatrix} b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \end{bmatrix}$$

## Two dimensional solid elements

## planrs

### Purpose:

Compute stresses and strains in a rectangular (Melosh) element in plane strain or plane stress.



### Syntax:

`[es,et]=planrs(ex,ey,ep,D,ed)`

### Description:

`planrs` computes the stresses `es` and the strains `et` in a rectangular (Melosh) element in plane strain or plane stress. The stress and strain components are computed at the center of the element.

The input variables `ex`, `ey`, `ep` and `D` are defined in `planre`. The vector `ed` contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function `extract` as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_8]$$

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = [\sigma_{xx} \ \sigma_{yy} \ [\sigma_{zz}] \ \sigma_{xy} \ [\sigma_{xz}] \ [\sigma_{yz}]]$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = [\varepsilon_{xx} \ \varepsilon_{yy} \ [\varepsilon_{zz}] \ \gamma_{xy} \ [\gamma_{xz}] \ [\gamma_{yz}]]$$

contain the stress and strain components. The size of `es` and `et` follows the size of `D`. Note that for plane stress  $\varepsilon_{zz} \neq 0$ , and for plane strain  $\sigma_{zz} \neq 0$ .

### Theory:

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}$$

where the matrices `D`, `Be`, and `ae` are described in `planre`, and where the evaluation point  $(x, y)$  is chosen to be at the center of the element.

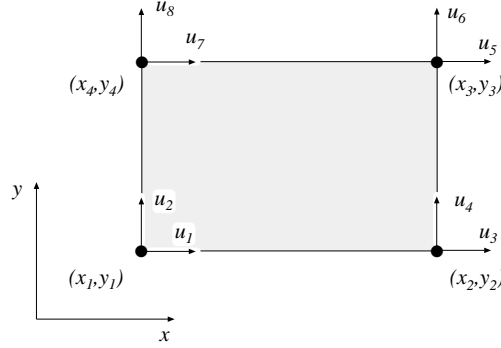
## plantce

## Two dimensional solid elements

---

### Purpose:

Compute element matrices for a rectangular (Turner-Clough) element in plane strain or plane stress.



### Syntax:

```
Ke=plantce(ex,ey,ep)
[Ke,fe]=plantce(ex,ey,ep,eq)
```

### Description:

plantce provides an element stiffness matrix **Ke** and an element load vector **fe** for a rectangular (Turner-Clough) element in plane strain or plane stress. This element can only be used if the material is isotropic and if the element edges are parallel to the coordinate axis.

The element nodal coordinates  $(x_1, y_1)$  and  $(x_3, y_3)$  are supplied to the function by **ex** and **ey**. The state of stress *ptype*, the element thickness *t* and the material properties *E* and  $\nu$  are supplied by **ep**. For plane stress *ptype* = 1 and for plane strain *ptype* = 2.

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_3 \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_3 \end{bmatrix} \end{aligned} \quad \mathbf{ep} = \begin{bmatrix} ptype & t & E & \nu \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector **fe** is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume,  $b_x$  and  $b_y$ , is then given.

## Two dimensional solid elements

plantce

### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in **Ke** and **fe**, respectively, are computed according to

$$\begin{aligned}\mathbf{K}^e &= \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e t dA \\ \mathbf{f}_l^e &= \int_A \mathbf{N}^{eT} \mathbf{b} t dA\end{aligned}$$

where the constitutive matrix  $\mathbf{D}$  is described in **hooke**, see Section 4, and the body force vector  $\mathbf{b}$  is defined by **eq**.

The evaluation of the integrals for the Turner-Clough element is based on a displacement field  $\mathbf{u}(x, y)$  built up of a bilinear displacement approximation superposed by bubble functions in order to create a linear stress field over the element. The displacement field is expressed in terms of the nodal variables  $u_1, u_2, \dots, u_8$  as

$$\mathbf{u}(x, y) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & N_5^e & N_2^e & N_5^e & N_3^e & N_5^e & N_4^e & N_5^e \\ N_6^e & N_1^e & N_6^e & N_2^e & N_6^e & N_3^e & N_6^e & N_4^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_8 \end{bmatrix}$$

With a local coordinate system located at the center of the element, the element shape functions  $N_1^e - N_6^e$  are obtained as

$$\begin{aligned}N_1^e &= \frac{1}{4ab}(a-x)(b-y) \\ N_2^e &= \frac{1}{4ab}(a+x)(b-y) \\ N_3^e &= \frac{1}{4ab}(a+x)(b+y) \\ N_4^e &= \frac{1}{4ab}(a-x)(b+y) \\ N_5^e &= \frac{1}{8ab}[(b^2 - y^2) + \nu(a^2 - x^2)] \\ N_6^e &= \frac{1}{8ab}[(a^2 - x^2) + \nu(b^2 - y^2)]\end{aligned}$$

where

$$a = \frac{1}{2}(x_3 - x_1) \quad \text{and} \quad b = \frac{1}{2}(y_3 - y_1)$$

The matrix  $\mathbf{B}^e$  is obtained as

$$\mathbf{B}^e = \tilde{\nabla} \mathbf{N}^e \quad \text{where} \quad \tilde{\nabla} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

Evaluation of the integrals for the Turner-Clough element can be done either analytically or numerically by use of a  $2 \times 2$  point Gauss integration. The element load vector  $\mathbf{f}_l^e$  yields

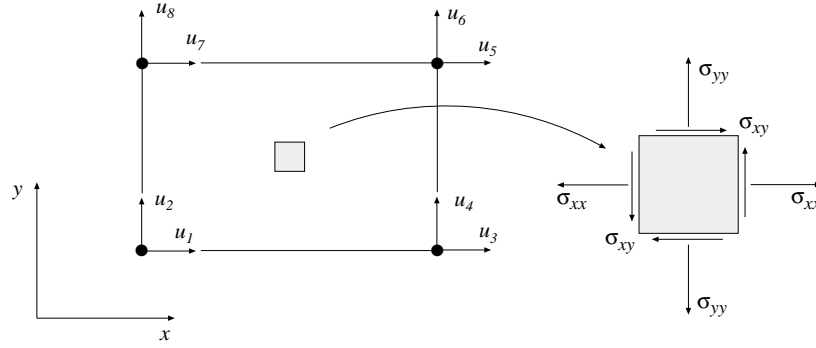
$$\mathbf{f}_l^e = abt \begin{bmatrix} b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \\ b_x \\ b_y \end{bmatrix}$$

## Two dimensional solid elements

plantcs

### Purpose:

Compute stresses and strains in a Turner-Clough element in plane strain or plane stress.



### Syntax:

[es,et]=plantcs(ex,ey,ep,ed)

### Description:

plantcs computes the stresses **es** and the strains **et** in a rectangular Turner-Clough element in plane strain or plane stress. The stress and strain components are computed at the center of the element.

The input variables **ex**, **ey**, and **ep** are defined in **plantce**. The vector **ed** contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function **extract** as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_8]$$

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = [\sigma_{xx} \ \sigma_{yy} \ [\sigma_{zz}] \ \sigma_{xy} \ [\sigma_{xz}] \ [\sigma_{yz}]]$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = [\varepsilon_{xx} \ \varepsilon_{yy} \ [\varepsilon_{zz}] \ \gamma_{xy} \ [\gamma_{xz}] \ [\gamma_{yz}]]$$

contain the stress and strain components. The size of **es** and **et** follows the size of **D**. Note that for plane stress  $\varepsilon_{zz} \neq 0$ , and for plane strain  $\sigma_{zz} \neq 0$ .

### Theory:

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}$$

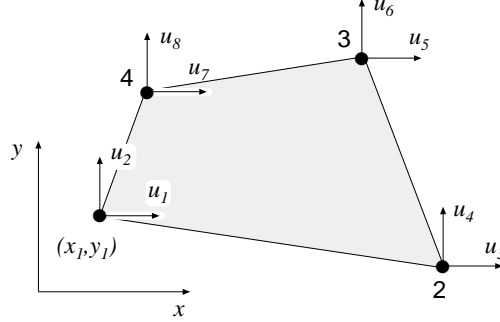
where the matrices **D**, **B<sup>e</sup>**, and  $\mathbf{a}^e$  are described in **plantce**, and where the evaluation point  $(x, y)$  is chosen to be at the center of the element.

## plani4e

## Two dimensional solid elements

### Purpose:

Compute element matrices for a 4 node isoparametric element in plane strain or plane stress.



### Syntax:

```
Ke=plani4e(ex,ey,ep,D)
[Ke,fe]=plani4e(ex,ey,ep,D,eq)
```

### Description:

plani4e provides an element stiffness matrix **Ke** and an element load vector **fe** for a 4 node isoparametric element in plane strain or plane stress.

The element nodal coordinates  $x_1, y_1, x_2$  etc. are supplied to the function by **ex** and **ey**. The type of analysis *ptype*, the element thickness *t*, and the number of Gauss points *n* are supplied by **ep**.

$ptype = 1$	plane stress	$(n \times n)$ integration points
$ptype = 2$	plane strain	$n = 1, 2, 3$

The material properties are supplied by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions from  $(3 \times 3)$  to  $(6 \times 6)$  maybe given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

$ex = [x_1 \ x_2 \ x_3 \ x_4]$	$ep = [ptype \ t \ n]$
$ey = [y_1 \ y_2 \ y_3 \ y_4]$	

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \text{ or } D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

## Two dimensional solid elements

plani4e

If different  $\mathbf{D}_i$  -matrices are used in the Gauss points these  $\mathbf{D}_i$  -matrices are stored in a global vector  $\mathbf{D}$ . For numbering of the Gauss points, see `eci` in `plani4s`.

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_{n^2} \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector  $\mathbf{f}_l$  is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume,  $b_x$  and  $b_y$ , is then given.

### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in `Ke` and `fe`, respectively, are computed according to

$$\begin{aligned} \mathbf{K}^e &= \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e t dA \\ \mathbf{f}_l^e &= \int_A \mathbf{N}^{eT} \mathbf{b} t dA \end{aligned}$$

with the constitutive matrix  $\mathbf{D}$  defined by `D`, and the body force vector  $\mathbf{b}$  defined by `eq`.

The evaluation of the integrals for the isoparametric 4 node element is based on a displacement approximation  $\mathbf{u}(\xi, \eta)$ , expressed in a local coordinates system in terms of the nodal variables  $u_1, u_2, \dots, u_8$  as

$$\mathbf{u}(\xi, \eta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e & 0 \\ 0 & N_1^e & 0 & N_2^e & 0 & N_3^e & 0 & N_4^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_8 \end{bmatrix}$$

The element shape functions are given by

$$\begin{aligned} N_1^e &= \frac{1}{4}(1 - \xi)(1 - \eta) & N_2^e &= \frac{1}{4}(1 + \xi)(1 - \eta) \\ N_3^e &= \frac{1}{4}(1 + \xi)(1 + \eta) & N_4^e &= \frac{1}{4}(1 - \xi)(1 + \eta) \end{aligned}$$



The matrix  $\mathbf{B}^e$  is obtained as

$$\mathbf{B}^e = \tilde{\mathbf{V}} \mathbf{N}^e \quad \text{where} \quad \tilde{\mathbf{V}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

and where

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \quad \mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane stress ( $p_{type} = 1$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix by static condensation using  $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$ . These stress components are connected with the rows 3, 5 and 6 in the  $\mathbf{D}$ -matrix respectively.

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane strain ( $p_{type} = 2$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix using  $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$ . This implies that a  $(3 \times 3)$   $\mathbf{D}$ -matrix is created by the rows and the columns 1, 2 and 4 from the original  $\mathbf{D}$ -matrix.

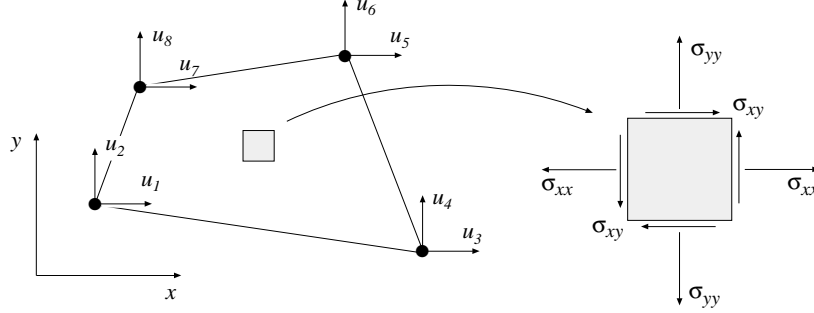
Evaluation of the integrals is done by Gauss integration.

## Two dimensional solid elements

## plani4s

## Purpose:

Compute stresses and strains in a 4 node isoparametric element in plane strain or plane stress.



## Syntax:

`[es,et,eci]=planis4s(ex,ey,ep,D,ed)`

## Description:

`planis4s` computes stresses `es` and the strains `et` in a 4 node isoparametric element in plane strain or plane stress.

The input variables `ex`, `ey`, `ep` and the matrix `D` are defined in `planis4e`. The vector `ed` contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function `extract` as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_8]$$

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_{xx}^1 & \sigma_{yy}^1 & [\sigma_{zz}^1] & \sigma_{xy}^1 & [\sigma_{xz}^1] & [\sigma_{yz}^1] \\ \sigma_{xx}^2 & \sigma_{yy}^2 & [\sigma_{zz}^2] & \sigma_{xy}^2 & [\sigma_{xz}^2] & [\sigma_{yz}^2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_{xx}^{n^2} & \sigma_{yy}^{n^2} & [\sigma_{zz}^{n^2}] & \sigma_{xy}^{n^2} & [\sigma_{xz}^{n^2}] & [\sigma_{yz}^{n^2}] \end{bmatrix}$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon_{xx}^1 & \varepsilon_{yy}^1 & [\varepsilon_{zz}^1] & \gamma_{xy}^1 & [\gamma_{xz}^1] & [\gamma_{yz}^1] \\ \varepsilon_{xx}^2 & \varepsilon_{yy}^2 & [\varepsilon_{zz}^2] & \gamma_{xy}^2 & [\gamma_{xz}^2] & [\gamma_{yz}^2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{xx}^{n^2} & \varepsilon_{yy}^{n^2} & [\varepsilon_{zz}^{n^2}] & \gamma_{xy}^{n^2} & [\gamma_{xz}^{n^2}] & [\gamma_{yz}^{n^2}] \end{bmatrix} \quad \mathbf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the stress and strain components, and the coordinates of the integration points. The index  $n$  denotes the number of integration points used within the element, cf. `planis4e`. The number of columns in `es` and `et` follows the size of `D`. Note that for plane stress  $\varepsilon_{zz} \neq 0$ , and for plane strain  $\sigma_{zz} \neq 0$ .

**plani4s**

**Two dimensional solid elements**

---

**Theory:**

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}$$

where the matrices  $\mathbf{D}$ ,  $\mathbf{B}^e$ , and  $\mathbf{a}^e$  are described in **plani4e**, and where the integration points are chosen as evaluation points.

**Two dimensional solid elements****plani4f**

---

**Purpose:**

Compute internal element force vector in a 4 node isoparametric element in plane strain or plane stress.

**Syntax:**

`ef=plani4f(ex,ey,ep,es)`

**Description:**

`plani4f` computes the internal element forces `ef` in a 4 node isoparametric element in plane strain or plane stress.

The input variables `ex`, `ey` and `ep` are defined in `plani4e`, and the input variable `es` is defined in `plani4s`.

The output variable

$$\mathbf{ef} = \mathbf{f}_i^{eT} = [ f_{i1} \ f_{i2} \ \dots \ f_{i8} ]$$

contains the components of the internal force vector.

**Theory:**

The internal force vector is computed according to

$$\mathbf{f}_i^e = \int_A \mathbf{B}^{eT} \boldsymbol{\sigma} t \, dA$$

where the matrices  $\mathbf{B}^e$  and  $\boldsymbol{\sigma}$  are defined in `plani4e` and `plani4s`, respectively.

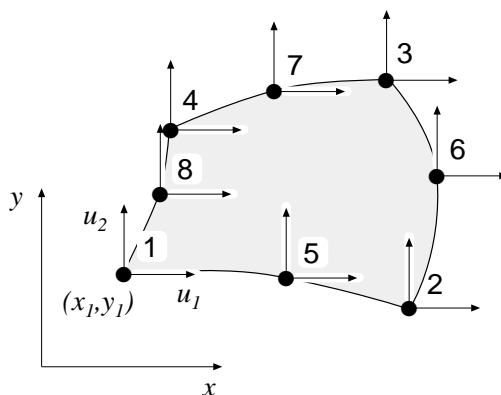
Evaluation of the integral is done by Gauss integration.

## plani8e

## Two dimensional solid elements

### Purpose:

Compute element matrices for an 8 node isoparametric element in plane strain or plane stress.



### Syntax:

```
Ke=plani8e(ex,ey,ep,D)
[Ke,fe]=plani8e(ex,ey,ep,D,eq)
```

### Description:

plani8e provides an element stiffness matrix **Ke** and an element load vector **fe** for an 8 node isoparametric element in plane strain or plane stress.

The element nodal coordinates  $x_1, y_1, x_2$  etc. are supplied to the function by **ex** and **ey**. The type of analysis *ptype*, the element thickness *t*, and the number of Gauss points *n* are supplied by **ep**.

$ptype = 1$	plane stress	$(n \times n)$ integration points
$ptype = 2$	plane strain	$n = 1, 2, 3$

The material properties are supplied by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions from  $(3 \times 3)$  to  $(6 \times 6)$  may be given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

$\mathbf{ex} = [x_1 \ x_2 \ \dots \ x_8]$	$\mathbf{ep} = [ptype \ t \ n]$
$\mathbf{ey} = [y_1 \ y_2 \ \dots \ y_8]$	

$$\mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix} \quad \text{or} \quad \mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & [D_{15}] & [D_{16}] \\ D_{21} & D_{22} & D_{23} & D_{24} & [D_{25}] & [D_{26}] \\ D_{31} & D_{32} & D_{33} & D_{34} & [D_{35}] & [D_{36}] \\ D_{41} & D_{42} & D_{43} & D_{44} & [D_{45}] & [D_{46}] \\ [D_{51}] & [D_{52}] & [D_{53}] & [D_{54}] & [D_{55}] & [D_{56}] \\ [D_{61}] & [D_{62}] & [D_{63}] & [D_{64}] & [D_{65}] & [D_{66}] \end{bmatrix}$$

## Two dimensional solid elements

plani8e

If different  $\mathbf{D}_i$  -matrices are used in the Gauss points these  $\mathbf{D}_i$  -matrices are stored in a global vector  $\mathbf{D}$ . For numbering of the Gauss points, see `eci` in `plani8s`.

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_{n^2} \end{bmatrix}$$

If uniformly distributed loads are applied to the element, the element load vector  $\mathbf{f}_e$  is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

containing loads per unit volume,  $b_x$  and  $b_y$ , is then given.

### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in `Ke` and `fe`, respectively, are computed according to

$$\mathbf{K}^e = \int_A \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e t dA$$

$$\mathbf{f}_l^e = \int_A \mathbf{N}^{eT} \mathbf{b} t dA$$

with the constitutive matrix  $\mathbf{D}$  defined by `D`, and the body force vector  $\mathbf{b}$  defined by `eq`.

The evaluation of the integrals for the isoparametric 8 node element is based on a displacement approximation  $\mathbf{u}(\xi, \eta)$ , expressed in a local coordinates system in terms of the nodal variables  $u_1, u_2, \dots, u_{16}$  as

$$\mathbf{u}(\xi, \eta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & \dots & N_8^e & 0 \\ 0 & N_1^e & 0 & N_2^e & \dots & 0 & N_8^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{16} \end{bmatrix}$$

The element shape functions are given by

$$\begin{aligned} N_1^e &= -\frac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta) & N_5^e &= \frac{1}{2}(1-\xi^2)(1-\eta) \\ N_2^e &= -\frac{1}{4}(1+\xi)(1-\eta)(1-\xi+\eta) & N_6^e &= \frac{1}{2}(1+\xi)(1-\eta^2) \\ N_3^e &= -\frac{1}{4}(1+\xi)(1+\eta)(1-\xi-\eta) & N_7^e &= \frac{1}{2}(1-\xi^2)(1+\eta) \\ N_4^e &= -\frac{1}{4}(1-\xi)(1+\eta)(1+\xi-\eta) & N_8^e &= \frac{1}{2}(1-\xi)(1-\eta^2) \end{aligned}$$

The matrix  $\mathbf{B}^e$  is obtained as

$$\mathbf{B}^e = \tilde{\mathbf{V}} \mathbf{N}^e \quad \text{where} \quad \tilde{\mathbf{V}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

and where

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \quad \mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane stress ( $p\text{type} = 1$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix by static condensation using  $\sigma_{zz} = \sigma_{xz} = \sigma_{yz} = 0$ . These stress components are connected with the rows 3, 5 and 6 in the  $\mathbf{D}$ -matrix respectively.

If a larger  $\mathbf{D}$ -matrix than  $(3 \times 3)$  is used for plane strain ( $p\text{type} = 2$ ), the  $\mathbf{D}$ -matrix is reduced to a  $(3 \times 3)$  matrix using  $\varepsilon_{zz} = \gamma_{xz} = \gamma_{yz} = 0$ . This implies that a  $(3 \times 3)$   $\mathbf{D}$ -matrix is created by the rows and the columns 1, 2 and 4 from the original  $\mathbf{D}$ -matrix.

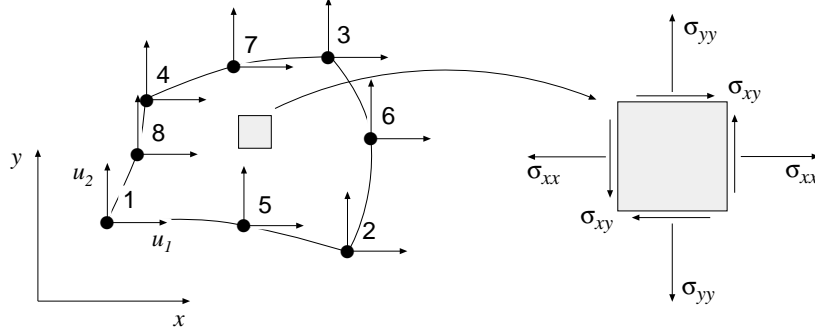
Evaluation of the integrals is done by Gauss integration.

## Two dimensional solid elements

plani8s

### Purpose:

Compute stresses and strains in an 8 node isoparametric element in plane strain or plane stress.



### Syntax:

`[es,et,eci]=plani8s(ex,ey,ep,D,ed)`

### Description:

plani8s computes stresses **es** and the strains **et** in an 8 node isoparametric element in plane strain or plane stress.

The input variables **ex**, **ey**, **ep** and the matrix **D** are defined in **plani8e**. The vector **ed** contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function **extract** as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_{16}]$$

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_{xx}^1 & \sigma_{yy}^1 & [\sigma_{zz}^1] & \sigma_{xy}^1 & [\sigma_{xz}^1] & [\sigma_{yz}^1] \\ \sigma_{xx}^2 & \sigma_{yy}^2 & [\sigma_{zz}^2] & \sigma_{xy}^2 & [\sigma_{xz}^2] & [\sigma_{yz}^2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_{xx}^{n^2} & \sigma_{yy}^{n^2} & [\sigma_{zz}^{n^2}] & \sigma_{xy}^{n^2} & [\sigma_{xz}^{n^2}] & [\sigma_{yz}^{n^2}] \end{bmatrix}$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon_{xx}^1 & \varepsilon_{yy}^1 & [\varepsilon_{zz}^1] & \gamma_{xy}^1 & [\gamma_{xz}^1] & [\gamma_{yz}^1] \\ \varepsilon_{xx}^2 & \varepsilon_{yy}^2 & [\varepsilon_{zz}^2] & \gamma_{xy}^2 & [\gamma_{xz}^2] & [\gamma_{yz}^2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{xx}^{n^2} & \varepsilon_{yy}^{n^2} & [\varepsilon_{zz}^{n^2}] & \gamma_{xy}^{n^2} & [\gamma_{xz}^{n^2}] & [\gamma_{yz}^{n^2}] \end{bmatrix} \quad \mathbf{eci} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n^2} & y_{n^2} \end{bmatrix}$$

contain the stress and strain components, and the coordinates of the integration points. The index  $n$  denotes the number of integration points used within the element, cf. **plani8e**. The number of columns in **es** and **et** follows the size of **D**. Note that for plane stress  $\varepsilon_{zz} \neq 0$ , and for plane strain  $\sigma_{zz} \neq 0$ .



**plani8s**

**Two dimensional solid elements**

---

**Theory:**

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}$$

where the matrices  $\mathbf{D}$ ,  $\mathbf{B}^e$ , and  $\mathbf{a}^e$  are described in **plani8e**, and where the integration points are chosen as evaluation points.

**Two dimensional solid elements****plani8f**

---

**Purpose:**

Compute internal element force vector in an 8 node isoparametric element in plane strain or plane stress.

**Syntax:**

`ef=plani8f(ex,ey,ep,es)`

**Description:**

`plani8f` computes the internal element forces `ef` in an 8 node isoparametric element in plane strain or plane stress.

The input variables `ex`, `ey` and `ep` are defined in `plani8e`, and the input variable `es` is defined in `plani8s`.

The output variable

$$\mathbf{ef} = \mathbf{f}_i^{eT} = [ f_{i1} \ f_{i2} \ \dots \ f_{i16} ]$$

contains the components of the internal force vector.

**Theory:**

The internal force vector is computed according to

$$\mathbf{f}_i^e = \int_A \mathbf{B}^{eT} \boldsymbol{\sigma} t \, dA$$

where the matrices  $\mathbf{B}^e$  and  $\boldsymbol{\sigma}$  are defined in `plani8e` and `plani8s`, respectively.

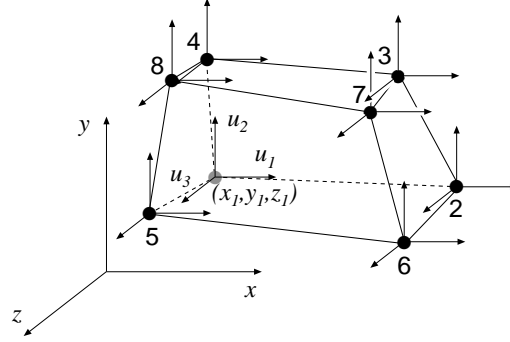
Evaluation of the integral is done by Gauss integration.

## sol8e

## Three dimensional solid elements

### Purpose:

Compute element matrices for an 8 node isoparametric solid element.



### Syntax:

```
Ke=sol8e(ex,ey,ez,ep,D)
[Ke,fe]=sol8e(ex,ey,ez,ep,D,eq)
```

### Description:

sol8e provides an element stiffness matrix **Ke** and an element load vector **fe** for an 8 node isoparametric solid element.

The element nodal coordinates  $x_1, y_1, z_1, x_2$  etc. are supplied to the function by **ex**, **ey** and **ez**, and the number of Gauss points  $n$  are supplied by **ep**.

$(n \times n)$  integration points,  $n = 1, 2, 3$

The material properties are supplied by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions  $(6 \times 6)$  may be given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

$$\begin{aligned} \mathbf{ex} &= [x_1 \ x_2 \ \dots \ x_8] \\ \mathbf{ey} &= [y_1 \ y_2 \ \dots \ y_8] \\ \mathbf{ez} &= [z_1 \ z_2 \ \dots \ z_8] \end{aligned} \quad \mathbf{ep} = [n] \quad \mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & \dots & D_{16} \\ D_{21} & D_{22} & \dots & D_{26} \\ \vdots & \vdots & \ddots & \vdots \\ D_{61} & D_{62} & \dots & D_{66} \end{bmatrix}$$

If different  $\mathbf{D}_i$  -matrices are used in the Gauss points these  $\mathbf{D}_i$  -matrices are stored in a global vector **D**. For numbering of the Gauss points, see **eci** in **sol8s**.

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_{n^3} \end{bmatrix}$$

### Three dimensional solid elements

sol8e

If uniformly distributed loads are applied to the element, the element load vector  $\mathbf{f}_e$  is computed. The input variable

$$\mathbf{eq} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

containing loads per unit volume,  $b_x$ ,  $b_y$ , and  $b_z$ , is then given.

#### Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{Ke}$  and  $\mathbf{fe}$ , respectively, are computed according to

$$\begin{aligned} \mathbf{K}^e &= \int_V \mathbf{B}^{eT} \mathbf{D} \mathbf{B}^e dV \\ \mathbf{f}_l^e &= \int_V \mathbf{N}^{eT} \mathbf{b} dV \end{aligned}$$

with the constitutive matrix  $\mathbf{D}$  defined by  $\mathbf{D}$ , and the body force vector  $\mathbf{b}$  defined by  $\mathbf{eq}$ .

The evaluation of the integrals for the isoparametric 8 node solid element is based on a displacement approximation  $\mathbf{u}(\xi, \eta, \zeta)$ , expressed in a local coordinates system in terms of the nodal variables  $u_1, u_2, \dots, u_{24}$  as

$$\mathbf{u}(\xi, \eta, \zeta) = \mathbf{N}^e \mathbf{a}^e$$

where

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \quad \mathbf{N}^e = \begin{bmatrix} N_1^e & 0 & 0 & N_2^e & 0 & 0 & \dots & N_8^e & 0 & 0 \\ 0 & N_1^e & 0 & 0 & N_2^e & 0 & \dots & 0 & N_8^e & 0 \\ 0 & 0 & N_1^e & 0 & 0 & N_2^e & \dots & 0 & 0 & N_8^e \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{24} \end{bmatrix}$$

The element shape functions are given by

$$\begin{aligned} N_1^e &= \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta) & N_5^e &= \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta) \\ N_2^e &= \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta) & N_6^e &= \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta) \\ N_3^e &= \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta) & N_7^e &= \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta) \\ N_4^e &= \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta) & N_8^e &= \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta) \end{aligned}$$

The  $\mathbf{B}^e$ -matrix is obtained as

$$\mathbf{B}^e = \tilde{\nabla} \mathbf{N}^e$$

where

$$\tilde{\nabla} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}$$

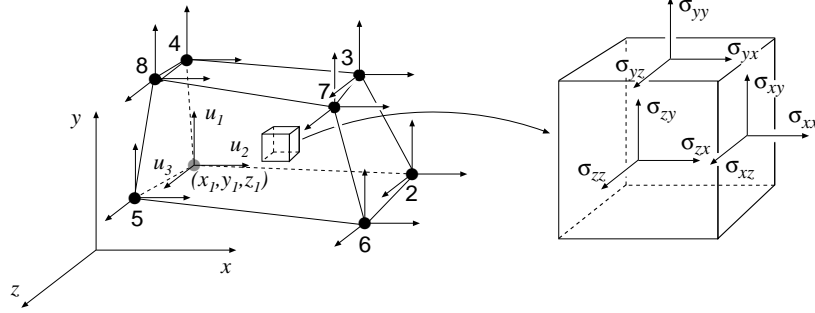
Evaluation of the integrals is done by Gauss integration.

## Three dimensional solid elements

soli8s

## Purpose:

Compute stresses and strains in an 8 node isoparametric solid element.



## Syntax:

`[es,et,eci]=soli8s(ex,ey,ez,ep,D,ed)`

## Description:

`soli8s` computes stresses `es` and the strains `et` in an 8 node isoparametric solid element.

The input variables `ex`, `ey`, `ez`, `ep` and the matrix `D` are defined in `soli8e`. The vector `ed` contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function `extract` as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_{24}]$$

The output variables

$$\mathbf{es} = \boldsymbol{\sigma}^T = \begin{bmatrix} \sigma_{xx}^1 & \sigma_{yy}^1 & \sigma_{zz}^1 & \sigma_{xy}^1 & \sigma_{xz}^1 & \sigma_{yz}^1 \\ \sigma_{xx}^2 & \sigma_{yy}^2 & \sigma_{zz}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 & \sigma_{yz}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sigma_{xx}^{n^3} & \sigma_{yy}^{n^3} & \sigma_{zz}^{n^3} & \sigma_{xy}^{n^3} & \sigma_{xz}^{n^3} & \sigma_{yz}^{n^3} \end{bmatrix}$$

$$\mathbf{et} = \boldsymbol{\varepsilon}^T = \begin{bmatrix} \varepsilon_{xx}^1 & \varepsilon_{yy}^1 & \varepsilon_{zz}^1 & \gamma_{xy}^1 & \gamma_{xz}^1 & \gamma_{yz}^1 \\ \varepsilon_{xx}^2 & \varepsilon_{yy}^2 & \varepsilon_{zz}^2 & \gamma_{xy}^2 & \gamma_{xz}^2 & \gamma_{yz}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{xx}^{n^3} & \varepsilon_{yy}^{n^3} & \varepsilon_{zz}^{n^3} & \gamma_{xy}^{n^3} & \gamma_{xz}^{n^3} & \gamma_{yz}^{n^3} \end{bmatrix} \quad \mathbf{eci} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_{n^3} & y_{n^3} & z_{n^3} \end{bmatrix}$$

contain the stress and strain components, and the coordinates of the integration points. The index  $n$  denotes the number of integration points used within the element, cf. `soli8e`.

**solis8s**

**Three dimensional solid elements**

---

**Theory:**

The strains and stresses are computed according to

$$\boldsymbol{\varepsilon} = \mathbf{B}^e \mathbf{a}^e$$

$$\boldsymbol{\sigma} = \mathbf{D} \boldsymbol{\varepsilon}$$

where the matrices  $\mathbf{D}$ ,  $\mathbf{B}^e$ , and  $\mathbf{a}^e$  are described in **solis8e**, and where the integration points are chosen as evaluation points.

**Three dimensional solid elements****solli8f**

---

**Purpose:**

Compute internal element force vector in an 8 node isoparametric solid element.

**Syntax:**

`ef=solli8f(ex,ey,ez,ep,es)`

**Description:**

`solli8f` computes the internal element forces `ef` in an 8 node isoparametric solid element.

The input variables `ex`, `ey`, `ez` and `ep` are defined in `solli8e`, and the input variable `es` is defined in `solli8s`.

The output variable

$$\mathbf{ef} = \mathbf{f}_i^{eT} = [ f_{i1} \ f_{i2} \ \dots \ f_{i24} ]$$

contains the components of the internal force vector.

**Theory:**

The internal force vector is computed according to

$$\mathbf{f}_i^e = \int_V \mathbf{B}^{eT} \boldsymbol{\sigma} \, dV$$

where the matrices  $\mathbf{B}$  and  $\boldsymbol{\sigma}$  are defined in `solli8e` and `solli8s`, respectively.

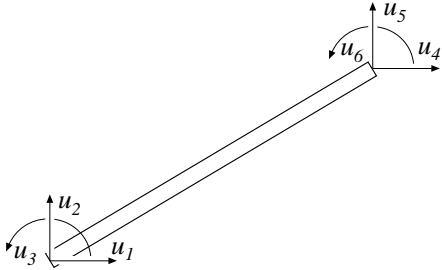
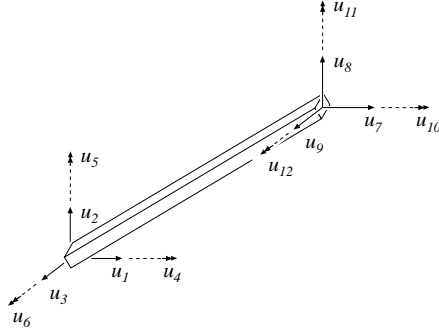
Evaluation of the integral is done by Gauss integration.





## 5.6 Beam elements

Beam elements are available for two, and three dimensional linear static analysis. Two dimensional beam elements for nonlinear geometric and dynamic analysis are also available.

Beam elements	
 <p>beam2e beam2t beam2w beam2g beam2d</p>	 <p>beam3e</p>

2D beam elements	
beam2e	Compute element matrices
beam2s	Compute section forces
beam2t	Compute element matrices for Timoshenko beam element
beam2ts	Compute section forces for Timoshenko beam element
beam2w	Compute element matrices for beam element on elastic foundation
beam2ws	Compute section forces for beam element on elastic foundation
beam2g	Compute element matrices with respect to geometric nonlinearity
beam2gs	Compute section forces for geometric nonlinear beam element
beam2d	Compute element matrices, dynamic analysis
beam2ds	Compute section forces, dynamic analysis

3D beam elements	
beam3e	Compute element matrices
beam3s	Compute section forces

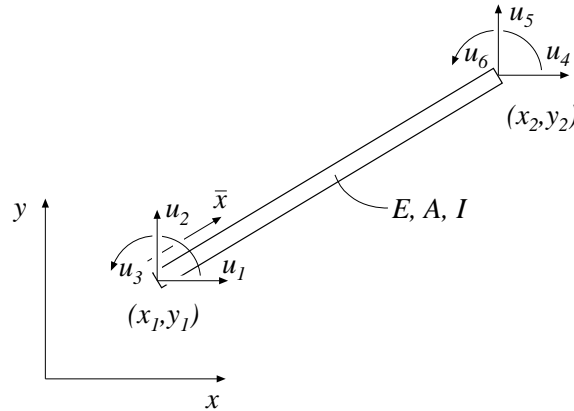
## beam2e

## Two dimensional beam element

---

### Purpose:

Compute element stiffness matrix for a two dimensional beam element.



### Syntax:

```
Ke=beam2e(ex,ey,ep)
[Ke,fe]=beam2e(ex,ey,ep,eq)
```

### Description:

beam2e provides the global element stiffness matrix **Ke** for a two dimensional beam element.

The input variables

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \end{aligned} \quad \mathbf{ep} = \begin{bmatrix} E & A & I \end{bmatrix}$$

supply the element nodal coordinates  $x_1$ ,  $y_1$ ,  $x_2$ , and  $y_2$ , the modulus of elasticity  $E$ , the cross section area  $A$ , and the moment of inertia  $I$ .

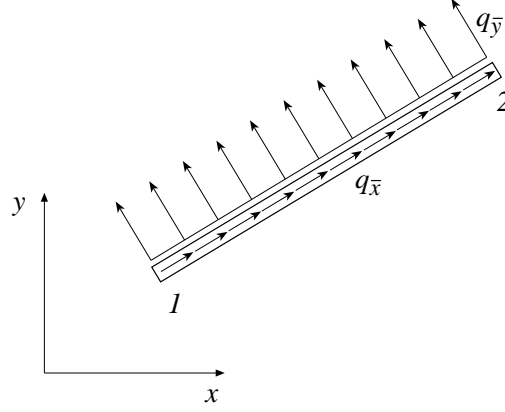
The element load vector **fe** can also be computed if uniformly distributed loads are applied to the element. The optional input variable

$$\mathbf{eq} = \begin{bmatrix} q_{\bar{x}} & q_{\bar{y}} \end{bmatrix}$$

then contains the distributed loads per unit length,  $q_{\bar{x}}$  and  $q_{\bar{y}}$ .

Two dimensional beam element

beam2e



Theory:

The element stiffness matrix  $\mathbf{K}^e$ , stored in  $\mathbf{Ke}$ , is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & 0 & 0 & 0 & 0 \\ n_{x\bar{y}} & n_{y\bar{y}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & 0 \\ 0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix  $\mathbf{G}$  contains the direction cosines

$$n_{x\bar{x}} = n_{y\bar{y}} = \frac{x_2 - x_1}{L} \quad n_{y\bar{x}} = -n_{x\bar{y}} = \frac{y_2 - y_1}{L}$$

where the length

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

**beam2e**

**Two dimensional beam element**

---

The element load vector  $\mathbf{f}_l^e$ , stored in `fe`, is computed according to

$$\mathbf{f}_l^e = \mathbf{G}^T \bar{\mathbf{f}}_l^e$$

where

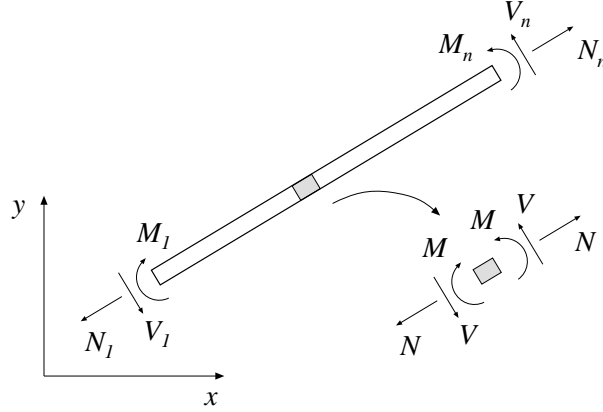
$$\bar{\mathbf{f}}_l^e = \begin{bmatrix} \frac{q_{\bar{x}} L}{2} \\ \frac{q_{\bar{y}} L}{2} \\ \frac{q_{\bar{y}} L^2}{12} \\ \frac{q_{\bar{x}} L}{2} \\ \frac{q_{\bar{y}} L}{2} \\ -\frac{q_{\bar{y}} L^2}{12} \end{bmatrix}$$

## Two dimensional beam element

beam2s

### Purpose:

Compute section forces in a two dimensional beam element.



### Syntax:

```
es=beam2s(ex,ey,ep,ed)
es=beam2s(ex,ey,ep,ed,eq)
[es,edi,eci]=beam2s(ex,ey,ep,ed,eq,n)
```

### Description:

beam2s computes the section forces and displacements in local directions along the beam element beam2e.

The input variables **ex**, **ey**, **ep** and **eq** are defined in **beam2e**, and the element displacements, stored in **ed**, are obtained by the function **extract**. If distributed loads are applied to the element, the variable **eq** must be included. The number of evaluation points for section forces and displacements are determined by **n**. If **n** is omitted, only the ends of the beam are evaluated.

The output variables

$$es = [ \mathbf{N} \quad \mathbf{V} \quad \mathbf{M} ] \quad edi = [ \bar{\mathbf{u}} \quad \bar{\mathbf{v}} ] \quad eci = [ \bar{\mathbf{x}} ]$$

consist of column matrices that contain the section forces, the displacements, and the evaluation points on the local x-axis. The explicit matrix expressions are

$$es = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \\ \vdots & \vdots & \vdots \\ N_n & V_n & M_n \end{bmatrix} \quad edi = \begin{bmatrix} \bar{u}_1 & \bar{v}_1 \\ \bar{u}_2 & \bar{v}_2 \\ \vdots & \vdots \\ \bar{u}_n & \bar{v}_n \end{bmatrix} \quad eci = \begin{bmatrix} 0 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_{n-1} \\ L \end{bmatrix}$$

where  $L$  is the length of the beam element.

**beam2s**

**Two dimensional beam element**

---

**Theory:**

The evaluation of the section forces is based on the solutions of the basic equations

$$EA \frac{d^2 \bar{u}}{d\bar{x}^2} + q_{\bar{x}} = 0 \quad EI \frac{d^4 \bar{v}}{d\bar{x}^4} - q_{\bar{y}} = 0$$

From these equations, the displacements along the beam element are obtained as the sum of the homogeneous and the particular solutions

$$\mathbf{u} = \begin{bmatrix} \bar{u}(\bar{x}) \\ \bar{v}(\bar{x}) \end{bmatrix} = \mathbf{u}_h + \mathbf{u}_p$$

where

$$\mathbf{u}_h = \bar{\mathbf{N}} \mathbf{C}^{-1} \mathbf{G} \mathbf{a}^e \quad \mathbf{u}_p = \begin{bmatrix} \bar{u}_p(\bar{x}) \\ \bar{v}_p(\bar{x}) \end{bmatrix} = \begin{bmatrix} \frac{q_{\bar{x}} L \bar{x}}{2EA} (1 - \frac{\bar{x}}{L}) \\ \frac{q_{\bar{y}} L^2 \bar{x}^2}{24EI} (1 - \frac{\bar{x}}{L})^2 \end{bmatrix}$$

and

$$\bar{\mathbf{N}} = \begin{bmatrix} 1 & \bar{x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & L & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & L^2 & L^3 \\ 0 & 0 & 0 & 1 & 2L & 3L^2 \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_6 \end{bmatrix}$$

The transformation matrix  $\mathbf{G}^e$  and nodal displacements  $\mathbf{a}^e$  are described in **beam2e**. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.

Finally the section forces are obtained from

$$N = EA \frac{d\bar{u}}{d\bar{x}} \quad V = -EI \frac{d^3 \bar{v}}{d\bar{x}^3} \quad M = EI \frac{d^2 \bar{v}}{d\bar{x}^2}$$

**Examples:**

Section forces or element displacements can easily be plotted. The bending moment  $M$  along the beam is plotted by

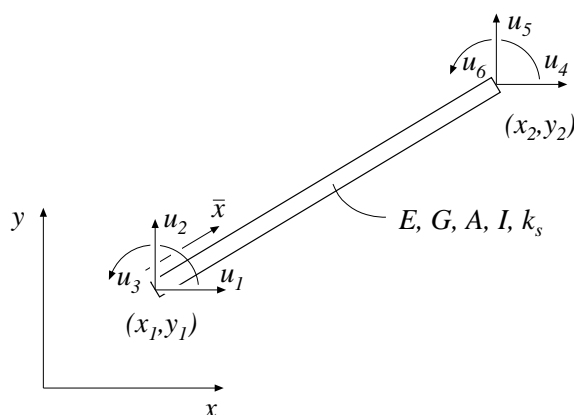
```
>> plot(eci,es(:,3))
```

## Two dimensional Timoshenko beam element

beam2t

### Purpose:

Compute element stiffness matrix for a two dimensional Timoshenko beam element.



### Syntax:

```
Ke=beam2t(ex,ey,ep)
[Ke,fe]=beam2t(ex,ey,ep,eq)
```

### Description:

beam2t provides the global element stiffness matrix **Ke** for a two dimensional Timoshenko beam element.

The input variables

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} & \mathbf{ep} &= \begin{bmatrix} E & G & A & I & k_s \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \end{aligned}$$

supply the element nodal coordinates  $x_1$ ,  $y_1$ ,  $x_2$ , and  $y_2$ , the modulus of elasticity  $E$ , the shear modulus  $G$ , the cross section area  $A$ , the moment of inertia  $I$  and the shear correction factor  $k_s$ .

The element load vector **fe** can also be computed if uniformly distributed loads are applied to the element. The optional input variable

$$\mathbf{eq} = \begin{bmatrix} q_{\bar{x}} & q_{\bar{y}} \end{bmatrix}$$

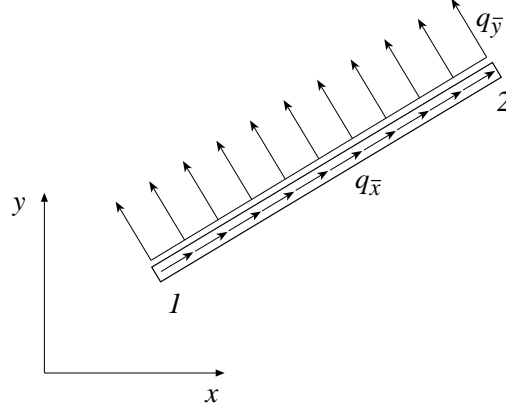
then contains the distributed loads per unit length,  $q_{\bar{x}}$  and  $q_{\bar{y}}$ .



beam2t

Two dimensional Timoshenko beam element

---



**Theory:**

The element stiffness matrix  $\mathbf{K}^e$ , stored in `Ke`, is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where  $\mathbf{G}$  is described in `beam2e`, and  $\bar{\mathbf{K}}^e$  is given by

$$\bar{\mathbf{K}}^e = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3(1+\mu)} & \frac{6EI}{L^2(1+\mu)} & 0 & -\frac{12EI}{L^3(1+\mu)} & \frac{6EI}{L^2(1+\mu)} \\ 0 & \frac{6EI}{L^2(1+\mu)} & \frac{4EI(1+\frac{\mu}{4})}{L(1+\mu)} & 0 & -\frac{6EI}{L^2(1+\mu)} & \frac{2EI(1-\frac{\mu}{2})}{L(1+\mu)} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3(1+\mu)} & -\frac{6EI}{L^2(1+\mu)} & 0 & \frac{12EI}{L^3(1+\mu)} & -\frac{6EI}{L^2(1+\mu)} \\ 0 & \frac{6EI}{L^2(1+\mu)} & \frac{2EI(1-\frac{\mu}{2})}{L(1+\mu)} & 0 & -\frac{6EI}{L^2(1+\mu)} & \frac{4EI(1+\frac{\mu}{4})}{L(1+\mu)} \end{bmatrix}$$

with

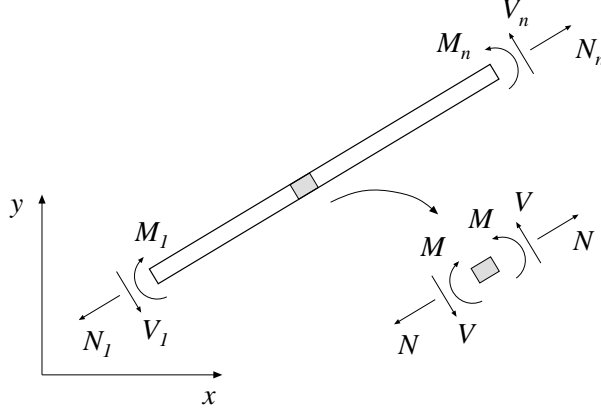
$$\mu = \frac{12EI}{L^2 G A k_s}$$

## Two dimensional Timoshenko beam element

beam2ts

### Purpose:

Compute section forces in a two dimensional Timoshenko beam element.



### Syntax:

```
es=beam2ts(ex,ey,ep,ed)
es=beam2ts(ex,ey,ep,ed,eq)
[es,edi,eci]=beam2ts(ex,ey,ep,ed,eq,n)
```

### Description:

beam2ts computes the section forces and displacements in local directions along the beam element beam2t.

The input variables **ex**, **ey**, **ep** and **eq** are defined in **beam2t**. The element displacements, stored in **ed**, are obtained by the function **extract**. If distributed loads are applied to the element, the variable **eq** must be included. The number of evaluation points for section forces and displacements are determined by **n**. If **n** is omitted, only the ends of the beam are evaluated.

The output variables

$$\mathbf{es} = [ \mathbf{N} \ \mathbf{V} \ \mathbf{M} ] \quad \mathbf{edi} = [ \bar{\mathbf{u}} \ \bar{\mathbf{v}} ] \quad \mathbf{eci} = [ \bar{\mathbf{x}} ]$$

consist of column matrices that contain the section forces, the displacements and rotation of the cross section (note that the rotation  $\theta$  is not equal to  $\frac{dv}{dx}$ ), and the evaluation points on the local x-axis. The explicit matrix expressions are

$$\mathbf{es} = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \\ \vdots & \vdots & \vdots \\ N_n & V_n & M_n \end{bmatrix} \quad \mathbf{edi} = \begin{bmatrix} \bar{u}_1 & \bar{v}_1 & \theta_1 \\ \bar{u}_2 & \bar{v}_2 & \theta_2 \\ \vdots & \vdots & \vdots \\ \bar{u}_n & \bar{v}_n & \theta_n \end{bmatrix} \quad \mathbf{eci} = \begin{bmatrix} 0 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_{n-1} \\ L \end{bmatrix}$$

where  $L$  is the length of the beam element.

**beam2ts**

**Two dimensional Timoshenko beam element**

---

**Theory:**

The evaluation of the section forces is based on the solutions of the basic equations

$$EA \frac{d^2 \bar{u}}{d\bar{x}^2} + q_{\bar{x}} = 0 \quad EI \frac{d^3 \theta}{d\bar{x}^3} - q_{\bar{y}} = 0 \quad EI \frac{d^4 \bar{v}}{d\bar{x}^4} - q_{\bar{y}} = 0$$

(The equations are valid if  $q_{\bar{y}}$  is not more than a linear function of  $\bar{x}$ ). From these equations, the displacements along the beam element are obtained as the sum of the homogeneous and the particular solutions

$$\mathbf{u} = \begin{bmatrix} \bar{u}(\bar{x}) \\ \bar{v}(\bar{x}) \\ \theta(\bar{x}) \end{bmatrix} = \mathbf{u}_h + \mathbf{u}_p$$

where

$$\mathbf{u}_h = \bar{\mathbf{N}} \mathbf{C}^{-1} \mathbf{G} \mathbf{a}^e \quad \mathbf{u}_p = \begin{bmatrix} \bar{u}_p(\bar{x}) \\ \bar{v}_p(\bar{x}) \\ \theta_p(\bar{x}) \end{bmatrix} = \begin{bmatrix} \frac{q_{\bar{x}} L \bar{x}}{2EA} (1 - \frac{\bar{x}}{L}) \\ \frac{q_{\bar{y}} L^2 \bar{x}^2}{24EI} (1 - \frac{\bar{x}}{L})^2 + \frac{q_{\bar{y}} L \bar{x}}{2GA k_s} (1 - \frac{\bar{x}}{L}) \\ \frac{q_{\bar{y}} L^2 \bar{x}}{12EI} (1 - \frac{2\bar{x}}{L}) (1 - \frac{\bar{x}}{L}) \end{bmatrix}$$

and

$$\bar{\mathbf{N}} = \begin{bmatrix} 1 & \bar{x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 \\ 0 & 0 & 0 & 1 & 2\bar{x} & 3(\bar{x}^2 + 2\alpha) \end{bmatrix} \quad \alpha = \frac{EI}{GA k_s}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 6\alpha \\ 1 & L & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & L^2 & L^3 \\ 0 & 0 & 0 & 1 & 2L & 3(L^2 + 2\alpha) \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_6 \end{bmatrix}$$

The transformation matrix  $\mathbf{G}$  and nodal displacements  $\mathbf{a}^e$  are described in **beam2e**. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.

Finally the section forces are obtained from

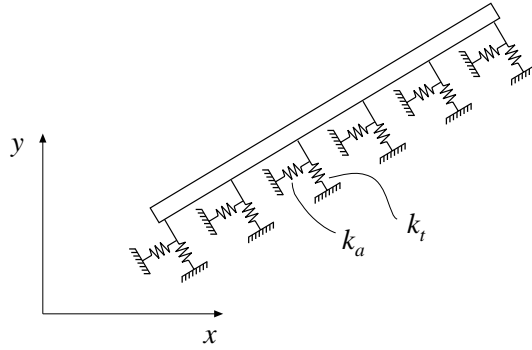
$$N = EA \frac{d\bar{u}}{d\bar{x}} \quad V = GA k_s (\frac{d\bar{v}}{d\bar{x}} - \theta) \quad M = EI \frac{d\theta}{d\bar{x}}$$

**Two dimensional beam element****beam2w**

---

**Purpose:**

Compute element stiffness matrix for a two dimensional beam element on elastic foundation.

**Syntax:**

```
Ke=beam2w(ex,ey,ep)
[Ke,fe]=beam2w(ex,ey,ep,eq)
```

**Description:**

beam2w provides the global element stiffness matrix **Ke** for a two dimensional beam element on elastic foundation.

The input variables **ex** and **ey** are described in **beam2e**, and

$$\mathbf{ep} = [ E \ A \ I \ k_a \ k_t ]$$

contains the modulus of elasticity  $E$ , the cross section area  $A$ , the moment of inertia  $I$ , the spring stiffness in the axial direction  $k_a$ , and the spring stiffness in the transverse direction  $k_t$ .

The element load vector **fe** can also be computed if uniformly distributed loads are applied to the element. The optional input variable **eq**, described in **beam2e**, contains the distributed loads.

**beam2w**

**Two dimensional beam element**

---

**Theory:**

The element stiffness matrix  $\mathbf{K}^e$ , stored in  $\mathbf{Ke}$ , is computed according to

$$\mathbf{K}^e = \mathbf{G}^T (\bar{\mathbf{K}}^e + \bar{\mathbf{K}}_s^e) \mathbf{G}$$

where  $\mathbf{G}$  and  $\bar{\mathbf{K}}^e$  are described in beam2e.

The matrix  $\bar{\mathbf{K}}_s^e$  is given by

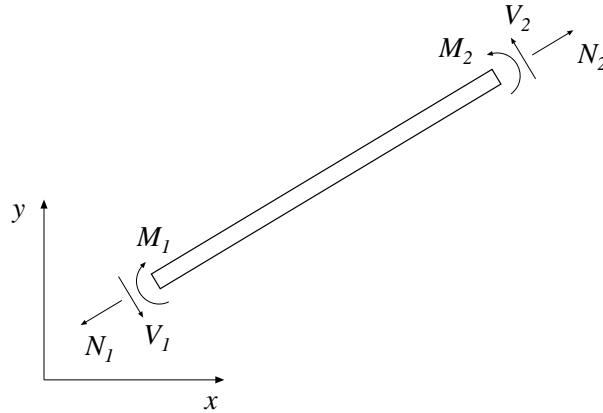
$$\bar{\mathbf{K}}_s^e = \frac{L}{420} \begin{bmatrix} 140k_a & 0 & 0 & 70k_a & 0 & 0 \\ 0 & 156k_t & 22k_t L & 0 & 54k_t & -13k_t L \\ 0 & 22k_t L & 4k_t L^2 & 0 & 13k_t L & -3k_t L^2 \\ 70k_a & 0 & 0 & 140k_a & 0 & 0 \\ 0 & 54k_t & 13k_t L & 0 & 156k_t & -22k_t L \\ 0 & -13k_t L & -3k_t L^2 & 0 & -22k_t L & 4k_t L^2 \end{bmatrix}$$

where the length

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

**Two dimensional beam element****beam2ws****Purpose:**

Compute section forces in a two dimensional beam element on elastic foundation.

**Syntax:**

```
es=beam2ws(ex,ey,ep,ed)
es=beam2ws(ex,ey,ep,ed,eq)
```

**Description:**

**beam2ws** computes the section forces at the ends of the beam element on elastic foundation **beam2w**.

The input variables **ex**, **ey**, **ep** and **eq** are defined in **beam2w**, and the element displacements, stored in **ed**, are obtained by the function **extract**. If distributed loads are applied to the element the variable **eq** must be included.

The output variable

$$\mathbf{es} = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \end{bmatrix}$$

contains the section forces at the ends of the beam.

**beam2ws**

**Two dimensional beam element**

---

**Theory:**

The section forces at the ends of the beam are obtained from the element force vector

$$\bar{\mathbf{P}} = [ -N_1 \quad -V_1 \quad -M_1 \quad N_2 \quad V_2 \quad M_2 ]^T$$

computed according to

$$\bar{\mathbf{P}} = (\bar{\mathbf{K}}^e + \bar{\mathbf{K}}_s^e) \mathbf{G} \mathbf{a}^e - \bar{\mathbf{f}}_l^e$$

The matrices  $\bar{\mathbf{K}}^e$  and  $\mathbf{G}$ , are described in **beam2e**, and the matrix  $\bar{\mathbf{K}}_s^e$  is described in **beam2w**. The nodal displacements

$$\mathbf{a}^e = [ u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 ]^T$$

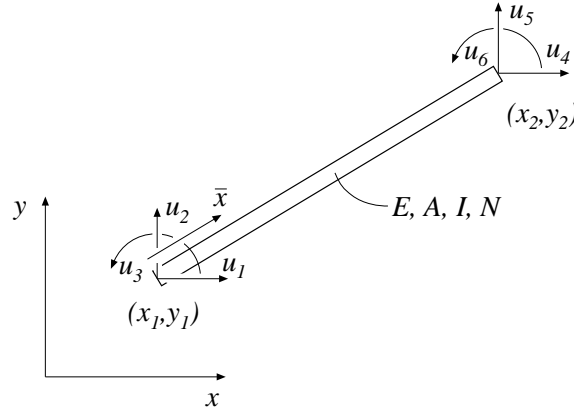
are shown in **beam2e**. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.

## Two dimensional beam element

beam2g

### Purpose:

Compute element stiffness matrix for a two dimensional nonlinear beam element.



### Syntax:

```
Ke=beam2g(ex,ey,ep,N)
[Ke,fe]=beam2g(ex,ey,ep,N,eq)
```

### Description:

beam2g provides the global element stiffness matrix **Ke** for a two dimensional beam element with respect to geometrical nonlinearity.

The input variables **ex**, **ey**, and **ep** are described in **beam2e**, and

$$\mathbf{N} = [ N ]$$

contains the value of the predefined normal force  $N$ , which is positive in tension.

The element load vector **fe** can also be computed if a uniformly distributed transverse load is applied to the element. The optional input variable

$$\mathbf{eq} = [ q_{\bar{y}} ]$$

then contains the distributed transverse load per unit length,  $q_{\bar{y}}$ . Note that **eq** is a scalar and not a vector as in **beam2e**



beam2g

Two dimensional beam element

---

**Theory:**

The element stiffness matrix  $\mathbf{K}^e$ , stored in the variable  $\mathbf{Ke}$ , is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where  $\mathbf{G}$  is described in beam2e, and  $\bar{\mathbf{K}}^e$  is given by

$$\bar{\mathbf{K}}^e = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} \phi_5 & \frac{6EI}{L^2} \phi_2 & 0 & -\frac{12EI}{L^3} \phi_5 & \frac{6EI}{L^2} \phi_2 \\ 0 & \frac{6EI}{L^2} \phi_2 & \frac{4EI}{L} \phi_3 & 0 & -\frac{6EI}{L^2} \phi_2 & \frac{2EI}{L} \phi_4 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} \phi_5 & -\frac{6EI}{L^2} \phi_2 & 0 & \frac{12EI}{L^3} \phi_5 & -\frac{6EI}{L^2} \phi_2 \\ 0 & \frac{6EI}{L^2} \phi_2 & \frac{2EI}{L} \phi_4 & 0 & -\frac{6EI}{L^2} \phi_2 & \frac{4EI}{L} \phi_3 \end{bmatrix}$$

For axial compression ( $N < 0$ ), we have

$$\begin{aligned} \phi_1 &= \frac{kL}{2} \cot \frac{kL}{2} & \phi_2 &= \frac{1}{12} \frac{k^2 L^2}{(1 - \phi_1)} & \phi_3 &= \frac{1}{4} \phi_1 + \frac{3}{4} \phi_2 \\ \phi_4 &= -\frac{1}{2} \phi_1 + \frac{3}{2} \phi_2 & \phi_5 &= \phi_1 \phi_2 \end{aligned}$$

with

$$k = \frac{\pi}{L} \sqrt{\rho}$$

For axial tension ( $N > 0$ ), we have

$$\begin{aligned} \phi_1 &= \frac{kL}{2} \coth \frac{kL}{2} & \phi_2 &= -\frac{1}{12} \frac{k^2 L^2}{(1 - \phi_1)} & \phi_3 &= \frac{1}{4} \phi_1 + \frac{3}{4} \phi_2 \\ \phi_4 &= -\frac{1}{2} \phi_1 + \frac{3}{2} \phi_2 & \phi_5 &= \phi_1 \phi_2 \end{aligned}$$

with

$$k = \frac{\pi}{L} \sqrt{-\rho}$$

The parameter  $\rho$  is given by

$$\rho = -\frac{NL^2}{\pi^2 EI}$$

**Two dimensional beam element****beam2g**

The equivalent nodal loads  $\mathbf{f}_l^e$  stored in the variable **fe** are computed according to

$$\mathbf{f}_l^e = \mathbf{G}^T \bar{\mathbf{f}}_l^e$$

where

$$\bar{\mathbf{f}}_l^e = qL \begin{bmatrix} 0 & \frac{1}{2} & \frac{L}{12}\psi & 0 & \frac{1}{2} & -\frac{L}{12}\psi \end{bmatrix}^T$$

For an axial compressive force, we have

$$\psi = 6 \left( \frac{2}{(kL)^2} - \frac{1 + \cos kL}{kL \sin kL} \right)$$

and for an axial tensile force

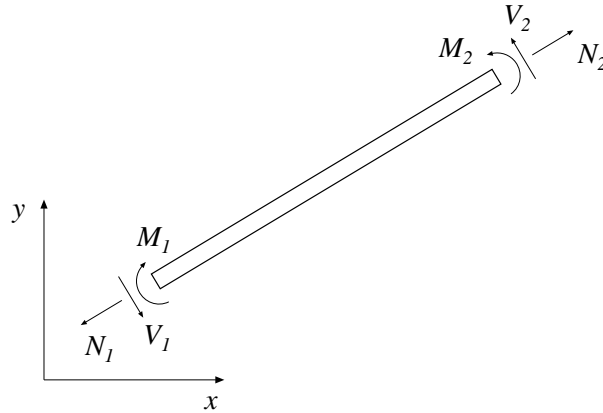
$$\psi = 6 \left( \frac{1 + \cosh kL}{kL \sinh kL} - \frac{2}{(kL)^2} \right)$$

**beam2gs****Two dimensional beam element**

---

**Purpose:**

Compute section forces in a two dimensional nonlinear beam element.

**Syntax:**

```
es=beam2gs(ex,ey,ep,ed,N)
es=beam2gs(ex,ey,ep,ed,N,eq)
```

**Description:**

`beam2gs` computes the section forces at the ends of the nonlinear beam element `beam2g`.

The input variables `ex`, `ey`, and `ep` are defined in `beam2e`, and the variables `N` and `eq` in `beam2g`. The element displacements, stored in `ed`, are obtained by the function `extract`. If a distributed transversal load is applied to the element, the variable `eq` must be included.

The output variable

$$es = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \end{bmatrix}$$

contains the section forces at the ends of the beam.

**Two dimensional beam element****beam2gs****Theory:**

The section forces at the ends of the beam are obtained from the element force vector

$$\bar{\mathbf{P}} = [ -N_1 \quad -V_1 \quad -M_1 \quad N_2 \quad V_2 \quad M_2 ]^T$$

computed according to

$$\bar{\mathbf{P}} = \bar{\mathbf{K}}^e \mathbf{G} \mathbf{a}^e - \bar{\mathbf{f}}_l^e$$

The matrix  $\mathbf{G}$  is described in **beam2e**. The matrix  $\bar{\mathbf{K}}^e$  and the nodal displacements

$$\mathbf{a}^e = [ u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 ]^T$$

are described in **beam2g**. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.

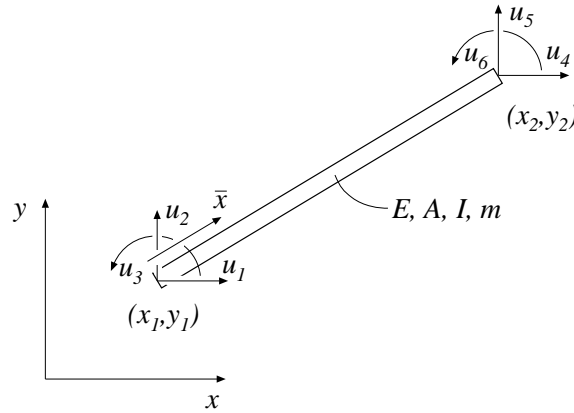
## beam2d

## Two dimensional beam element

---

### Purpose:

Compute element stiffness, mass and damping matrices for a two dimensional beam element.



### Syntax:

```
[Ke,Me]=beam2d(ex,ey,ep)
[Ke,Me,Ce]=beam2d(ex,ey,ep)
```

### Description:

beam2d provides the global element stiffness matrix **Ke**, the global element mass matrix **Me**, and the global element damping matrix **Ce**, for a two dimensional beam element.

The input variables **ex** and **ey** are described in **beam2e**, and

$$\mathbf{ep} = [ E \ A \ I \ m \ [ \ a_0 \ a_1 ] ]$$

contains the modulus of elasticity  $E$ , the cross section area  $A$ , the moment of inertia  $I$ , the mass per unit length  $m$ , and the Raleigh damping coefficients  $a_0$  and  $a_1$ . If  $a_0$  and  $a_1$  are omitted, the element damping matrix **Ce** is not computed.

---

Two dimensional beam element

beam2d

---

**Theory:**

The element stiffness matrix  $\mathbf{K}^e$ , the element mass matrix  $\mathbf{M}^e$  and the element damping matrix  $\mathbf{C}^e$ , stored in the variables **Ke**, **Me** and **Ce**, respectively, are computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G} \quad \mathbf{M}^e = \mathbf{G}^T \bar{\mathbf{M}}^e \mathbf{G} \quad \mathbf{C}^e = \mathbf{G}^T \bar{\mathbf{C}}^e \mathbf{G}$$

where  $\mathbf{G}$  and  $\bar{\mathbf{K}}^e$  are described in **beam2e**.

The matrix  $\bar{\mathbf{M}}^e$  is given by

$$\bar{\mathbf{M}}^e = \frac{mL}{420} \begin{bmatrix} 140 & 0 & 0 & 70 & 0 & 0 \\ 0 & 156 & 22L & 0 & 54 & -13L \\ 0 & 22L & 4L^2 & 0 & 13L & -3L^2 \\ 70 & 0 & 0 & 140 & 0 & 0 \\ 0 & 54 & 13L & 0 & 156 & -22L \\ 0 & -13L & -3L^2 & 0 & -22L & 4L^2 \end{bmatrix}$$

and the matrix  $\bar{\mathbf{C}}^e$  is computed by combining  $\bar{\mathbf{K}}^e$  and  $\bar{\mathbf{M}}^e$

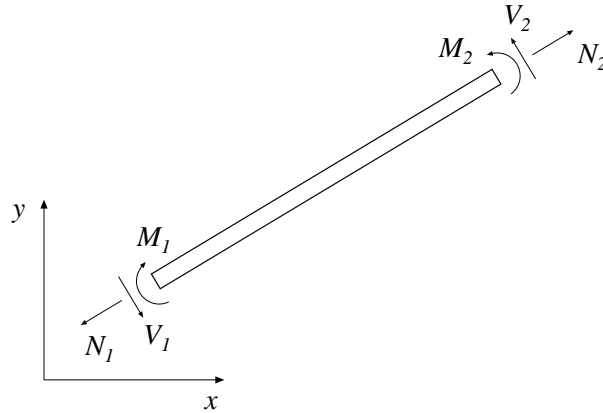
$$\bar{\mathbf{C}}^e = a_0 \bar{\mathbf{M}}^e + a_1 \bar{\mathbf{K}}^e$$

**beam2ds****Two dimensional beam element**

---

**Purpose:**

Compute section forces for a two dimensional beam element in dynamic analysis.

**Syntax:**

```
es=beam2ds(ex,ey,ep,ed,ev,ea)
```

**Description:**

`beam2ds` computes the section forces at the ends of the dynamic beam element `beam2d`.

The input variables `ex`, `ey`, and `ep` are defined in `beam2d`. The element displacements, the element velocities, and the element accelerations, stored in `ed`, `ev`, and `ea` respectively, are obtained by the function `extract`.

The output variable

$$\mathbf{es} = \begin{bmatrix} N_1 & V_1 & M_1 \\ N_2 & V_2 & M_2 \end{bmatrix}$$

contains the section forces at the ends of the beam.

**Two dimensional beam element****beam2ds****Theory:**

The section forces at the ends of the beam are obtained from the element force vector

$$\bar{\mathbf{P}} = [ -N_1 \quad -V_1 \quad -M_1 \quad N_2 \quad V_2 \quad M_2 ]^T$$

computed according to

$$\bar{\mathbf{P}} = \bar{\mathbf{K}}^e \mathbf{G} \mathbf{a}^e + \bar{\mathbf{C}}^e \mathbf{G} \dot{\mathbf{a}}^e + \bar{\mathbf{M}}^e \mathbf{G} \ddot{\mathbf{a}}^e$$

The matrices  $\bar{\mathbf{K}}^e$  and  $\mathbf{G}$  are described in **beam2e**, and the matrices  $\bar{\mathbf{M}}^e$  and  $\bar{\mathbf{C}}^e$  are described in **beam2d**. The nodal displacements

$$\mathbf{a}^e = [ u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 ]^T$$

shown in **beam2d** also define the directions of the nodal velocities

$$\dot{\mathbf{a}}^e = [ \dot{u}_1 \quad \dot{u}_2 \quad \dot{u}_3 \quad \dot{u}_4 \quad \dot{u}_5 \quad \dot{u}_6 ]^T$$

and the nodal accelerations

$$\ddot{\mathbf{a}}^e = [ \ddot{u}_1 \quad \ddot{u}_2 \quad \ddot{u}_3 \quad \ddot{u}_4 \quad \ddot{u}_5 \quad \ddot{u}_6 ]^T$$

Note that the transposes of  $\mathbf{a}^e$ ,  $\dot{\mathbf{a}}^e$ , and  $\ddot{\mathbf{a}}^e$  are stored in **ed**, **ev**, and **ea** respectively.

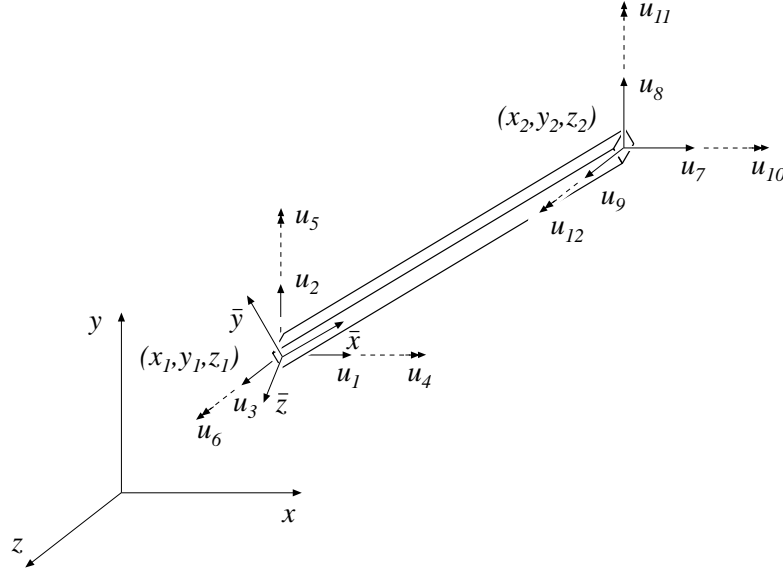


## beam3e

## Three dimensional beam element

### Purpose:

Compute element stiffness matrix for a three dimensional beam element.



### Syntax:

```
Ke=beam3e(ex,ey,ez,eo,ep)
[Ke,fe]=beam3e(ex,ey,ez,eo,ep,eq)
```

### Description:

beam3e provides the global element stiffness matrix **Ke** for a three dimensional beam element.

The input variables

$$\begin{aligned} \mathbf{ex} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \\ \mathbf{ey} &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \\ \mathbf{ez} &= \begin{bmatrix} z_1 & z_2 \end{bmatrix} \end{aligned} \quad \mathbf{eo} = \begin{bmatrix} x_{\bar{z}} & y_{\bar{z}} & z_{\bar{z}} \end{bmatrix}$$

supply the element nodal coordinates  $x_1, y_1$ , etc. as well as the direction of the local beam coordinate system  $(\bar{x}, \bar{y}, \bar{z})$ . By giving a global vector  $(x_{\bar{z}}, y_{\bar{z}}, z_{\bar{z}})$  parallel with the positive local  $\bar{z}$  axis of the beam, the local beam coordinate system is defined. The variable

$$\mathbf{ep} = \begin{bmatrix} E & G & A & I_{\bar{y}} & I_{\bar{z}} & K_v \end{bmatrix}$$

supplies the modulus of elasticity  $E$ , the shear modulus  $G$ , the cross section area  $A$ , the moment of inertia with respect to the  $\bar{y}$  axis  $I_{\bar{y}}$ , the moment of inertia with respect to the  $\bar{z}$  axis  $I_{\bar{z}}$ , and St Venant torsional stiffness  $K_v$ .

## Three dimensional beam element

beam3e

The element load vector  $\mathbf{fe}$  can also be computed if uniformly distributed loads are applied to the element. The optional input variable

$$\mathbf{eq} = [q_{\bar{x}} \ q_{\bar{y}} \ q_{\bar{z}} \ q_{\bar{\omega}}]$$

then contains the distributed loads. The positive directions of  $q_{\bar{x}}$ ,  $q_{\bar{y}}$ , and  $q_{\bar{z}}$  follow the local beam coordinate system. The distributed torque  $q_{\bar{\omega}}$  is positive if directed in the local  $\bar{x}$ -direction, i.e. from local  $\bar{y}$  to local  $\bar{z}$ . All the loads are per unit length.

**Theory:**

The element stiffness matrix  $\mathbf{K}^e$  is computed according to

$$\mathbf{K}^e = \mathbf{G}^T \bar{\mathbf{K}}^e \mathbf{G}$$

where

$$\bar{\mathbf{K}}^e = \begin{bmatrix} k_1 & 0 & 0 & 0 & 0 & 0 & -k_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_{\bar{x}}}{L^3} & 0 & 0 & 0 & \frac{6EI_{\bar{x}}}{L^2} & 0 & -\frac{12EI_{\bar{x}}}{L^3} & 0 & 0 & 0 & \frac{6EI_{\bar{x}}}{L^2} \\ 0 & 0 & \frac{12EI_{\bar{y}}}{L^3} & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 & 0 & 0 & -\frac{12EI_{\bar{y}}}{L^3} & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 \\ 0 & 0 & 0 & k_2 & 0 & 0 & 0 & 0 & 0 & -k_2 & 0 & 0 \\ 0 & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{4EI_{\bar{y}}}{L} & 0 & 0 & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{2EI_{\bar{y}}}{L} & 0 \\ 0 & \frac{6EI_{\bar{x}}}{L^2} & 0 & 0 & 0 & \frac{4EI_{\bar{x}}}{L} & 0 & -\frac{6EI_{\bar{x}}}{L^2} & 0 & 0 & 0 & \frac{2EI_{\bar{x}}}{L} \\ -k_1 & 0 & 0 & 0 & 0 & 0 & k_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI_{\bar{x}}}{L^3} & 0 & 0 & 0 & -\frac{6EI_{\bar{x}}}{L^2} & 0 & \frac{12EI_{\bar{x}}}{L^3} & 0 & 0 & 0 & -\frac{6EI_{\bar{x}}}{L^2} \\ 0 & 0 & -\frac{12EI_{\bar{y}}}{L^3} & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 & 0 & 0 & \frac{12EI_{\bar{y}}}{L^3} & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 \\ 0 & 0 & 0 & -k_2 & 0 & 0 & 0 & 0 & 0 & k_2 & 0 & 0 \\ 0 & 0 & -\frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{2EI_{\bar{y}}}{L} & 0 & 0 & 0 & \frac{6EI_{\bar{y}}}{L^2} & 0 & \frac{4EI_{\bar{y}}}{L} & 0 \\ 0 & \frac{6EI_{\bar{x}}}{L^2} & 0 & 0 & 0 & \frac{2EI_{\bar{x}}}{L} & 0 & -\frac{6EI_{\bar{x}}}{L^2} & 0 & 0 & 0 & \frac{4EI_{\bar{x}}}{L} \end{bmatrix}$$

in which  $k_1 = \frac{EA}{L}$  and  $k_2 = \frac{GK_v}{L}$ , and where

$$\mathbf{G} = \begin{bmatrix} n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{x}} & n_{y\bar{x}} & n_{z\bar{x}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{y}} & n_{y\bar{y}} & n_{z\bar{y}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_{x\bar{z}} & n_{y\bar{z}} & n_{z\bar{z}} \end{bmatrix}$$

**beam3e**
**Three dimensional beam element**


---

The element length  $L$  is computed according to

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

In the transformation matrix  $\mathbf{G}$ ,  $n_{x\bar{x}}$  specifies the cosine of the angle between the  $x$  axis and  $\bar{x}$  axis, and so on.

The element load vector  $\mathbf{f}_l^e$ , stored in `fe`, is computed according to

$$\mathbf{f}_l^e = \mathbf{G}^T \bar{\mathbf{f}}_l^e$$

where

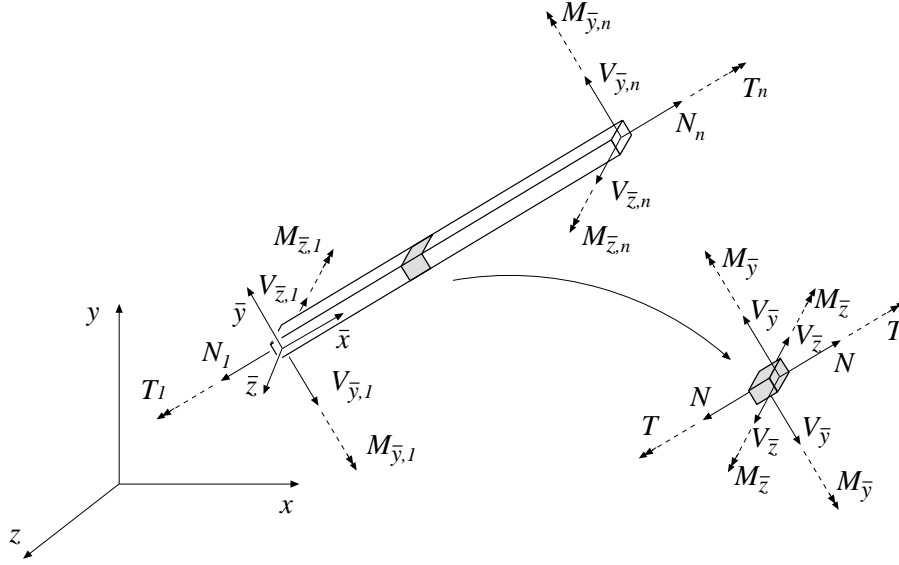
$$\bar{\mathbf{f}}_l^e = \begin{bmatrix} \frac{q_{\bar{x}}L}{2} \\ \frac{q_{\bar{y}}L}{2} \\ \frac{q_{\bar{z}}L}{2} \\ \frac{q_{\bar{\omega}}L}{2} \\ -\frac{q_{\bar{z}}L^2}{12} \\ \frac{q_{\bar{y}}L^2}{12} \\ \frac{q_{\bar{x}}L}{2} \\ \frac{q_{\bar{y}}L}{2} \\ \frac{q_{\bar{z}}L}{2} \\ \frac{q_{\bar{\omega}}L}{2} \\ \frac{q_{\bar{z}}L^2}{12} \\ -\frac{q_{\bar{y}}L^2}{12} \end{bmatrix}$$

### Three dimensional beam element

beam3s

#### Purpose:

Compute section forces in a three dimensional beam element.



#### Syntax:

```
es=beam3s(ex,ey,e,ez,eo,ep,ed)
es=beam3s(ex,ey,e,ez,eo,ep,ed,eq)
[es,edi,eci]=beam3s(ex,ey,e,ez,eo,ep,ed,eq,n)
```

#### Description:

beam3s computes the section forces and displacements in local directions along the beam element beam3e.

The input variables  $ex$ ,  $ey$ ,  $e$ ,  $ez$ ,  $eo$ , and  $ep$  are defined in beam3e, and the element displacements, stored in  $ed$ , are obtained by the function extract. If distributed loads are applied to the element, the variable  $eq$  must be included. The number of evaluation points for section forces and displacements are determined by  $n$ . If  $n$  is omitted, only the ends of the beam are evaluated.

The output variables

$$es = [ \mathbf{N} \quad \mathbf{V}_{\bar{y}} \quad \mathbf{V}_{\bar{z}} \quad \mathbf{T} \quad \mathbf{M}_{\bar{y}} \quad \mathbf{M}_{\bar{z}} ] \quad edi = [ \bar{u} \quad \bar{v} \quad \bar{w} \quad \bar{\varphi} ] \quad eci = [ \bar{x} ]$$

consist of column matrices that contain the section forces, the displacements, and the evaluation points on the local  $\bar{x}$ -axis. The explicit matrix expressions are

$$es = \begin{bmatrix} N_1 & V_{\bar{y}1} & V_{\bar{z}1} & T & M_{\bar{y}1} & M_{\bar{z}1} \\ N_2 & V_{\bar{y}2} & V_{\bar{z}2} & T & M_{\bar{y}2} & M_{\bar{z}2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ N_n & V_{\bar{y}n} & V_{\bar{z}n} & T & M_{\bar{y}n} & M_{\bar{z}n} \end{bmatrix}$$

**beam3s**

**Three dimensional beam element**

$$\text{edi} = \begin{bmatrix} \bar{u}_1 & \bar{v}_1 & \bar{w}_1 & \bar{\varphi}_1 \\ \bar{u}_2 & \bar{v}_2 & \bar{w}_2 & \bar{\varphi}_2 \\ \vdots & \vdots & \vdots & \vdots \\ \bar{u}_n & \bar{v}_n & \bar{w}_n & \bar{\varphi}_n \end{bmatrix} \quad \text{eci} = \begin{bmatrix} 0 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_{n-1} \\ L \end{bmatrix}$$

where  $L$  is the length of the beam element.

**Theory:**

The evaluation of the section forces is based on the solutions of the basic equations

$$\begin{aligned} EA \frac{d^2 \bar{u}}{d\bar{x}^2} + q_{\bar{x}} &= 0 & EI_z \frac{d^4 \bar{v}}{d\bar{x}^4} - q_{\bar{y}} &= 0 \\ EI_y \frac{d^4 \bar{w}}{d\bar{x}^4} - q_{\bar{z}} &= 0 & GK_v \frac{d^2 \bar{\varphi}}{d\bar{x}^2} + q_{\bar{\omega}} &= 0 \end{aligned}$$

From these equations, the displacements along the beam element are obtained as the sum of the homogeneous and the particular solutions

$$\mathbf{u} = \begin{bmatrix} \bar{u}(\bar{x}) \\ \bar{v}(\bar{x}) \\ \bar{w}(\bar{x}) \\ \bar{\varphi}(\bar{x}) \end{bmatrix} = \mathbf{u}_h + \mathbf{u}_p$$

where

$$\mathbf{u}_h = \bar{\mathbf{N}} \mathbf{C}^{-1} \mathbf{G} \mathbf{a}^e \quad \mathbf{u}_p = \begin{bmatrix} \bar{u}_p(\bar{x}) \\ \bar{v}_p(\bar{x}) \\ \bar{w}_p(\bar{x}) \\ \bar{\varphi}_p(\bar{x}) \end{bmatrix} = \begin{bmatrix} \frac{q_{\bar{x}} L \bar{x}}{2EA} \left(1 - \frac{\bar{x}}{L}\right) \\ \frac{q_{\bar{y}} L^2 \bar{x}^2}{24EI_z} \left(1 - \frac{\bar{x}}{L}\right)^2 \\ \frac{q_{\bar{z}} L^2 \bar{x}^2}{24EI_y} \left(1 - \frac{\bar{x}}{L}\right)^2 \\ \frac{q_{\bar{\omega}} L \bar{x}}{2GK_v} \left(1 - \frac{\bar{x}}{L}\right) \end{bmatrix}$$

and

$$\bar{\mathbf{N}} = \begin{bmatrix} 1 & \bar{x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \bar{x} & \bar{x}^2 & \bar{x}^3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \bar{x} \end{bmatrix}$$

Three dimensional beam element

beam3s

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & L^2 & L^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & L & L^2 & L^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & L & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2L & 3L^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2L & 3L^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{a}^e = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{12} \end{bmatrix}$$

The transformation matrix  $\mathbf{G}^e$  and nodal displacements  $\mathbf{a}^e$  are described in beam3e. Note that the transpose of  $\mathbf{a}^e$  is stored in **ed**.

Finally the section forces are obtained from

$$\begin{aligned} N &= EA \frac{d\bar{u}}{d\bar{x}} & V_{\bar{y}} &= -EI_z \frac{d^3\bar{v}}{d\bar{x}^3} & V_{\bar{z}} &= -EI_y \frac{d^3\bar{w}}{d\bar{x}^3} \\ T &= GK_v \frac{d\bar{\varphi}}{d\bar{x}} & M_{\bar{y}} &= -EI_y \frac{d^2\bar{w}}{d\bar{x}^2} & M_{\bar{z}} &= EI_z \frac{d^2\bar{v}}{d\bar{x}^2} \end{aligned}$$

**Examples:**

Section forces or element displacements can easily be plotted. The bending moment  $M_{\bar{y}}$  along the beam is plotted by

```
>> plot(eci,es(:,5))
```



5.7 Plate element

Only one plate element is currently available, a rectangular 12 dof element. The element presumes a linear elastic material which can be isotropic or anisotropic.

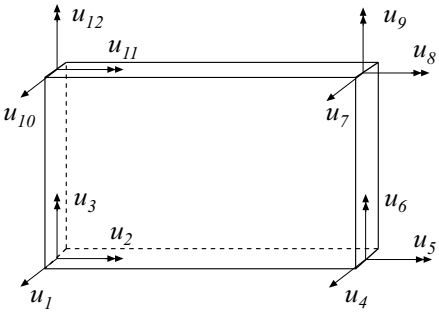
Plate elements	
 <p>Diagram of a rectangular plate element showing 12 degrees of freedom (DOFs) at the four nodes. The DOFs are labeled as follows:</p> <ul style="list-style-type: none"><li>Bottom-left node: <math>u_1</math> (horizontal), <math>u_2</math> (vertical), <math>u_3</math> (rotation)</li><li>Bottom-right node: <math>u_4</math> (horizontal), <math>u_5</math> (vertical), <math>u_6</math> (rotation)</li><li>Top-left node: <math>u_{10}</math> (horizontal), <math>u_{11}</math> (vertical), <math>u_{12}</math> (rotation)</li><li>Top-right node: <math>u_7</math> (horizontal), <math>u_8</math> (vertical), <math>u_9</math> (rotation)</li></ul> <p>The word "platte" is written below the diagram.</p>	

Plate functions	
platte	Compute element matrices
platr	Compute section forces

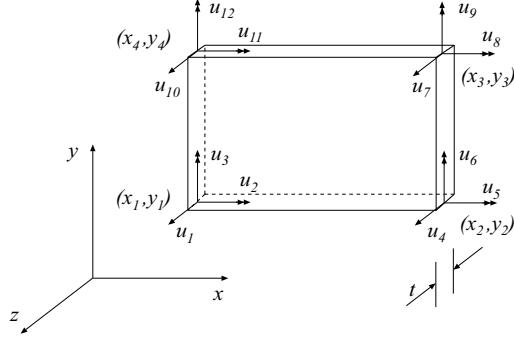


**platte**

**Plate element**

**Purpose:**

Compute element stiffness matrix for a rectangular plate element.



**Syntax:**

```
Ke=platte(ex,ey,ep,D)
[Ke,fe]=platte(ex,ey,ep,D,eq)
```

**Description:**

**platte** provides an element stiffness matrix **Ke**, and an element load vector **fe**, for a rectangular plate element. This element can only be used if the element edges are parallel to the coordinate axis.

The element nodal coordinates  $x_1, y_1, x_2$  etc. are supplied to the function by **ex** and **ey**, the element thickness  $t$  by **ep**, and the material properties by the constitutive matrix **D**. Any arbitrary **D**-matrix with dimensions  $(3 \times 3)$  and valid for plane stress may be given. For an isotropic elastic material the constitutive matrix can be formed by the function **hooke**, see Section 4.

$$\begin{aligned} \mathbf{ex} &= [x_1 \ x_2 \ x_3 \ x_4] \\ \mathbf{ey} &= [y_1 \ y_2 \ y_3 \ y_4] \end{aligned} \quad \mathbf{ep} = [t] \quad \mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{bmatrix}$$

If a uniformly distributed load is applied to the element, the element load vector **fe** is computed. The input variable

$$\mathbf{eq} = [q_z]$$

then contains the load  $q_z$  per unit area in the  $z$ -direction.

Plate element

platte

Theory:

The element stiffness matrix  $\mathbf{K}^e$  and the element load vector  $\mathbf{f}_l^e$ , stored in  $\mathbf{Ke}$  and  $\mathbf{fe}$  respectively, are computed according to

$$\mathbf{K}^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{B}}^T \tilde{\mathbf{D}} \bar{\mathbf{B}} dA \mathbf{C}^{-1}$$

$$\mathbf{f}_l^e = (\mathbf{C}^{-1})^T \int_A \bar{\mathbf{N}}^T q_z dA$$

where the constitutive matrix

$$\tilde{\mathbf{D}} = \frac{t^3}{12} \mathbf{D}$$

and where  $\mathbf{D}$  is defined by  $\mathbf{D}$ .

The evaluation of the integrals for the rectangular plate element is based on the displacement approximation  $w(x, y)$  and is expressed in terms of the nodal variables  $u_1, u_2, \dots, u_{12}$  as

$$w(x, y) = \mathbf{N}^e \mathbf{a}^e = \bar{\mathbf{N}} \mathbf{C}^{-1} \mathbf{a}^e$$

where

$$\bar{\mathbf{N}} = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y & xy^2 & y^3 & x^3y & xy^3 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & -a & -b & a^2 & ab & b^2 & -a^3 & -a^2b & -ab^2 & -b^3 & a^3b & ab^3 \\ 0 & 0 & 1 & 0 & -a & -2b & 0 & a^2 & 2ab & 3b^2 & -a^3 & -3ab^2 \\ 0 & -1 & 0 & 2a & b & 0 & -3a^2 & -2ab & -b^2 & 0 & 3a^2b & b^3 \\ 1 & a & -b & a^2 & -ab & b^2 & a^3 & -a^2b & ab^2 & -b^3 & -a^3b & -ab^3 \\ 0 & 0 & 1 & 0 & a & -2b & 0 & a^2 & -2ab & 3b^2 & a^3 & 3ab^2 \\ 0 & -1 & 0 & -2a & b & 0 & -3a^2 & 2ab & -b^2 & 0 & 3a^2b & b^3 \\ 1 & a & b & a^2 & ab & b^2 & a^3 & a^2b & ab^2 & b^3 & a^3b & ab^3 \\ 0 & 0 & 1 & 0 & a & 2b & 0 & a^2 & 2ab & 3b^2 & a^3 & 3ab^2 \\ 0 & -1 & 0 & -2a & -b & 0 & -3a^2 & -2ab & -b^2 & 0 & -3a^2b & -b^3 \\ 1 & -a & b & a^2 & -ab & b^2 & -a^3 & a^2b & -ab^2 & b^3 & -a^3b & -ab^3 \\ 0 & 0 & 1 & 0 & -a & 2b & 0 & a^2 & -2ab & 3b^2 & -a^3 & -3ab^2 \\ 0 & -1 & 0 & 2a & -b & 0 & -3a^2 & 2ab & -b^2 & 0 & -3a^2b & -b^3 \end{bmatrix}$$

$$\mathbf{a}^e = [u_1 \quad u_2 \quad \dots \quad u_{12}]^T$$

and where

$$a = \frac{1}{2}(x_3 - x_1) \quad \text{and} \quad b = \frac{1}{2}(y_3 - y_1)$$

**platte**

**Plate element**

The matrix  $\bar{\mathbf{B}}$  is obtained as

$$\bar{\mathbf{B}} = \nabla^* \bar{\mathbf{N}} = \begin{bmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 6x & 2y & 0 & 0 & 6xy & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2x & 6y & 0 & 6xy \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 4x & 4y & 0 & 6x^2 & 6y^2 \end{bmatrix}$$

where

$$\nabla^* = \begin{bmatrix} \frac{\partial^2}{\partial x^2} \\ \frac{\partial^2}{\partial y^2} \\ 2 \frac{\partial^2}{\partial x \partial y} \end{bmatrix}$$

Evaluation of the integrals for the rectangular plate element is done analytically. Computation of the integrals for the element load vector  $\mathbf{f}_l^e$  yields

$$\mathbf{f}_l^e = q_z L_x L_y \left[ \frac{1}{4} \frac{L_y}{24} - \frac{L_x}{24} \frac{1}{4} \frac{L_y}{24} \frac{L_x}{24} \frac{1}{4} - \frac{L_y}{24} \frac{L_x}{24} \frac{1}{4} - \frac{L_y}{24} - \frac{L_x}{24} \right]^T$$

where

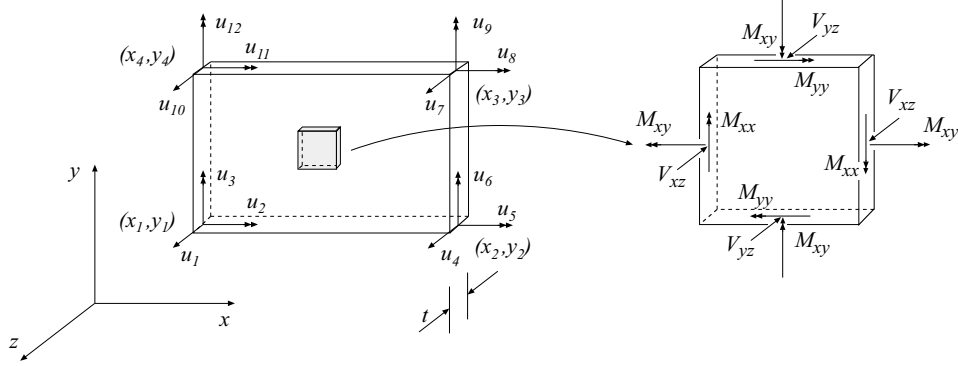
$$L_x = x_3 - x_1 \quad \text{and} \quad L_y = y_3 - y_1$$

Plate element

platrs

**Purpose:**

Compute section forces in a rectangular plate element.



**Syntax:**

`[es,et]=platrs(ex,ey,ep,D,ed)`

**Description:**

**platrs** computes the section forces **es** and the curvature matrix **et** in a rectangular plate element. The section forces and the curvatures are computed at the center of the element.

The input variables **ex**, **ey**, **ep** and **D** are defined in **platre**. The vector **ed** contains the nodal displacements  $\mathbf{a}^e$  of the element and is obtained by the function **extract** as

$$\mathbf{ed} = (\mathbf{a}^e)^T = [u_1 \ u_2 \ \dots \ u_{12}]$$

The output variables

$$\mathbf{es} = [\mathbf{M}^T \ \mathbf{V}^T] = [M_{xx} \ M_{yy} \ M_{xy} \ V_{xz} \ V_{yz}]$$

$$\mathbf{et} = \boldsymbol{\kappa}^T = [\kappa_{xx} \ \kappa_{yy} \ \kappa_{xy}]$$

contain the section forces and curvatures in global directions.

**platr**

**Plate element**

**Theory:**

The curvatures and the section forces are computed according to

$$\boldsymbol{\kappa} = \begin{bmatrix} \kappa_{xx} \\ \kappa_{yy} \\ \kappa_{xy} \end{bmatrix} = \bar{\mathbf{B}} \mathbf{C}^{-1} \mathbf{a}^e$$

$$\mathbf{M} = \begin{bmatrix} M_{xx} \\ M_{yy} \\ M_{xy} \end{bmatrix} = \tilde{\mathbf{D}} \boldsymbol{\kappa}$$

$$\mathbf{V} = \begin{bmatrix} V_{xz} \\ V_{yz} \end{bmatrix} = \tilde{\mathbf{V}} \mathbf{M}$$

where the matrices  $\tilde{\mathbf{D}}$ ,  $\bar{\mathbf{B}}$ ,  $\mathbf{C}$  and  $\mathbf{a}^e$  are described in `platre`, and where

$$\tilde{\mathbf{V}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

## 6 System functions

### 6.1 Introduction

The group of system functions comprises functions for the setting up, solving, and elimination of systems of equations. The functions are separated in two groups:

<b>Static system functions</b>
<b>Dynamic system functions</b>

Static system functions concern the linear system of equations

$$\mathbf{K}\mathbf{a} = \mathbf{f}$$

where  $\mathbf{K}$  is the global stiffness matrix and  $\mathbf{f}$  is the global load vector. Often used static system functions are `assem` and `solveq`. The function `assem` assembles the global stiffness matrix and `solveq` computes the global displacement vector  $\mathbf{a}$  considering the boundary conditions. It should be noted that  $\mathbf{K}$ ,  $\mathbf{f}$ , and  $\mathbf{a}$  also represent analogous quantities in systems others than structural mechanical systems. For example, in a heat flow problem  $\mathbf{K}$  represents the conductivity matrix,  $\mathbf{f}$  the heat flow, and  $\mathbf{a}$  the temperature.

Dynamic system functions are related to different aspects of linear dynamic systems of coupled ordinary differential equations according to

$$\mathbf{C} \dot{\mathbf{d}} + \mathbf{K} \mathbf{d} = \mathbf{f}$$

for first-order systems and

$$\mathbf{M} \ddot{\mathbf{d}} + \mathbf{C} \dot{\mathbf{d}} + \mathbf{K} \mathbf{d} = \mathbf{f}$$

for second-order systems. First-order systems occur typically in transient heat conduction and second-order systems occur in structural dynamics.



## 6.2 Static system functions

The group of static system functions comprises functions for setting up and solving the global system of equations. It also contains a function for eigenvalue analysis, a function for static condensation, a function for extraction of element displacements from the global displacement vector and a function for extraction of element coordinates.

The following functions are available for static analysis:

Static system functions	
<code>assem</code>	Assemble element matrices
<code>coordxtr</code>	Extract element coordinates from a global coordinate matrix.
<code>eigen</code>	Solve a generalized eigenvalue problem
<code>extract</code>	Extract values from a global vector
<code>insert</code>	Assemble element internal force vector
<code>solveq</code>	Solve a system of equations
<code>statcon</code>	Perform static condensation



**Purpose:**

Assemble element matrices.

$$\begin{array}{c}
 \begin{array}{cc} i & j \\ \left[ \begin{array}{cc} k_{ii}^e & k_{ij}^e \\ k_{ji}^e & k_{jj}^e \end{array} \right] & \begin{array}{c} i \\ j \end{array} \\ \mathbf{K}^e \\ i = dof_i \\ j = dof_j \end{array} \quad \longrightarrow \quad \begin{array}{c} \begin{array}{cc} i & j \\ \left[ \begin{array}{cccc} k_{11} & k_{12} & \vdots & \vdots \\ k_{21} & & \vdots & \vdots \\ \cdots & \cdots & k_{ii} + k_{ii}^e & k_{ij} + k_{ij}^e \\ \cdots & \cdots & k_{ji} + k_{ji}^e & k_{jj} + k_{jj}^e \\ & & \vdots & \vdots \\ & & \vdots & \vdots \end{array} \right] & \begin{array}{c} i \\ j \end{array} \\ \mathbf{K} \end{array} \end{array}
 \end{array}$$

**Syntax:**

$\mathbf{K} = \text{assem}(\text{edof}, \mathbf{K}, \mathbf{K}^e)$   
 $[\mathbf{K}, \mathbf{f}] = \text{assem}(\text{edof}, \mathbf{K}, \mathbf{K}^e, \mathbf{f}, \mathbf{f}^e)$

**Description:**

**assem** adds the element stiffness matrix  $\mathbf{K}^e$ , stored in **Ke**, to the structure stiffness matrix  $\mathbf{K}$ , stored in **K**, according to the topology matrix **edof**.

The element topology matrix **edof** is defined as

$$\text{edof} = [el \quad \underbrace{dof_1 \quad dof_2 \quad \dots \quad dof_{ned}}_{\text{global dof.}}]$$

where the first column contains the element number, and the columns 2 to  $(ned + 1)$  contain the corresponding global degrees of freedom ( $ned$  = number of element degrees of freedom).

In the case where the matrix  $\mathbf{K}^e$  is identical for several elements, assembling of these can be carried out simultaneously. Each row in **Edof** then represents one element, i.e.  $nel$  is the total number of considered elements.

$$\text{Edof} = \left[ \begin{array}{cccccc} el_1 & dof_1 & dof_2 & . & . & dof_{ned} \\ el_2 & dof_1 & dof_2 & . & . & dof_{ned} \\ \vdots & \vdots & \vdots & & & \vdots \\ el_{nel} & dof_1 & dof_2 & . & . & dof_{ned} \end{array} \right] \left. \vphantom{\begin{array}{cccccc} el_1 & dof_1 & dof_2 & . & . & dof_{ned} \\ el_2 & dof_1 & dof_2 & . & . & dof_{ned} \\ \vdots & \vdots & \vdots & & & \vdots \\ el_{nel} & dof_1 & dof_2 & . & . & dof_{ned} \end{array}} \right\} \text{one row for each element}$$

If **fe** and **f** are given in the function, the element load vector  $\mathbf{f}^e$  is also added to the global load vector **f**.



**coordxtr****Static system functions**

---

The output variables **Ex**, **Ey**, and **Ez** are matrices defined according to

$$\mathbf{Ex} = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & \dots & x_{nen}^1 \\ x_1^2 & x_2^2 & x_3^2 & \dots & x_{nen}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{nel} & x_2^{nel} & x_3^{nel} & \dots & x_{nen}^{nel} \end{bmatrix}$$

where row  $i$  gives the  $x$ -coordinates of the element defined in row  $i$  of **Edof**, and where  $nel$  is the number of considered elements.

The element coordinate data extracted by the function **coordxtr** can be used for plotting purposes and to create input data for the element stiffness functions.

**Static system functions****eigen**

---

**Purpose:**

Solve the generalized eigenvalue problem.

**Syntax:**

```
L=eigen(K,M)
L=eigen(K,M,b)
[L,X]=eigen(K,M)
[L,X]=eigen(K,M,b)
```

**Description:**

`eigen` solves the eigenvalue problem

$$|K - \lambda M| = 0$$

where  $K$  and  $M$  are square matrices. The eigenvalues  $\lambda$  are stored in the vector  $L$  and the corresponding eigenvectors in the matrix  $X$ .

If certain rows and columns in matrices  $K$  and  $M$  are to be eliminated in computing the eigenvalues,  $b$  must be given in the function. The rows (and columns) that are to be eliminated are described in the vector  $b$  defined as

$$b = \begin{bmatrix} dof_1 \\ dof_2 \\ \vdots \\ dof_{nb} \end{bmatrix}$$

The computed eigenvalues are given in order ranging from the smallest to the largest. The eigenvectors are normalized in order that

$$X^T M X = I$$

where  $I$  is the identity matrix.

**extract**

**Static system functions**

**Purpose:**

Extract element nodal quantities from a global solution vector.

$$\begin{bmatrix} \vdots \\ a_i \\ a_j \\ \vdots \\ a_m \\ a_n \\ \vdots \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} a_i \\ a_j \\ a_m \\ a_n \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \quad \begin{array}{l} \text{edof} = [eln \quad i \quad j \quad m \quad n] \\ \text{ed} = [u_1 \quad u_2 \quad u_3 \quad u_4] \end{array}$$

**Syntax:**

`ed=extract(edof,a)`

**Description:**

**extract** extracts element displacements or corresponding quantities  $\mathbf{a}^e$  from the global solution vector  $\mathbf{a}$ , stored in  $\mathbf{a}$ .

Input variables are the element topology matrix **edof**, defined in **assem**, and the global solution vector  $\mathbf{a}$ .

The output variable

$$\text{ed} = (\mathbf{a}^e)^T$$

contains the element displacement vector.

If **Edof** contains more than one element, **Ed** will be a matrix

$$\text{Ed} = \begin{bmatrix} (\mathbf{a}^e)_1^T \\ (\mathbf{a}^e)_2^T \\ \vdots \\ (\mathbf{a}^e)_{nel}^T \end{bmatrix}$$

where row  $i$  gives the element displacements for the element defined in row  $i$  of **Edof**, and  $nel$  is the total number of considered elements.

## Static system functions

## extract

### Example:

For the two dimensional beam element, the **extract** function will extract six nodal displacements for each element given in **Edof**, and create a matrix **Ed** of size  $(nel \times 6)$ .

$$\mathbf{Ed} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \\ u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{bmatrix}$$

**insert****Static system functions**

---

**Purpose:**

Assemble internal element forces in a global force vector.

$$\begin{array}{c}
 \begin{bmatrix} f_i^e \\ f_j^e \end{bmatrix} \\
 \mathbf{f}^e \\
 i = dof_i \\
 j = dof_j
 \end{array}
 \xrightarrow{\quad}
 \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_i + f_i^e \\ f_j + f_j^e \\ \vdots \\ f_n \end{bmatrix} \\
 \mathbf{f}$$

**Syntax:**

`f=insert(edof,f,ef)`

**Description:**

`insert` adds the internal element load vector  $\mathbf{f}_i^e$ , stored in `ef`, to the global internal force vector  $\mathbf{f}$ , stored in `f`, according to the topology matrix `edof`. The function is for use in nonlinear analysis.

The element topology matrix `edof` is defined in `assem`. The vector `f` is the global internal force vector, and the vector `ef` is the internal element force vector computed from the element stresses, see for example `plani4f`.

**Static system functions****solveq**

---

**Purpose:**

Solve equation system.

**Syntax:**

```
a=solveq(K,f)
a=solveq(K,f,bc)
[a,r]=solveq(K,f,bc)
```

**Description:**

solveq solves the equation system

$$\mathbf{K} \mathbf{a} = \mathbf{f}$$

where  $\mathbf{K}$  is a matrix and  $\mathbf{a}$  and  $\mathbf{f}$  are vectors.

The matrix  $\mathbf{K}$  and the vector  $\mathbf{f}$  must be predefined. The solution of the system of equations is stored in a vector  $\mathbf{a}$  which is created by the function.

If some values of  $\mathbf{a}$  are to be prescribed, the row number and the corresponding values are given in the boundary condition matrix

$$\mathbf{bc} = \begin{bmatrix} dof_1 & u_1 \\ dof_2 & u_2 \\ \vdots & \vdots \\ dof_{nbc} & u_{nbc} \end{bmatrix}$$

where the first column contains the row numbers and the second column the corresponding prescribed values.

If  $\mathbf{r}$  is given in the function, support forces are computed according to

$$\mathbf{r} = \mathbf{K} \mathbf{a} - \mathbf{f}$$



**statcon****Static system functions**

---

**Purpose:**

Reduce system of equations by static condensation.

**Syntax:**

$[K1, f1] = \text{statcon}(K, f, b)$

**Description:**

**statcon** reduces a system of equations

$$K \mathbf{a} = \mathbf{f}$$

by static condensation.

The degrees of freedom to be eliminated are supplied to the function by the vector

$$\mathbf{b} = \begin{bmatrix} dof_1 \\ dof_2 \\ \vdots \\ dof_{nb} \end{bmatrix}$$

where each row in **b** contains one degree of freedom to be eliminated.

The elimination gives the reduced system of equations

$$K_1 \mathbf{a}_1 = \mathbf{f}_1$$

where  $K_1$  and  $\mathbf{f}_1$  are stored in **K1** and **f1** respectively.

### 6.3 Dynamic system functions

The group of system functions comprises functions for solving linear dynamic systems by time stepping or modal analysis, functions for frequency domain analysis, etc.

Dynamic system functions	
dyna2	Solve a set of uncoupled second-order differential equations
dyna2f	Solve a set of uncoupled second-order differential equations in the frequency domain
fft	Fast Fourier transform
freqresp	Compute frequency response
gfunc	Linear interpolation between equally spaced points
ifft	Inverse Fast Fourier transform
ritz	Compute approximative eigenvalues and eigenvectors by the Lanczos method
spectra	Compute seismic response spectra
step1	Carry out step-by-step integration in first-order systems
step2	Carry out step-by-step integration in second-order systems
sweep	Compute frequency response function

**Note:** Eigenvalue analysis is performed by using the function `eigen`; see static system functions.

**dyna2****Dynamic system functions**

---

**Purpose:**

Compute the dynamic solution to a set of uncoupled second-order differential equations.

**Syntax:**

`X=dyna2(w2,xi,f,g,dt)`

**Description:**

`dyna2` computes the solution to the set

$$\ddot{x}_i + 2\xi_i\omega_i\dot{x}_i + \omega_i^2x_i = f_i g(t), \quad i = 1, \dots, m$$

of differential equations, where  $g(t)$  is a piecewise linear time function.

The set of vectors `w2`, `xi` and `f` contains the squared circular frequencies  $\omega_i^2$ , the damping ratios  $\xi_i$  and the applied forces  $f_i$ , respectively. The vector `g` defines the load function in terms of straight line segments between equally spaced points in time. This function may have been formed by the command `gfunc`.

The dynamic solution is computed at equal time increments defined by `dt`. Including the initial zero vector as the first column vector, the result is stored in the  $m$ -by- $n$  matrix `X`,  $n - 1$  being the number of time steps.

**Note:**

The accuracy of the solution is *not* a function of the output time increment `dt`, since the command produces the exact solution for straight line segments in the loading time function.

**See also:**

`gfunc`

**Dynamic system functions****dyna2f****Purpose:**

Compute the dynamic solution to a set of uncoupled second-order differential equations.

**Syntax:**

`Y=dyna2f(w2,xi,f,p,dt)`

**Description:**

`dyna2f` computes the solution to the set

$$\ddot{x}_i + 2\xi_i\omega_i\dot{x}_i + \omega_i^2x_i = f_i g(t), \quad i = 1, \dots, m$$

of differential equations in the frequency domain.

The vectors `w2`, `xi` and `f` are the squared circular frequencies  $\omega_i^2$ , the damping ratios  $\xi_i$  and the applied forces  $f_i$ , respectively. The force vector `p` contains the Fourier coefficients  $p(k)$  formed by the command `fft`.

The solution in the frequency domain is computed at equal time increments defined by `dt`. The result is stored in the  $m$ -by- $n$  matrix `Y`, where  $m$  is the number of equations and  $n$  is the number of frequencies resulting from the `fft` command. The dynamic solution in the time domain is achieved by the use of the command `ifft`.

**Example:**

The dynamic solution to a set of uncoupled second-order differential equations can be computed by the following sequence of commands:

```
>> g=gfunc(G,dt);  
>> p=fft(g);  
>> Y=dyna2f(w2,xi,f,p,dt);  
>> X=(real(ifft(Y.')))';
```

where it is assumed that the input variables `G`, `dt`, `w2`, `xi` and `f` are properly defined. Note that the `ifft` command operates on column vectors if `Y` is a matrix; therefore use the transpose of `Y`. The output from the `ifft` command is complex. Therefore use `Y.'` to transpose rows and columns in `Y` in order to avoid the complex conjugate transpose of `Y`; see Section 3. The time response is represented by the real part of the output from the `ifft` command. If the transpose is used and the result is stored in a matrix `X`, each row will represent the time response for each equation as the output from the command `dyna2`.

**See also:**

`gfunc`, `fft`, `ifft`

**fft****Dynamic system functions**

---

**Purpose:**

Transform functions in time domain to frequency domain.

**Syntax:**

$p = \text{fft}(g)$   
 $p = \text{fft}(g, N)$

**Description:**

`fft` transforms a time dependent function to the frequency domain.

The function to be transformed is stored in the vector  $\mathbf{g}$ . Each row in  $\mathbf{g}$  contains the value of the function at equal time intervals. The function represents a span  $-\infty \leq t \leq +\infty$ ; however, only the values within a typical period are specified by  $\mathbf{g}$ .

The `fft` command can be used with one or two input arguments. If  $N$  is not specified, the numbers of frequencies used in the transformation is equal to the the numbers of points in the time domain, i.e. the length of the variable  $\mathbf{g}$ , and the output will be a vector of the same size containing complex values representing the frequency content of the input signal.

The scalar variable  $N$  can be used to specify the numbers of frequencies used in the Fourier transform. The size of the output vector in this case will be equal to  $N$ . It should be remembered that the highest harmonic component in the time signal that can be identified by the Fourier transform corresponds to half the sampling frequency. The sampling frequency is equal to  $1/dt$ , where  $dt$  is the time increment of the time signal.

The complex Fourier coefficients  $p(k)$  are stored in the vector  $\mathbf{p}$ , and are computed according to

$$p(k) = \sum_{j=1}^N x(j) \omega_N^{(j-1)(k-1)},$$

where

$$\omega_N = e^{-2\pi i/N}.$$

**Note:**

This is a MATLAB built-in function.

**Dynamic system functions****freqresp**

---

**Purpose:**

Compute frequency response of a known discrete time response.

**Syntax:**

`[Freq,Resp] = freqresp(D,dt)`

**Description:**

`freqresp` computes the frequency response of a discrete dynamic system.

`D` is the time history function and `dt` is the sampling time increment, i.e. the time increment used in the time integration procedure.

`Resp` contains the computed response as a function of frequency. `Freq` contains the corresponding frequencies.

**Example:**

The result can be visualized by

```
>> plot(Freq,Resp)
>> xlabel('frequency (Hz)')
```

or

```
>> semilogy(Freq,Resp)
>> xlabel('frequency (Hz)')
```

The dimension of `Resp` is the same as that of the original time history function.

**Note:**

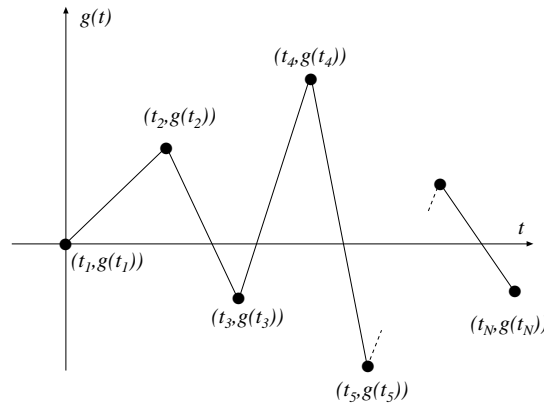
The time history function of a discrete system computed by direct integration behaves often in an unstructured manner. The reason for this is that the time history is a mixture of several participating eigenmodes at different eigenfrequencies. By using a Fourier transform, however, the response as a function of frequency can be computed efficiently. In particular it is possible to identify the participating frequencies.

**gfunc**
**Dynamic system functions**


---

**Purpose:**

Form vector with function values at equally spaced points by linear interpolation.


**Syntax:**

`[t,g]=gfunc(G,dt)`

**Description:**

**gfunc** uses linear interpolation to compute values at equally spaced points for a discrete function  $g$  given by

$$G = \begin{bmatrix} t_1 & g(t_1) \\ t_2 & g(t_2) \\ \vdots & \\ t_N & g(t_N) \end{bmatrix},$$

as shown in the figure above.

Function values are computed in the range  $t_1 \leq t \leq t_N$ , at equal increments,  $dt$  being defined by the variable **dt**. The number of linear segments (steps) is  $(t_N - t_1)/dt$ . The corresponding vector **t** is also computed. The result can be plotted by using the command `plot(t,g)`.

**Dynamic system functions****ifft**

---

**Purpose:**

Transform function in frequency domain to time domain.

**Syntax:**

`x=ifft(y)`  
`x=ifft(y,N)`

**Description:**

ifft transforms a function in the frequency domain to a function in the time domain.

The function to be transformed is given in the vector `y`. Each row in `y` contains Fourier terms in the interval  $-\infty \leq \omega \leq +\infty$ .

The `fft` command can be used with one or two input arguments. The scalar variable `N` can be used to specify the numbers of frequencies used in the Fourier transform. The size of the output vector in this case will be equal to `N`. See also the description of the command `fft`.

The inverse Fourier coefficients  $x(j)$ , stored in the variable `x`, are computed according to

$$x(j) = (1/N) \sum_{k=1}^N y(k) \omega_N^{-(j-1)(k-1)},$$

where

$$\omega_N = e^{-2\pi i/N}.$$

**Note:**

This is a MATLAB built-in function.

**See also:**

`fft`



**ritz****Dynamic system functions**

---

**Purpose:**

Compute approximative eigenvalues and eigenvectors by the Lanczos method.

**Syntax:**

```
L=ritz(K,M,f,m)
L=ritz(K,M,f,m,b)
[L,X]=ritz(K,M,f,m)
[L,X]=ritz(K,M,f,m,b)
```

**Description:**

`ritz` computes, by the use of the Lanczos algorithm, `m` approximative eigenvalues and `m` corresponding eigenvectors for a given pair of  $n$ -by- $n$  matrices `K` and `M` and a given non-zero starting vector `f`.

If certain rows and columns in matrices `K` and `M` are to be eliminated in computing the eigenvalues, `b` must be given in the command. The rows (and columns) to be eliminated are described in the vector `b` defined as

$$\mathbf{b} = \begin{bmatrix} dof_1 \\ dof_2 \\ \vdots \\ dof_{nb} \end{bmatrix}.$$

**Note:**

If the number of vectors, `m`, is chosen less than the total number of degrees-of-freedom,  $n$ , only about the first  $m/2$  Ritz vectors are good approximations of the true eigenvectors. Recall that the Ritz vectors satisfy the `M`-orthonormality condition

$$\mathbf{X}^T \mathbf{M} \mathbf{X} = \mathbf{I},$$

where `I` is the identity matrix.

**Dynamic system functions****spectra**

---

**Purpose:**

Compute seismic response spectra for elastic design.

**Syntax:**

`s=spectra(a,xi,dt,f)`

**Description:**

`spectra` computes the seismic response spectrum for a known acceleration history function.

The computation is based on the vector `a`, that contains an acceleration time history function defined at equal time steps. The time step is specified by the variable `dt`. The value of the damping ratio is given by the variable `xi`.

Output from the computation, stored in the vector `s`, is achieved at frequencies specified by the column vector `f`.

**Example:**

The following procedure can be used to produce a seismic response spectrum for a damping ratio  $\xi = 0.05$ , defined at 34 logarithmically spaced frequency points. The acceleration time history `a` has been sampled at a frequency of 50 Hz, corresponding to a time increment  $dt = 0.02$  between collected points:

```
>> freq=logspace(0,log10(2^(33/6)),34);  
>> xi=0.05;  
>> dt=0.02;  
>> s=spectra(a,xi,dt,freq');
```

The resulting spectrum can be plotted by the command

```
>> loglog(freq,s,'*')
```

## step1

## Dynamic system functions

### Purpose:

Compute the dynamic solution to a set of first order differential equations.

### Syntax:

```
Tsnap=step1(K,C,d0,ip,f,pbound)
[Tsnap,D,V]=step1(K,C,d0,ip,f,pbound)
```

### Description:

**step1** computes at equal time steps the solution to a set of first order differential equations of the form

$$\mathbf{C}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}(x, t),$$

$$\mathbf{d}(0) = \mathbf{d}_0.$$

The command solves transient field problems. In the case of heat conduction,  $\mathbf{K}$  and  $\mathbf{C}$  represent the  $n \times n$  conductivity and capacity matrices, respectively.

The initial conditions are given by the vector  $\mathbf{d}_0$  containing initial values of  $\mathbf{d}$ . The time integration procedure is governed by the parameters given in the vector  $\mathbf{ip}$  defined as

$$\mathbf{ip} = [dt \ T \ \alpha \ \underbrace{[nsnap \ nhist \ time_i \ \dots \ dof_i \ \dots]}_{\substack{\text{list of} \\ nsnap \\ \text{moments}}} \ \underbrace{]}_{\substack{\text{list of} \\ nhist \\ \text{dofs}}},$$

where  $dt$  specifies the time increment in the time stepping scheme,  $T$  total time and  $\alpha$  a time integration constant; see [1]. The parameter  $nsnap$  denotes the number of snapshots stored in **Tsnap**. The selected elapsed times are specified in  $(time_i \ \dots)$ , whereas  $nhist$  is the number of time histories stored in **D** and **V**. The selected degrees-of-freedom are specified in  $(dof_i \ \dots)$ . The following table lists frequently used values of  $\alpha$ :

- $\alpha = 0$  Forward difference; forward Euler,
- $\alpha = \frac{1}{2}$  Trapezoidal rule; Crank-Nicholson,
- $\alpha = 1$  Backward difference; backward Euler.

The matrix  $\mathbf{f}$  contains the time-dependent load vectors. If no external loads are active, the matrix corresponding to  $\mathbf{f}$  should be replaced by  $\mathbf{[]}$ . The matrix  $\mathbf{f}$  contains the time-dependent prescribed values of the field variable. If no field variables are prescribed the matrix corresponding to **pbound** should be replaced by  $\mathbf{[]}$ . Matrix  $\mathbf{f}$  is organized in the following manner:

$$\mathbf{f} = \begin{bmatrix} \text{time history of the load at } dof_1 \\ \text{time history of the load at } dof_2 \\ \vdots \\ \text{time history of the load at } dof_n \end{bmatrix}.$$

The dimension of  $\mathbf{f}$  is

$$(\text{number of degrees-of-freedom}) \times (\text{number of timesteps} + 1).$$

The matrix **pbound** is organized in the following manner:

$$\mathbf{pbound} = \begin{bmatrix} dof_1 & \text{time history of the field variable} \\ dof_2 & \text{time history of the field variable} \\ \vdots & \vdots \\ dof_{m_2} & \text{time history of the field variable} \end{bmatrix}.$$

The dimension of **pbound** is

$$(\text{number of dofs with prescribed field values}) \times (\text{number of timesteps} + 2).$$

The time history functions can be generated using the command **gfunc**. If all the values of the time histories of  $\mathbf{f}$  or **pbound** are kept constant, these values need to be stated only once. In this case the number of columns in  $\mathbf{f}$  is one and in **pbound** two.

It is highly recommended to define  $\mathbf{f}$  as **sparse** (a MATLAB built-in function). In most cases only a few degrees-of-freedom are affected by the exterior load, and hence the matrix contains only few non-zero entries.

The computed snapshots are stored in **Tsnap**, one column for each requested snapshot according to **ip**, i.e. the dimension of **Tsnap** is  $(\text{number of dofs}) \times nsnap$ . The computed time histories of  $\mathbf{d}$  and  $\dot{\mathbf{d}}$  are stored in **D** and **V**, respectively, one line for each requested degree-of-freedom according to **ip**. The dimension of **D** is  $nhist \times (\text{number of timesteps} + 1)$ .

## step2

## Dynamic system functions

### Purpose:

Compute the dynamic solution to a set of second order differential equations.

### Syntax:

```
Dsnap=step2(K,C,M,d0,v0,ip,f,pdisp)
[Dsnap,D,V,A]=step2(K,C,M,d0,v0,ip,f,pdisp)
```

### Description:

**step2** computes at equal time steps the solution to a second order differential equations of the form

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{d}} + \mathbf{C}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} &= \mathbf{f}(x, t), \\ \mathbf{d}(0) &= \mathbf{d}_0, \\ \dot{\mathbf{d}}(0) &= \mathbf{v}_0. \end{aligned}$$

In structural mechanics problems,  $\mathbf{K}$ ,  $\mathbf{C}$  and  $\mathbf{M}$  represent the  $n \times n$  stiffness, damping and mass matrices, respectively.

The initial conditions are given by the vectors  $\mathbf{d}_0$  and  $\mathbf{v}_0$ , containing initial displacements and initial velocities. The time integration procedure is governed by the parameters given in the vector  $\mathbf{ip}$  defined as

$$\mathbf{ip} = [dt \ T \ \alpha \ \delta \ \underbrace{[nsnap \ nhist]}_{\substack{\text{list of} \\ nsnap \\ \text{moments}}} \ \underbrace{[time_i \ \dots \ dof_i \ \dots]}_{\substack{\text{list of} \\ nhist \\ \text{dofs}}}],$$

where  $dt$  specifies the time increment in the time stepping scheme,  $T$  the total time and  $\alpha$  and  $\delta$  time integration constants for the Newmark family of methods; see [1]. The parameter  $nsnap$  denotes the number of snapshots stored in **Dsnap**. The selected elapsed times are specified in  $(time_i \ \dots)$ , whereas  $nhist$  is the number of time histories stored in **D**, **V** and **A**. The selected degrees-of-freedom are specified in  $(dof_i \ \dots)$ . The following table lists frequently used values of  $\alpha$  and  $\delta$ :

$$\begin{aligned} \alpha = \frac{1}{4} \quad \delta = \frac{1}{2} & \quad \text{Average acceleration (trapezoidal) rule,} \\ \alpha = \frac{1}{6} \quad \delta = \frac{1}{2} & \quad \text{Linear acceleration,} \\ \alpha = 0 \quad \delta = \frac{1}{2} & \quad \text{Central difference.} \end{aligned}$$

The matrix **f** contains the time-dependent load vectors. If no external loads are active, the matrix corresponding to **f** should be replaced by **[]**. The matrix **pdisp** contains the time-dependent prescribed displacement. If no displacements are prescribed the matrix corresponding to **pdisp** should be replaced by **[]**.

**Dynamic system functions****step2**

The matrix **f** is organized in the following manner:

$$\mathbf{f} = \begin{bmatrix} \text{time history of the load at } dof_1 \\ \text{time history of the load at } dof_2 \\ \vdots \\ \text{time history of the load at } dof_n \end{bmatrix}.$$

The dimension of **f** is

$$(\text{number of degrees-of-freedom}) \times (\text{number of timesteps} + 1).$$

The matrix **pdisp** is organized in the following manner

$$\mathbf{pdisp} = \begin{bmatrix} dof_1 & \text{time history of the displacement} \\ dof_2 & \text{time history of the displacement} \\ \vdots & \vdots \\ dof_{m_2} & \text{time history of the displacement} \end{bmatrix}.$$

The dimension of **pdisp** is

$$(\text{number of dofs with prescribed displacement}) \times (\text{number of timesteps} + 2).$$

The time history functions can be generated using the command **gfunc**. If all the values of the time histories of **f** or **pdisp** are kept constant, these values need to be stated only once. In this case the number of columns in **f** is one and in **pdisp** two.

It is highly recommended to define **f** as **sparse** (a MATLAB built-in function). In most cases only a few degrees-of-freedom are affected by the exterior load, and hence the matrix contains only few non-zero entries.

The computed displacement snapshots are stored in **Dsnap**, one column for each requested snapshot according to **ip**, i.e. the dimension of **Dsnap** is  $(\text{number of dofs}) \times n_{snap}$ . The computed time histories of **d**, **ḍ** and **ḍ̈** are stored in **D**, **V** and **A**, respectively, one line for each requested degree-of-freedom according to **ip**. The dimension of **D** is  $n_{hist} \times (\text{number of timesteps} + 1)$ .

**sweep****Dynamic system functions**

---

**Purpose:**

Compute complex frequency response functions.

**Syntax:**

`Y=sweep(K,C,M,p,w)`

**Description:**

`sweep` computes the complex frequency response function for a system of the form

$$[\mathbf{K} + i\omega\mathbf{C} - \omega^2\mathbf{M}]\mathbf{y}(\omega) = \mathbf{p}.$$

Here  $\mathbf{K}$ ,  $\mathbf{C}$  and  $\mathbf{M}$  represent the  $m$ -by- $m$  stiffness, damping and mass matrices, respectively. The vector  $\mathbf{p}$  defines the amplitude of the force. The frequency response function is computed for the values of  $\omega$  given by the vector  $\mathbf{w}$ .

The complex frequency response function is stored in the matrix  $\mathbf{Y}$  with dimension  $m$ -by- $n$ , where  $n$  is equal to the number of circular frequencies defined in  $\mathbf{w}$ .

**Example:**

The steady-state response can be computed by

```
>> X=real(Y*exp(i*w*t));
```

and the amplitude by

```
>> Z=abs(Y);
```

## 7 Statements and macros

Statements describe algorithmic actions that can be executed. There are two different types of control statements, conditional and repetitive. The first group defines conditional jumps whereas the latter one defines repetition until a conditional statement is fulfilled. Macros are used to define new functions to the MATLAB or CALFEM structure, or to store a sequence of statements in an `.m`-file.

Control statements	
<code>if</code>	Conditional jump
<code>for</code>	Initiate a loop
<code>while</code>	Define a conditional loop

Macros	
<code>function</code>	Define a new function
<code>script</code>	Store a sequence of statements



## if

---

### Purpose:

Conditional jump.

### Syntax:

```
if logical expression
:
elseif logical expression
:
else
:
end
```

### Description:

if initiates a conditional jump. If *logical expression* produces the value *True* the statements following if are executed. If *logical expression* produces the value *False* the next conditional statement **elseif** is checked.

**elseif** works like if. One or more of the conditional statement **elseif** can be added after the initial conditional statement if.

If **else** is present, the statements following **else** are executed if the *logical expressions* in all if and **elseif** statements produce the value *False*. The if loop is closed by **end** to define the loop sequence.

The following relation operators can be used

==	equal
>=	greater than or equal to
>	greater than
<=	less than or equal to
<	less than
~=	not equal

### Note:

This is MATLAB built-in language.

**for**

---

**Purpose:**

Initiate a loop.

**Syntax:**

```
for i = start : inc : stop
:
end
```

**Description:**

**for** initiates a loop which terminates when **i>stop**. The **for** loop is closed by **end** to define the loop sequence.

**Examples:**

```
for i = 1 : 10           i takes values from 1 to 10.
for i = 1 : 2 : 10       i equals 1, 3, 5, 7, 9.
for i = 20 : -1 : 1      i equals 20, 19 ... 2, 1.
```

**Note:**

This is MATLAB built-in language.

## **while**

---

### **Purpose:**

Define a conditional loop.

### **Syntax:**

```
while logical expression
:
end
```

### **Description:**

**while** initiates a conditional loop which terminates when *logical expression* equals *False*. The **while** loop is closed by **end** to define the loop sequence.

The different relation operators that can be used can be found under the if command.

### **Examples:**

A loop continuing until **a** equals **b**

```
while a~=b
:
end
```

### **Note:**

This is MATLAB built-in language.

**Purpose:**

Define a new function.

**Syntax:**

`function[ out1 , out2 , ... ]=name( in1 , in2 , ... )`

**Description:**

*name* is replaced by the name of the function. The input variables `in1`, `in2`, ... can be scalars, vectors or matrices, and the same holds for the output variables `out1`, `out2`, ... .

**Example:**

To define the CALFEM function `spring1e` a file named `spring1e.m` is created. The file contains the following statements:

```
function [Ke]=spring1e(k)
% Define the stiffness matrix
% for a one dimensional spring
% with spring stiffness k
Ke=[ k, -k; -k, k ]
```

i.e. the function `spring1e` is defined to return a stiffness matrix.

**Note:**

This is MATLAB built-in language.

*script*

---

**Purpose:**

Execute a stored sequence of statements.

**Syntax:**

*name*

**Description:**

*name* is replaced by the name of the script.

**Example:**

The statements below are stored in a file named **spring.m** and executed by typing **spring** in the MATLAB command window.

```
% Stiffness matrix for a one dimensional  
% spring with stiffness k=10  
k=10;  
[Ke]=spring1e(k);
```

**Note:**

This is MATLAB built-in language.

## 8 Graphics functions

The group of graphics functions comprises functions for element based graphics. Mesh plots, displacements, section forces, flows, iso lines and principal stresses can be displayed. The functions are divided into two dimensional, and general graphics functions.

Two dimensional graphics functions	
plot	Plot lines and points in 2D space
fill	Draw filled 2D polygons
eldraw2	Draw undeformed finite element mesh
eldisp2	Draw deformed finite element mesh
eldia2	Draw section force diagram
elflux2	Plot flux vectors
eliso2	Draw isolines for nodal quantities
elprinc2	Plot principal stresses
scalfact2	Determine scale factor
pltscalb2	Draw scale bar

General graphics functions	
axis	Axis scaling and appearance
clf	Clear current figure
figure	Create figures
grid	Grid lines
hold	Hold current graph
print	Print graph or save graph to file
text	Add text to current plot
title	Titles for 2D and 3D plots
xlabel, ylabel, zlabel	Axis labels for 2D and 3D plots

## axis

---

### Purpose:

Plot axis scaling and appearance.

### Syntax:

```
axis([xmin xmax ymin ymax])  
axis([xmin xmax ymin ymax zmin zmax])  
axis auto  
axis square  
axis equal  
axis off  
axis on
```

### Description:

`axis([xmin xmax ymin ymax])` sets scaling for the x- and y-axes on the current 2D plot.

`axis([xmin xmax ymin ymax zmin zmax])` sets the scaling for the x-, y- and z-axes on the current 3D plot.

`axis auto` returns the axis scaling to its default automatic mode where, for each plot,  $xmin = \min(x)$ ,  $xmax = \max(x)$ , etc.

`axis square` makes the current axis box square in shape.

`axis equal` changes the current axis box size so that equal tick mark increments on the x- and y-axes are equal in size. This makes `plot(sin(x),cos(x))` look like a circle, instead of an oval.

`axis normal` restores the current axis box to full size and removes any restrictions on the scaling of the units. This undoes the effects of `axis square` and `axis equal`.

`axis off` turns off all axis labeling and tick marks.

`axis on` turns axis labeling and tick marks back on.

### Note:

This is a MATLAB built-in function. For more information about the `axis` function, type `help axis`.

**Purpose:**

Clear current figure (graph window).

**Syntax:**

clf

**Description:**

clf deletes all objects (axes) from the current figure.

**Note:**

This is a MATLAB built-in function. For more information about the clf function, type `help clf`.

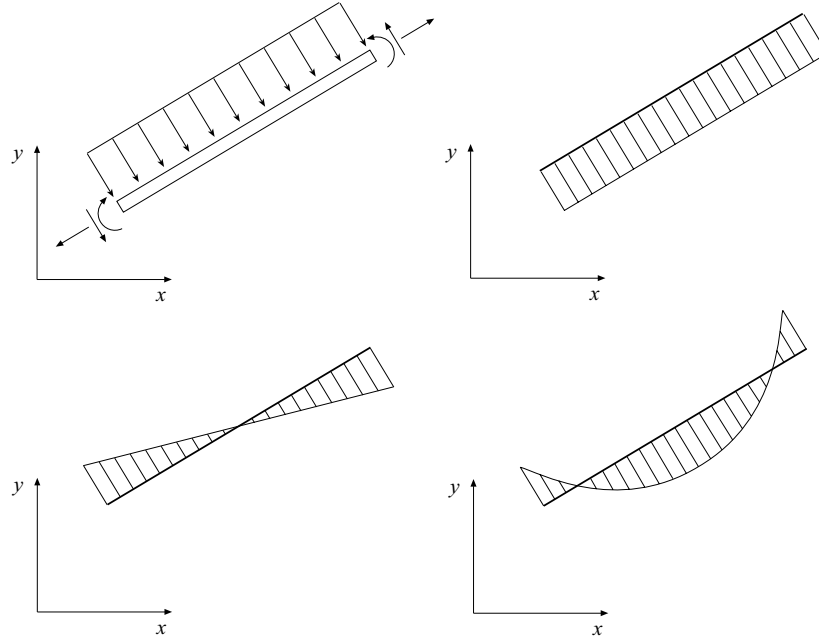


## eldia2

---

### Purpose:

Draw the section force diagrams of a two dimensional beam element.



### Syntax:

```
eldia2(ex,ey,es,plotpar,sfac)
eldia2(ex,ey,es,plotpar,sfac,eci)
[sfac]=eldia2(ex,ey,es)
[sfac]=eldia2(ex,ey,es,plotpar)
```

### Description:

eldia2 plots a section force diagram of a two dimensional beam element in its global position.

The input variables **ex** and **ey** are defined in **beam2e** and the input variable

$$\mathbf{es} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix}$$

consists of a column matrix that contains section forces. The values in **es** are computed in **beam2s**. It should be noted, however, that whereas all three section forces are computed in **beam2s** only one of them shall be given as input to **eldia2** by **es**.

The variable **plotpar** sets plot parameters for the diagram.

**plotpar**=[ *linecolor* *elementcolor* ]

<i>linecolor</i> = 1	black	<i>elementcolor</i> = 1	black
2	blue	2	blue
3	magenta	3	magenta
4	red	4	red

The scale factor **sfac** is a scalar that the section forces are multiplied with to get a suitable graphical representation. **sfac** is set automatically if it is omitted in the input list.

The input variable

$$\mathbf{eci} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_n \end{bmatrix}$$

specifies the local  $\bar{x}$ -coordinates of the quantities in **es**. If **eci** is not given, uniform distance is assumed.

**Limitations:**

Supported elements are two dimensional beam elements.

**eldisp2**

---

**Purpose:**

Draw the deformed mesh for a two dimensional structure.

**Syntax:**

```
[sfac]=eldisp2(Ex,Ey,Ed)
[sfac]=eldisp2(Ex,Ey,Ed,plotpar)
eldisp2(Ex,Ey,Ed,plotpar,sfac)
```

**Description:**

eldisp2 displays the deformed mesh for a two dimensional structure.

Input variables are the coordinate matrices **Ex** and **Ey** formed by the function **co-ordxtr**, and the element displacements **Ed** formed by the function **extract**.

The variable **plotpar** sets plot parameters for linetype, linecolor and node marker.

```
plotpar=[ linetype linecolor nodemark ]
```

<i>linetype</i> = 1	solid line	<i>linecolor</i> = 1	black
2	dashed line	2	blue
3	dotted line	3	magenta
		4	red

<i>nodemark</i> = 1	circle
2	star
0	no mark

Default is dashed black lines with circles at nodes.

The scale factor **sfac** is a scalar that the element displacements are multiplied with to get a suitable geometrical representation. The scale factor is set automatically if it is omitted in the input list.

**Limitations:**

Supported elements are bar, beam, triangular three node, and quadrilateral four node elements.

**Purpose:**

Draw the undeformed mesh for a two dimensional structure.

**Syntax:**

```
eldraw2(Ex,Ey)
eldraw2(Ex,Ey,plotpar)
eldraw2(Ex,Ey,plotpar,elnum)
```

**Description:**

eldraw2 displays the undeformed mesh for a two dimensional structure.

Input variables are the coordinate matrices **Ex** and **Ey** formed by the function **cordxtr**.

The variable **plotpar** sets plot parameters for linetype, linecolor and node marker.

**plotpar** = [ *linetype* *linecolor* *nodemark* ]

<i>linetype</i> = 1	solid line	<i>linecolor</i> = 1	black
2	dashed line	2	blue
3	dotted line	3	magenta
		4	red

<i>nodemark</i> = 1	circle
2	star
0	no mark

Default is solid black lines with circles at nodes.

Element numbers can be displayed at the center of the element if a column vector **elnum** with the element numbers is supplied. This column vector can be derived from the element topology matrix **Edof**,

**elnum**=**Edof**(:,1)

i.e. the first column of the topology matrix.

**Limitations:**

Supported elements are bar, beam, triangular three node, and quadrilateral four node elements.

**elflux2**

---

**Purpose:**

Draw element flow arrows for two dimensional elements.

**Syntax:**

```
[sfac]=elflux2(Ex,Ey,Es)
[sfac]=elflux2(Ex,Ey,Es,plotpar)
elflux2(Ex,Ey,Es,plotpar,sfac)
```

**Description:**

elflux2 displays element heat flux vectors (or corresponding quantities) for a number of elements of the same type. The flux vectors are displayed as arrows at the element centroids. Note that only the flux vectors are displayed. To display the element mesh, use **eldraw2**.

Input variables are the coordinate matrices **Ex** and **Ey**, and the element flux matrix **Es** defined in **flw2ts** or **flw2qs**.

The variable **plotpar** sets plot parameters for the flux arrows.

```
plotpar=[ arrowtype arrowcolor ]
```

<i>arrowtype</i> = 1	solid	<i>arrowcolor</i> = 1	black
2	dashed	2	blue
3	dotted	3	magenta
		4	red

Default, if **plotpar** is omitted, is solid black arrows.

The scale factor **sfac** is a scalar that the values are multiplied with to get a suitable arrow size in relation to the element size. The scale factor is set automatically if it is omitted in the input list.

**Limitations:**

Supported elements are triangular 3 node and quadrilateral 4 node elements.

**Purpose:**

Display element iso lines for two dimensional elements.

**Syntax:**

```
eliso2(Ex,Ey,Ed,isov)
eliso2(Ex,Ey,Ed,isov,plotpar)
```

**Description:**

`eliso2` displays element iso lines for a number of elements of the same type. Note that only the iso lines are displayed. To display the element mesh, use `eldraw2`.

Input variables are the coordinate matrices `Ex` and `Ey` formed by the function `co-ordxtr`, and the element nodal quantities (e.g displacement or energy potential) matrix `Ed` defined in `extract`.

If `isov` is a scalar it determines the number of iso lines to be displayed. If `isov` is a vector it determines the values of the iso lines to be displayed (number of iso lines equal to length of vector `isov`).

```
isov = [isolines]
isov = [isoval(1) ... isoval(n)]
```

The variable `plotpar` sets plot parameters for the iso lines.

```
plotpar=[ linetype linecolor textfcn ]
```

<i>arrowtype</i> = 1	solid	<i>arrowcolor</i> = 1	black
2	dashed	2	blue
3	dotted	3	magenta
		4	red

<i>textfcn</i> = 0	the iso values of the lines will not be printed
1	the iso values of the lines will be printed at the iso lines
2	the iso values of the lines will be printed where the cursor indicates

Default is solid, black lines and no iso values printed.

**Limitations:**

Supported elements are triangular 3 node and quadrilateral 4 node elements.

## elprinc2

---

### Purpose:

Draw element principal stresses as arrows for two dimensional elements.

### Syntax:

```
[sfac]=elprinc2(Ex,Ey,Es)
[sfac]=elprinc2(Ex,Ey,Es,plotpar)
elprinc2(Ex,Ey,Es,plotpar,sfac)
```

### Description:

elprinc2 displays element principal stresses for a number of elements of the same type. The principal stresses are displayed as arrows at the element centroids. Note that only the principal stresses are displayed. To display the element mesh, use `eldraw2`.

Input variables are the coordinate matrices `Ex` and `Ey`, and the element stresses matrix `Es` defined in `plants` or `planqs`

The variable `plotpar` sets plot parameters for the principal stress arrows.

```
plotpar=[ arrowtype arrowcolor ]
```

<i>arrowtype</i> = 1	solid	<i>arrowcolor</i> = 1	black
2	dashed	2	blue
3	dotted	3	magenta
		4	red

Default, if `plotpar` is omitted, is solid black arrows.

The scale factor `sfac` is a scalar that values are multiplied with to get a suitable arrow size in relation to the element size. The scale factor is set automatically if it is omitted in the input list.

### Limitations:

Supported elements are triangular 3 node and quadrilateral 4 node elements.

---

**figure**

---

**Purpose:**

Create figures (graph windows).

**Syntax:**

`figure(h)`

**Description:**

`figure(h)` makes the `h`'th figure the current figure for subsequent plotting functions. If *figure* `h` does not exist, a new one is created using the first available figure handle.

**Note:**

This is a MATLAB built-in function. For more information about the **figure** function, type `help figure`.



**fill**

---

**Purpose:**

Filled 2D polygons.

**Syntax:**

```
fill(x,y,c)
fill(X,Y,C)
```

**Description:**

`fill(x,y,c)` fills the 2D polygon defined by vectors `x` and `y` with the color specified by `c`. The vertices of the polygon are specified by pairs of components of `x` and `y`. If necessary, the polygon is closed by connecting the last vertex to the first.

If `c` is a vector of the same length as `x` and `y`, its elements are used to specify colors at the vertices. The color within the polygon is obtained by bilinear interpolation in the vertex colors.

If `X`, `Y` and `C` are matrices of the same size, `fill(X,Y,C)` draws one polygon per column with interpolated colors.

**Example:**

The solution of a heat conduction problem results in a vector `d` with nodal temperatures. The temperature distribution in a group of triangular 3 node (`nen=3`) or quadrilateral 4 node (`nen=4`) elements, with topology defined by `edof`, can be displayed by

```
[ex,ey]=cooridxtr(edof,Coord,Dof,nen)
ed=extract(edof,d)
colormap(hot)
fill(ex',ey',ed')
```

**Note:**

This is a MATLAB built-in function. For more information about the `fill` function, type `help fill`.

**Purpose:**

Grid lines for 2D and 3D plots.

**Syntax:**

grid on  
grid off  
grid

**Description:**

grid on adds grid lines on the current axes.

grid off takes them off.

grid by itself, toggles the grid state.

**Note:**

This is a MATLAB built-in function. For more information about the `grid` function, type `help grid`.

## **hold**

---

### **Purpose:**

Hold the current graph.

### **Syntax:**

hold on  
hold off  
hold

### **Description:**

hold on holds the current graph.

hold off returns to the default mode where plot functions erase previous plots.

hold by itself, toggles the hold state.

### **Note:**

This is a MATLAB built-in function. For more information about the **hold** function, type **help hold**.

---

**plot**

---

**Purpose:**

Linear two dimensional plot.

**Syntax:**

```
plot(x,y)
plot(x,y,'linetype')
```

**Description:**

`plot(x,y)` plots vector `x` versus vector `y`. Straight lines are drawn between each pair of values.

Various line types, plot symbols and colors may be obtained with `plot(x,y,s)` where `s` is a 1, 2, or 3 character string made from the following characters:

-	solid line	.	point	y	yellow
:	dotted line	o	circle	m	magenta
-.	dashdot line	x	x-mark	c	cyan
--	dashed line	+	plus	r	red
		*	star	g	green
				b	blue
				w	white
				k	black

Default is solid blue lines.

**Example:**

The statement

```
plot(x,y,'-',x,y,'ro')
```

plots the data twice, giving a solid blue line with red circles at the data points.

**Note:**

This is a MATLAB built-in function. For more information about the `plot` function, type `help plot`.

## pltscalb2

---

### Purpose:

Draw a scale bar.

### Syntax:

```
pltscalb2(sfac,magnitude)
pltscalb2(sfac,magnitude,plotpar)
```

### Description:

`pltscalb2` draws a scale bar to visualize the magnitude of displayed computational results. The input variable `sfact` is a scale factor determined by the function `scalfact2` and the variable

$$\text{magnitude} = [ S \ x \ y ]$$

specifies the value corresponding the length of the scale bar  $S$ , and  $(x,y)$  are the coordinates of the starting point. If no coordinates are given the starting point will be  $(0,-0.5)$ .

The variable `plotpar` sets the the scale bar colour.

`plotpar`=[*color* ]

<i>color</i> = 1	black
2	blue
3	magenta
4	red

---

**print**

---

**Purpose:**

Create hardcopy output of current figure window.

**Syntax:**

`print [filename]`

**Description:**

`print` with no arguments sends the contents of the current figure window to the default printer. `print filename` creates a PostScript file of the current figure window and writes it to the specified file.

**Note:**

This is a MATLAB built-in function. For more information about the `print` function, type `help print`.

## **scalfact2**

---

### **Purpose:**

Determine scale factor for drawing computational results.

### **Syntax:**

```
[sfac]=scalfact2(ex,ey,ed)  
[sfac]=scalfact2(ex,ey,ed,rat)
```

### **Description:**

**scalfact2** determines a scale factor **sfac** for drawing computational results, such as displacements, section forces or flux.

Input variables are the coordinate matrices **ex** and **ey** and the matrix **ed** containing the quantity to be displayed. The scalar **rat** defines the ratio between the geometric representation of the largest quantity to be displayed and the element size. If **rat** is not specified, 0.2 is used.

**Purpose:**

Add text to current plot.

**Syntax:**

`text(x,y,'string')`

**Description:**

`text` adds the text in the quotes to location  $(x,y)$  on the current axes, where  $(x,y)$  is in units from the current plot. If  $x$  and  $y$  are vectors, `text` writes the text at all locations given. If `'string'` is an array with the same number of rows as the length of  $x$  and  $y$ , `text` marks each point with the corresponding row of the `'string'` array.

**Note:**

This is a MATLAB built-in function. For more information about the `text` function, type `help text`.



## **title**

---

### **Purpose:**

Titles for 2D and 3D plots.

### **Syntax:**

```
title('text')
```

### **Description:**

`title` adds the text string `'text'` at the top of the current plot.

### **Note:**

This is a MATLAB built-in function. For more information about the `title` function, type `help title`.

---

**xlabel, ylabel, zlabel**

---

**Purpose:**

x-, y-, and z-axis labels for 2D and 3D plots.

**Syntax:**

```
xlabel('text')  
ylabel('text')  
zlabel('text')
```

**Description:**

xlabel adds text beside the x-axis on the current plot.

ylabel adds text beside the y-axis on the current plot.

zlabel adds text beside the z-axis on the current plot.

**Note:**

This is a MATLAB built-in function. For more information about the functions, type `help xlabel`, `help ylabel`, or `help zlabel`.



## 9 User's Manual, examples

### 9.1 Introduction

This set of examples is defined with the ambition to serve as a User's Manual. The examples, except the introductory ones, are written as `.m`-files (script files) and supplied together with the CALFEM functions.

The User's Manual examples are separated into four groups:

<b>MATLAB introduction</b>
<b>Static analysis</b>
<b>Dynamic analysis</b>
<b>Nonlinear analysis</b>

The MATLAB introduction examples explain some basic concepts and introduce a set of standard MATLAB functions usable in the finite element context. The static linear examples illustrate finite element analysis of different structures loaded by stationary loads. The dynamic linear examples illustrate some basic features in dynamics, such as modal analysis and time stepping procedures. The examples of nonlinear analysis cover subjects such as second order theory and buckling.



## 9.2 MATLAB introduction

The examples in this section illustrate basic MATLAB concepts such as handling of workspace, variables and functions. The examples are:

MATLAB introduction	
exi1	Handling matrices
exi2	Matrix and array operations
exi3	Create and handle .m-files
exi4	Display formats
exi5	Create a session .log-file
exi6	Graphic display of vectors

**Purpose:**

Show how to create and handle matrices in MATLAB.

**Description:**

The following commands create a scalar  $x$ , two vectors  $u$  and  $v$  and two matrices  $A$  and  $B$ .

Lines starting with the MATLAB prompt `>>` are command lines while the other lines show the results from these commands.

```
>> x=7
```

```
x =  
    7
```

```
>> u=[1 2 3 4]
```

```
u =  
    1     2     3     4
```

```
>> v=[0:0.4:2]
```

```
v =  
    0    0.4000    0.8000    1.2000    1.6000    2.0000
```

```
>> A=[1 2; 3 4]
```

```
A =  
    1     2  
    3     4
```

```
>> B=[5 6; 7 8]
```

```
B =  
    5     6  
    7     8
```

Both brief and detailed listing of variables is possible

```
>> who
```

Your variables are:

```
A          B          u          v          x
```

```
>> whos
```

Name	Size	Bytes	Class
A	2x2	32	double array
B	2x2	32	double array
u	1x4	32	double array
v	1x6	48	double array
x	1x1	8	double array

Grand total is 19 elements using 152 bytes

The value of a variable is displayed by writing the variable name,

```
>> u
```

```
u =  
    1    2    3    4
```

and the dimension ( $m \times n$ ) of a variable is obtained by

```
>> size(u)
```

```
ans =  
     1     4
```

where the answer is temporarily stored in the vector **ans**.

The variable **x** is removed from workspace by

```
>> clear x
```

To remove *all* variables from workspace, **clear** without argument is used.

Assignment of a value to an element in a matrix is made as

```
>> A(2,2)=9
```

```
A =  
    1    2  
    3    9
```



To select a complete row or column colon notation is used.

```
>> s=A(:,1)
```

```
s =  
    1  
    3
```

```
>> t=A(2,:)
```

```
t =  
    3     9
```

A zero matrix  $K$  ( $4 \times 4$ ) is generated by

```
>> K=zeros(4,4)
```

```
K =  
    0    0    0    0  
    0    0    0    0  
    0    0    0    0  
    0    0    0    0
```

Similarly an  $(m \times n)$  matrix of all ones can be generated by `ones(m,n)`.

Expand an already defined matrix

```
>> H=[A;B]
```

```
H =  
    1     2  
    3     9  
    5     6  
    7     8
```

```
>> J=[A B]
```

```
J =  
    1     2     5     6  
    3     9     7     8
```

**MATLAB introduction****exi2****Purpose:**

Show some examples of basic matrix and element-by-element operations.

**Description:**

Consider the following matrices

$$\mathbf{a} = \begin{bmatrix} 5 & 12 & 3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 & 0 & 4 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 6 & 3 \\ 2 & 8 & 4 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 2 & 5 & 4 \\ 7 & 2 & 0 \end{bmatrix}$$

The transpose of a matrix is generated by

```
>> A'
```

```
ans =
```

```
    1    2
    6    8
    3    4
```

```
>> a'
```

```
ans =
```

```
    5
   12
    3
```

Addition and subtraction of matrices

```
>> A+B
```

```
ans =
```

```
    3   11    7
    9   10    4
```

```
>> A-B
```

```
ans =
```

```
   -1    1   -1
   -5    6    4
```

Note that if the result of an operation is not assigned to a specific variable, the answer is temporarily stored in the variable **ans**.

Multiplication of matrices

```
>> a*b'
```

```
ans =  
    17
```

```
>> a'*b
```

```
ans =  
     5     0    20  
    12     0    48  
     3     0    12
```

```
>> A'*B
```

```
ans =  
    16     9     4  
    68    46    24  
    34    23    12
```

```
>> C=B*A'
```

```
C =  
    44    60  
    19    30
```

To perform arithmetic operations, matrix dimensions must agree

```
>> D=A*B  
??? Error using ==> *  
Inner matrix dimensions must agree.
```

The inverse of a square matrix

```
>> inv(C)
```

```
ans =  
    0.1667   -0.3333  
   -0.1056    0.2444
```

The determinant of a square matrix

```
>> det(C)
```

```
ans =  
    180
```

An array or element-by-element arithmetic operation is denoted by a period (.) preceding an operator. Examples are element-by-element multiplication (.\*), division (./), and powers (.^).

```
>> a.*b
```

```
ans =  
     5     0    12
```

```
>> A.*B
```

```
ans =  
     2    30    12  
    14    16     0
```

Matrices in element-by-element operations must obviously have the same dimensions. Mathematical functions applied to arrays are also evaluated element-by-element.

```
>> sin([0 1 2 3 4]*pi/2)
```

```
ans =  
     0     1     0    -1     0
```

**Purpose:**

Show how to handle script files and function files.

**Description:**

When starting a MATLAB session the default working directory is according to initial settings, for example C:\USER. A new working directory can be chosen by typing for example

```
>> cd A:
```

which makes the root directory in drive A the working directory.

Files containing MATLAB and/or CALFEM statements are characterized by the suffix `.m`. For example the file `bar2e.m` contains statements to evaluate the element stiffness matrix for a two dimensional bar element. An `.m`-file is allowed to include references to other `.m`-files, and also to call itself recursively.

Two types of `.m`-files can be used, *script files* and *function files*. *Script files* collect a sequence of statements under one command name. *Function files* allow new functions with input and/or output variables to be defined. Both *script files* and *function files* are ordinary ASCII text files and can therefore be created in an arbitrary editor. In the MATLAB environment an `.m`-file editor can be activated from the pull down menu on top of the MATLAB window.

An example of a *script file* is given below. The following sequence of statements is typed in the `.m`-file editor, and saved as `test.m`.

```
% ----- Script file test.m -----  
A=[0 4;2 8]  
B=[3 9;5 7]  
C=A*B  
% ----- end -----
```

A line starting with an `%` is regarded as a comment line.

The statements are executed by writing the file name (without the suffix `.m`) in the command window

```
>> test
```

```
A =  
    0    4  
    2    8
```

```
B =  
    3    9  
    5    7
```

```
C =  
    20    28  
    46    74
```

The second type of .m-files is the *function file*. The first line of these files contains the word **function**. The example below is a function that computes the second and third power of a scalar.

```
function [b,c]=func1(a)  
% ----- function file 'func1.m' -----  
b=a*a;  
c=a*a*a;  
% ----- end -----
```

The semi-colon prohibits the echo display of the variables on the screen.

The file can be created using an ordinary editor, and it must be saved as **func1.m** i.e. the file name without extension must be the same as the function name.

The second and third power of 2 are calculated by typing

```
>> [b,c]=func1(2)
```

producing

```
b =  
    4
```

```
c =  
    8
```

See also *function* and *script* in Section 7.

**Purpose:**

Show different display formats.

**Description:**

Consider the following matrix operation

```
>> A=[0 4;2 8];  
>> B=[3 9;5 7];  
>> C=A*B/2537
```

```
C =  
    0.0079    0.0110  
    0.0181    0.0292
```

The result from the computation of C above is shown in the default format, displaying four significant decimal digits.

Other formats can be defined by the command **format**

```
>> format long  
>> C
```

```
C =  
    0.00788332676389    0.01103665746945  
    0.01813165155696    0.02916830902641
```

```
>> format short e  
>> C
```

```
C =  
    7.8833e-003    1.1037e-002  
    1.8132e-002    2.9168e-002
```

```
>> format long e  
>> C
```

```
C =  
    7.883326763894364e-003    1.103665746945211e-002  
    1.813165155695703e-002    2.916830902640915e-002
```

**MATLAB introduction****exi5****Purpose:**

How to make a command window session .log-file.

**Description:**

The `diary` and `echo` commands are useful for presentation purposes since the complete set of statements can be saved together with some selected results.

The command

```
>> diary filename
```

saves all subsequent terminal input and resulting output in the file named *filename* on the default device. The file is closed using

```
>> diary off
```

Consider the script file `test.m`.

```
% ----- Script file test.m -----  
diary testlog  
echo on  
A=[0 4;2 8];  
B=[3 9;5 7];  
C=A*B/2537  
echo off  
diary off  
% ----- end -----
```

Normally, the statements in an `.m`-file do not display during execution. The commands `echo on` and `echo off` allow the statements to be viewed as they execute. Execution of `test.m` yields

```
>>test
```

```
A=[0 4;2 8];
```

```
B=[3 9;5 7];
```

```
C=A*B/2537
```

```
C =  
    0.0079    0.0110  
    0.0181    0.0292
```

in the command window and on the file `testlog` as well.



**Purpose:**

How to display vectors and handle the graphics window.

**Description:**

The contents of a vector versus the vector index or a vector versus another vector can be displayed in the graphics window. Consider the vectors

$$\mathbf{x} = \begin{bmatrix} 1 & 2 & 5 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 5 & 22 & 16 \end{bmatrix}$$

The function

```
>> plot(y)
```

plots the contents of the vector *y* versus vector index and

```
>> plot(x,y)
```

plots the contents of the vector *y* versus the vector *x*.

The commands

```
title('text')  
xlabel('xlabel')  
ylabel('ylabel')
```

write *text* as a title of the current plot, and *xlabel* and *ylabel* as labels of the coordinate axis.

Grid lines are added with

```
grid
```

and

```
clf
```

clears the current figure.

### 9.3 Static analysis

This section illustrates some linear static finite element calculations. The examples deal with structural problems as well as field problems such as heat conduction.

Static analysis	
exs1	Linear spring system
exs2	One-dimensional heat flow
exs3	Plane truss
exs4	Plane truss analysed using loops
exs5	Simply supported beam
exs6	Plane frame
exs7	Plane frame stabilized with bars
exs8	Two dimensional diffusion

The introductory example **exs1** illustrates the basic steps in the finite element method for a simple structure of one-dimensional linear springs. The linear spring or analogy element is also used in example **exs2** to solve a problem of heat conduction in a wall. A plane truss consisting of three bars is analysed in **exs3**. In example **exs4** a plane truss consisting of ten bars is analysed using loops. First the analysis is performed by defining coordinate data for each element directly, and then it is shown how this data can be obtained from global coordinate data. A simply supported beam is analysed in example **exs5**. Element forces and the support forces are calculated. A two dimensional plane frame is analysed in example **exs6**. A structure built up from both beams and bars is analysed in example **exs7**. Graphics facilities are also explained in examples **exs6**, **exs7**, and **exs8**.

**Note:** The examples listed above are supplied as **.m**-files under the directory **examples**. The example files are named according to the table.

exs1

Static analysis

**Purpose:**

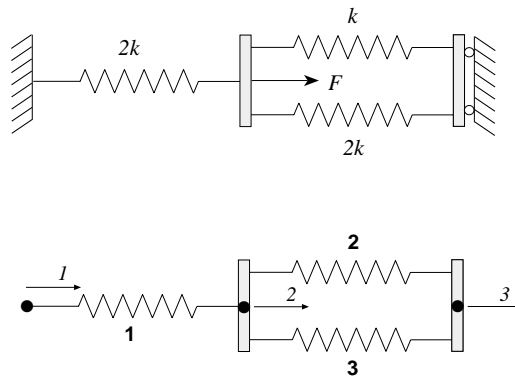
Show the basic steps in a finite element calculation.

**Description:**

The general procedure in linear finite element calculations is carried out for a simple structure. The steps are

- define the model
- generate element matrices
- assemble element matrices into the global system of equations
- solve the global system of equations
- evaluate element forces

Consider the system of three linear elastic springs, and the corresponding finite element model. The system of springs is fixed in its ends and loaded by a single load  $F$ .



The computation is initialized by defining the topology matrix **Edof**, containing element numbers and global element degrees of freedom,

```
>> Edof=[1  1 2;  
          2  2 3;  
          3  2 3];
```

the global stiffness matrix **K** ( $3 \times 3$ ) of zeros,

```
>> K=zeros(3,3)
```

```
K =  
    0    0    0  
    0    0    0  
    0    0    0
```

## Static analysis

exs1

and the load vector  $\mathbf{f}$  ( $3 \times 1$ ) with the load  $F = 100$  in position 2.

```
>> f=zeros(3,1); f(2)=100
```

```
f =  
    0  
   100  
    0
```

Element stiffness matrices are generated by the function `spring1e`. The element property `ep` for the springs contains the spring stiffnesses  $k$  and  $2k$  respectively, where  $k = 1500$ .

```
>> k=1500; ep1=k; ep2=2*k;  
>> Ke1=spring1e(ep1)
```

```
Ke1 =  
      1500      -1500  
     -1500       1500
```

```
>> Ke2=spring1e(ep2)
```

```
Ke2 =  
      3000      -3000  
     -3000       3000
```

The element stiffness matrices are assembled into the global stiffness matrix  $\mathbf{K}$  according to the topology.

```
>> K=assem(Edof(1,:),K,Ke2)
```

```
K =  
      3000      -3000         0  
     -3000       3000         0  
         0         0         0
```

```
>> K=assem(Edof(2,:),K,Ke1)
```

```
K =  
      3000      -3000         0  
     -3000       4500     -1500  
         0     -1500       1500
```

exs1

Static analysis

---

```
>> K=assem(Edof(3,:),K,Ke2)
```

```
K =  
      3000      -3000         0  
     -3000       7500     -4500  
         0      -4500      4500
```

The global system of equations is solved considering the boundary conditions given in bc.

```
>> bc= [1 0; 3 0];  
>> [a,r]=solveq(K,f,bc)
```

```
a =  
      0  
    0.0133  
      0
```

```
r =  
   -40.0000  
         0  
   -60.0000
```

Element forces are evaluated from the element displacements. These are obtained from the global displacements **a** using the function **extract**.

```
>> ed1=extract(Edof(1,:),a)
```

```
ed1 =  
      0      0.0133
```

```
>> ed2=extract(Edof(2,:),a)
```

```
ed2 =  
    0.0133      0
```

```
>> ed3=extract(Edof(3,:),a)
```

```
ed3 =  
    0.0133      0
```

**Static analysis****exs1**

The spring forces are evaluated using the function `spring1s`.

```
>> es1=spring1s(ep2,ed1)
```

```
es1 =  
      40
```

```
>> es2=spring1s(ep1,ed2)
```

```
es2 =  
     -20
```

```
>> es3=spring1s(ep2,ed3)
```

```
es3 =  
     -40
```

## exs2

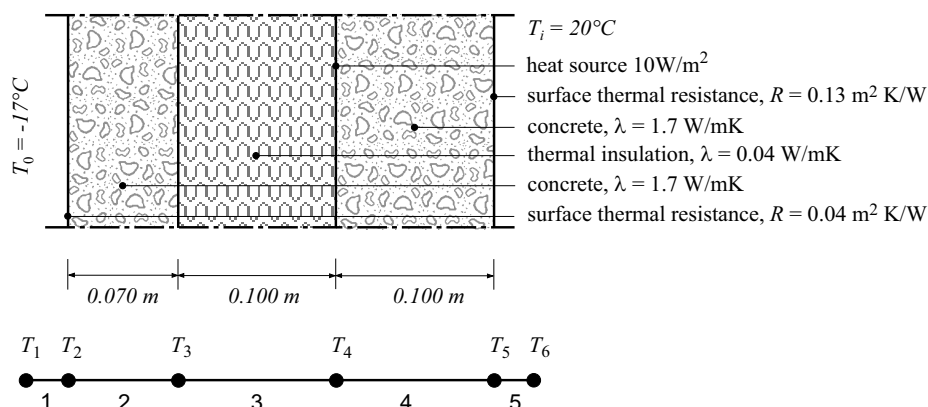
## Static analysis

**Purpose:**

Analysis of one-dimensional heat flow.

**Description:**

Consider a wall built up of concrete and thermal insulation. The outdoor temperature is  $-17^\circ\text{C}$  and the temperature inside is  $20^\circ\text{C}$ . At the inside of the thermal insulation there is a heat source yielding  $10\text{ W/m}^2$ .



The wall is subdivided into five elements and the one-dimensional spring (analogy) element `spring1e` is used. Equivalent spring stiffnesses are  $k_i = \lambda A/L$  for thermal conductivity and  $k_i = A/R$  for thermal surface resistance. Corresponding spring stiffnesses per  $\text{m}^2$  of the wall are:

$$\begin{aligned} k_1 &= 1/0.04 &= 25.0 &\text{ W/K} \\ k_2 &= 1.7/0.070 &= 24.3 &\text{ W/K} \\ k_3 &= 0.040/0.100 &= 0.4 &\text{ W/K} \\ k_4 &= 1.7/0.100 &= 17.0 &\text{ W/K} \\ k_5 &= 1/0.13 &= 7.7 &\text{ W/K} \end{aligned}$$

A global system matrix  $\mathbf{K}$  and a heat flow vector  $\mathbf{f}$  are defined. The heat source inside the wall is considered by setting  $f_4 = 10$ . The element matrices  $\mathbf{K}_e$  are computed using `spring1e`, and the function `assem` assembles the global stiffness matrix.

The system of equations is solved using `solveq` with considerations to the boundary conditions in `bc`. The prescribed temperatures are  $T_1 = -17^\circ\text{C}$  and  $T_6 = 20^\circ\text{C}$ .

```
>> Edof=[1  1 2
          2  2 3;
          3  3 4;
          4  4 5;
          5  5 6];
```

## Static analysis

exs2

```
>> K=zeros(6);
>> f=zeros(6,1); f(4)=10

f =

     0
     0
     0
    10
     0
     0

>> ep1=[25]; ep2=[24.3];
>> ep3=[0.4]; ep4=[17];
>> ep5=[7.7];

>> Ke1=spring1e(ep1); Ke2=spring1e(ep2);
>> Ke3=spring1e(ep3); Ke4=spring1e(ep4);
>> Ke5=spring1e(ep5);

>> K=assem(Edof(1,:),K,Ke1); K=assem(Edof(2,:),K,Ke2);
>> K=assem(Edof(3,:),K,Ke3); K=assem(Edof(4,:),K,Ke4);
>> K=assem(Edof(5,:),K,Ke5);

>> bc=[1 -17; 6 20];

>> [a,r]=solveq(K,f,bc)

a =

-17.0000
-16.4384
-15.8607
 19.2378
 19.4754
 20.0000

r =

-14.0394
  0.0000
 -0.0000
  0
  0.0000
  4.0394
```



The temperature values  $T_i$  in the node points are given in the vector **a** and the boundary flows in the vector **r**.

After solving the system of equations, the heat flow through the wall is computed using **extract** and **spring1s**.

```
>> ed1=extract(Edof(1,:),a);  
>> ed2=extract(Edof(2,:),a);  
>> ed3=extract(Edof(3,:),a);  
>> ed4=extract(Edof(4,:),a);  
>> ed5=extract(Edof(5,:),a);
```

```
>> q1=spring1s(ep1,ed1)
```

```
q1 =
```

```
14.0394
```

```
>> q2=spring1s(ep2,ed2)
```

```
q2 =
```

```
14.0394
```

```
>> q3=spring1s(ep3,ed3)
```

```
q3 =
```

```
14.0394
```

```
>> q4=spring1s(ep4,ed4)
```

```
q4 =
```

```
4.0394
```

```
>> q5=spring1s(ep5,ed5)
```

```
q5 =
```

```
4.0394
```

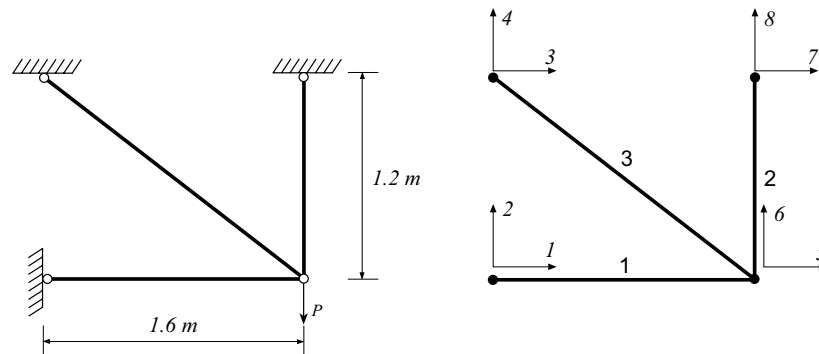
The heat flow through the wall is  $q = 14.0 \text{ W/m}^2$  in the part of the wall to the left of the heat source, and  $q = 4.0 \text{ W/m}^2$  in the part to the right of the heat source.

**Static analysis****exs3****Purpose:**

Analysis of a plane truss.

**Description:**

Consider a plane truss consisting of three bars with the properties  $E = 200$  GPa,  $A_1 = 6.0 \cdot 10^{-4}$  m<sup>2</sup>,  $A_2 = 3.0 \cdot 10^{-4}$  m<sup>2</sup> and  $A_3 = 10.0 \cdot 10^{-4}$  m<sup>2</sup>, and loaded by a single force  $P = 80$  kN. The corresponding finite element model consists of three elements and eight degrees of freedom.



The topology is defined by the matrix

```
>> Edof=[1   1   2   5   6;
          2   5   6   7   8;
          3   3   4   5   6];
```

The stiffness matrix  $K$  and the load vector  $f$ , are defined by

```
>> K=zeros(8);
    f=zeros(8,1); f(6)=-80e3;
```

The element property vectors  $ep1$ ,  $ep2$  and  $ep3$  are defined by

```
>> E=2.0e11;
>> A1=6.0e-4;    A2=3.0e-4;    A3=10.0e-4;
>> ep1=[E A1];   ep2=[E A2];   ep3=[E A3];
```

and the element coordinate vectors  $ex1$ ,  $ex2$ ,  $ex3$ ,  $ey1$ ,  $ey2$  and  $ey3$  by

```
>> ex1=[0 1.6];   ex2=[1.6 1.6];   ex3=[0 1.6];
>> ey1=[0 0];     ey2=[0 1.2];     ey3=[1.2 0];
```

## exs3

Static analysis

---

The element stiffness matrices **Ke1**, **Ke2** and **Ke3** are computed using **bar2e**.

```
>> Ke1=bar2e(ex1,ey1,ep1)
```

**Ke1** =

```
1.0e+007 *  
  
    7.5000         0    -7.5000         0  
         0         0         0         0  
   -7.5000         0    7.5000         0  
         0         0         0         0
```

```
>> Ke2=bar2e(ex2,ey2,ep2)
```

**Ke2** =

```
1.0e+007 *  
  
         0         0         0         0  
         0    5.0000         0   -5.0000  
         0         0         0         0  
         0   -5.0000         0    5.0000
```

```
>> Ke3=bar2e(ex3,ey3,ep3)
```

**Ke3** =

```
1.0e+007 *  
  
    6.4000   -4.8000   -6.4000    4.8000  
   -4.8000    3.6000    4.8000   -3.6000  
   -6.4000    4.8000    6.4000   -4.8000  
    4.8000   -3.6000   -4.8000    3.6000
```

Based on the topology information, the global stiffness matrix can be generated by assembling the element stiffness matrices

```
>> K=assem(Edof(1,:),K,Ke1);  
>> K=assem(Edof(2,:),K,Ke2);  
>> K=assem(Edof(3,:),K,Ke3)
```

## Static analysis

exs3

```
K =  
  
1.0e+008 *  
  
Columns 1 through 7  
  
    0.7500         0         0         0    -0.7500         0         0  
         0         0         0         0         0         0         0  
         0         0     0.6400    -0.4800    -0.6400     0.4800         0  
         0         0    -0.4800     0.3600     0.4800    -0.3600         0  
    -0.7500         0    -0.6400     0.4800     1.3900    -0.4800         0  
         0         0     0.4800    -0.3600    -0.4800     0.8600         0  
         0         0         0         0         0         0         0  
         0         0         0         0         0     -0.5000         0  
  
Column 8  
  
         0  
         0  
         0  
         0  
         0  
    -0.5000  
         0  
     0.5000
```

Considering the prescribed displacements in **bc**, the system of equations is solved using the function **solveq**, yielding displacements **a** and support forces **r**.

```
>> bc= [1 0;2 0;3 0;4 0;7 0;8 0];  
>> [a,r]=solveq(K,f,bc)
```

```
a =  
  
1.0e-002 *  
  
         0  
         0  
         0  
         0  
    -0.0398  
    -0.1152  
         0  
         0
```

exs3

Static analysis

---

```
r =  
  
1.0e+004 *  
  
    2.9845  
         0  
   -2.9845  
    2.2383  
    0.0000  
    0.0000  
         0  
    5.7617
```

The vertical displacement at the point of loading is 1.15 mm. The section forces **es1**, **es2** and **es3** are calculated using **bar2s** from element displacements **ed1**, **ed2** and **ed3** obtained using **extract**.

```
>> ed1=extract(Edof(1,:),a);  
>> N1=bar2s(ex1,ey1,ep1,ed1)
```

```
N1 =  
  
-2.9845e+004
```

```
>> ed2=extract(Edof(2,:),a);  
>> N2=bar2s(ex2,ey2,ep2,ed2)
```

```
N2 =  
  
5.7617e+004
```

```
>> ed3=extract(Edof(3,:),a);  
>> N3=bar2s(ex3,ey3,ep3,ed3)
```

```
N3 =  
  
3.7306e+004
```

i.e., the normal forces are  $N_1 = -29.84$  kN,  $N_2 = 57.62$  kN and  $N_3 = 37.31$  kN.

## Static analysis

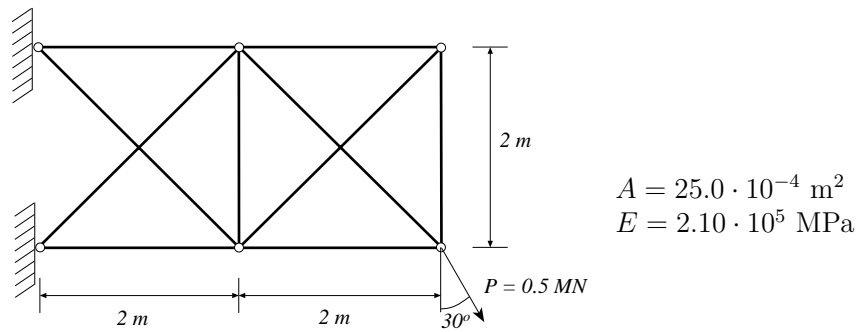
exs4

### Purpose:

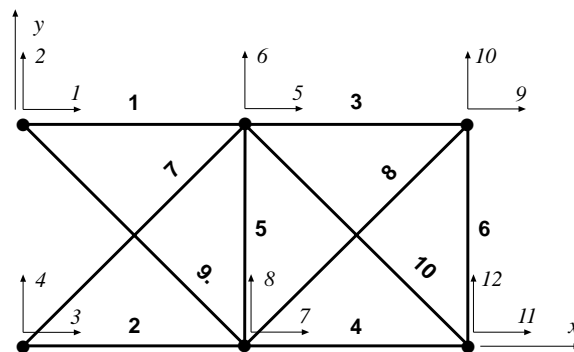
Analysis of a plane truss.

### Description:

Consider a plane truss, loaded by a single force  $P = 0.5$  MN.



The corresponding finite element model consists of ten elements and twelve degrees of freedom.



The topology is defined by the matrix

```
>> Edof=[1  1  2  5  6;
          2  3  4  7  8;
          3  5  6  9 10;
          4  7  8 11 12;
          5  7  8  5  6;
          6 11 12  9 10;
          7  3  4  5  6;
          8  7  8  9 10;
          9  1  2  7  8;
         10  5  6 11 12];
```

## exs4

Static analysis

---

A global stiffness matrix  $K$  and a load vector  $f$  are defined. The load  $P$  is divided into  $x$  and  $y$  components and inserted in the load vector  $f$ .

```
>> K=zeros(12);  
>> f=zeros(12,1); f(11)=0.5e6*sin(pi/6); f(12)=-0.5e6*cos(pi/6);
```

The element matrices  $K_e$  are computed by the function `bar2e`. These matrices are then assembled in the global stiffness matrix using the function `assem`.

```
>> A=25.0e-4; E=2.1e11; ep=[E A];
```

```
>> Ex=[0 2;  
      0 2;  
      2 4;  
      2 4;  
      2 2;  
      4 4;  
      0 2;  
      2 4;  
      0 2;  
      2 4];
```

```
>> Ey=[2 2;  
      0 0;  
      2 2;  
      0 0;  
      0 2;  
      0 2;  
      0 2;  
      0 2;  
      2 0;  
      2 0];
```

All the element matrices are computed and assembled in the loop

```
>> for i=1:10  
      Ke=bar2e(Ex(i,:),Ey(i,:),ep);  
      K=assem(Edof(i,:),K,Ke);  
end;
```

The displacements in  $a$  and the support forces in  $r$  are computed by solving the system of equations considering the boundary conditions in  $bc$ .

```
>> bc=[1 0;2 0;3 0;4 0];  
>> [a,r]=solveq(K,f,bc)
```

## Static analysis

exs4

```
a =  
      0  
      0  
      0  
      0  
    0.0024  
   -0.0045  
   -0.0016  
   -0.0042  
    0.0030  
   -0.0107  
   -0.0017  
   -0.0113  
  
r =  
  
1.0e+005 *  
  
   -8.6603  
    2.4009  
    6.1603  
    1.9293  
    0.0000  
   -0.0000  
   -0.0000  
   -0.0000  
    0.0000  
    0.0000  
    0.0000  
    0.0000  
    0.0000
```

The displacement at the point of loading is  $-1.7 \cdot 10^{-3}$  m in the x-direction and  $-11.3 \cdot 10^{-3}$  m in the y-direction. At the upper support the horizontal force is  $-0.866$  MN and the vertical  $0.240$  MN. At the lower support the forces are  $0.616$  MN and  $0.193$  MN, respectively.

Normal forces are evaluated from element displacements. These are obtained from the global displacements **a** using the function **extract**. The normal forces are evaluated using the function **bar2s**.

```
ed=extract(Edof,a);  
  
>> for i=1:10  
      N(i,:)=bar2s(Ex(i,:),Ey(i,:),ep,ed(i,:));  
    end
```



The obtained normal forces are

```
>> N
```

```
N =
```

```
1.0e+005 *  
  
    6.2594  
   -4.2310  
    1.7064  
   -0.1237  
   -0.6945  
    1.7064  
   -2.7284  
   -2.4132  
    3.3953  
    3.7105
```

The largest normal force  $N = 0.626$  MN is obtained in element 1 and is equivalent to a normal stress  $\sigma = 250$  MPa.

To reduce the quantity of input data, the element coordinate matrices **Ex** and **Ey** can alternatively be created from a global coordinate matrix **Coord** and a global topology matrix **Coord** using the function **coordxtr**, i.e.

```
>> Coord=[0 2;  
           0 0;  
           2 2;  
           2 0;  
           4 2;  
           4 0];  
  
>> Dof=[ 1 2;  
        3 4;  
        5 6;  
        7 8;  
        9 10;  
       11 12];  
  
>> [ex,ey]=coordxtr(Edof,Coord,Dof,2);
```

## Static analysis

exs5

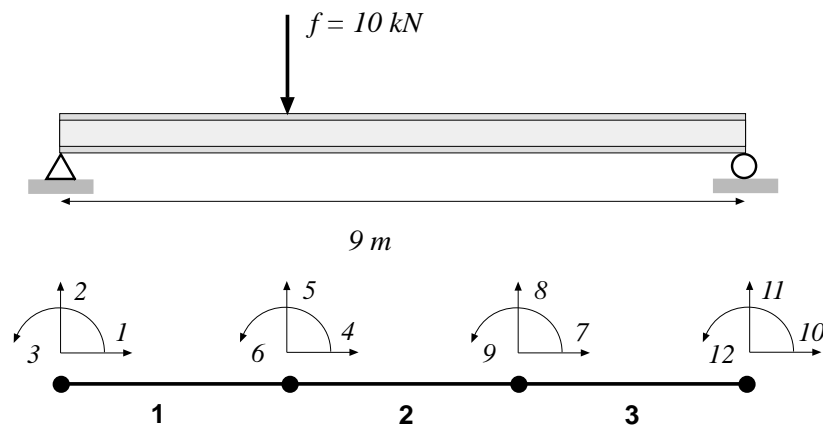
**Purpose:**

Analysis of a simply supported beam.

**Description:**

Consider the simply supported beam loaded by a single load  $f = 10000$  N, applied at a point 1 meter from the left support. The corresponding finite element mesh is also shown. The following data apply to the beam

Young's modulus  $E = 2.10 \cdot 10^{11}$  Pa  
 Cross section area  $A = 45.3 \cdot 10^{-4}$  m<sup>2</sup>  
 Moment of inertia  $I = 2510 \cdot 10^{-8}$  m<sup>4</sup>



The element topology is defined by the topology matrix

```
>> Edof=[1  1  2  3  4  5  6
          2  4  5  6  7  8  9
          3  7  8  9 10 11 12];
```

The system matrices, i.e. the stiffness matrix  $K$  and the load vector  $f$ , are defined by

```
>> K=zeros(12); f=zeros(12,1); f(5)=-10000;
```

The element property vector  $ep$ , the element coordinate vectors  $ex$  and  $ey$ , and the element stiffness matrix  $Ke$ , are generated. Note that the same coordinate vectors are applicable for all elements because they are identical.

exs5

Static analysis

---

```
>> E=2.1e11;      A=45.3e-4;      I=2510e-8;      ep=[E A I];
```

```
>> ex=[0 3];      ey=[0 0];
```

```
>> Ke=beam2e(ex,ey,ep)
```

Ke =

1.0e+008 \*

3.1710	0	0	-3.1710	0	0
0	0.0234	0.0351	0	-0.0234	0.0351
0	0.0351	0.0703	0	-0.0351	0.0351
-3.1710	0	0	3.1710	0	0
0	-0.0234	-0.0351	0	0.0234	-0.0351
0	0.0351	0.0351	0	-0.0351	0.0703

Based on the topology information, the global stiffness matrix can be generated by assembling the element stiffness matrices

```
>> K=assem(Edof,K,Ke);
```

Finally, the solution can be calculated by defining the boundary conditions in `bc` and solving the system of equations. Displacements `a` and support forces `r` are computed by the function `solveq`.

```
>> bc=[1 0; 2 0; 11 0];  
[a,r]=solveq(K,f,bc);
```

The section forces `es` are calculated from element displacements `Ed`

```
>> Ed=extract(Edof,a);
```

```
>> es1=beam2s(ex,ey,ep,Ed(1,:));
```

```
>> es2=beam2s(ex,ey,ep,Ed(2,:));
```

```
>> es3=beam2s(ex,ey,ep,Ed(3,:));
```

## Static analysis

exs5

## Results

```
a =
      0
      0
    -0.0095
      0
    -0.0228
    -0.0038
      0
    -0.0199
    0.0047
      0
      0
    0.0076

r =
1.0e+003 *
      0
    6.6667
    -0.0000
      0
    0.0000
    -0.0000
      0
    0.0000
    0.0000
      0
    3.3333
    -0.0000
```

## es1 =

```
1.0e+004 *
      0  -0.6667  0.0000
      0  -0.6667  2.0000
```

## es2 =

```
1.0e+004 *
      0  0.3333  2.0000
      0  0.3333  1.0000
```

## es3 =

```
1.0e+004 *
      0  0.3333  1.0000
      0  0.3333  -0.0000
```

## exs6

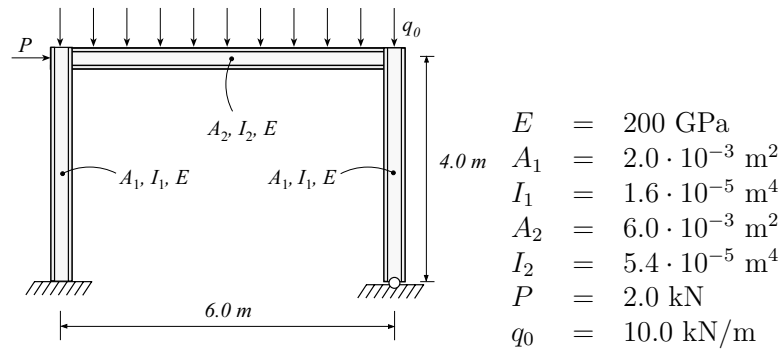
## Static analysis

**Purpose:**

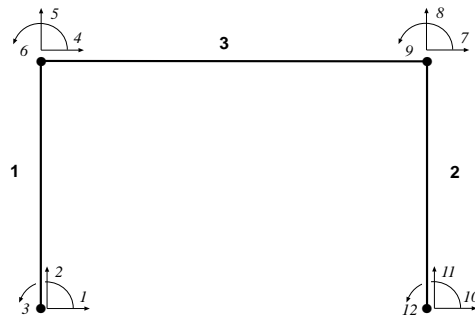
Analysis of a plane frame.

**Description:**

A frame consists of one horizontal and two vertical beams according to the figure.



The corresponding finite element model consists of three beam elements and twelve degrees of freedom.



A topology matrix **Edof**, a global stiffness matrix **K** and load vector **f** are defined. The element matrices **Ke** and **fe** are computed by the function **beam2e**. These matrices are then assembled in the global matrices using the function **assem**.

```
>> Edof=[1  4  5  6  1  2  3;
          2  7  8  9 10 11 12;
          3  4  5  6  7  8  9];

>> K=zeros(12); f=zeros(12,1); f(4)=2e+3;

>> E=200e9;
>> A1=2e-3;  A2=6e-3;
>> I1=1.6e-5; I2=5.4e-5;
>> ep1=[E A1 I1]; ep3=[E A2 I2];
```

## Static analysis

exs6

```
>> ex1=[0 0]; ex2=[6 6]; ex3=[0 6];
>> ey1=[0 4]; ey2=[0 4]; ey3=[4 4];
>> eq1=[0 0]; eq2=[0 0]; eq3=[0 -10e+3];

>> Ke1=beam2e(ex1,ey1,ep1);
>> Ke2=beam2e(ex2,ey2,ep1);
>> [Ke3,fe3]=beam2e(ex3,ey3,ep3,eq3);

>> K=assem(Edof(1,:),K,Ke1);
>> K=assem(Edof(2,:),K,Ke2);
>> [K,f]=assem(Edof(3,:),K,Ke3,f,fe3);
```

The system of equations are solved considering the boundary conditions in bc.

```
>> bc=[1 0; 2 0; 3 0; 10 0; 11 0];
>> [a,r]=solveq(K,f,bc)
```

a =	r =
0	1.0e+004 *
0	
0	0.1927
0.0075	2.8741
-0.0003	0.0445
-0.0054	0
0.0075	0.0000
-0.0003	-0.0000
0.0047	-0.0000
0	0
0	0.0000
-0.0052	-0.3927
	3.1259
	0

The element displacements are obtained from the function `extract`, and the function `beam2s` computes the section forces.

```
>> Ed=extract(Edof,a);
>> es1=beam2s(ex1,ey1,ep1,Ed(1,:),eq1,21)
es1 =

1.0e+004 *

-2.8741    0.1927    0.8152
-2.8741    0.1927    0.7767
.
-2.8741    0.1927    0.0445
```

exs6

Static analysis

---

```
>> es2=beam2s(ex2,ey2,ep1,Ed(2,:),eq2,21)
```

```
es2 =
```

```
1.0e+004 *  
  
-3.1259   -0.3927   -1.5707  
-3.1259   -0.3927   -1.4922  
.  
-3.1259   -0.3927   -0.0000
```

```
>> es3=beam2s(ex3,ey3,ep3,Ed(3,:),eq3,21)
```

```
es3 =
```

```
1.0e+004 *  
  
-0.3927   -2.8741   -0.8152  
-0.3927   -2.5741    0.0020  
.  
-0.3927    3.1259   -1.5707
```

A displacement diagram is displayed using the function `eldisp2` and section force diagrams using the function `eldia2`.

```
>> figure(1)  
>> plotpar=[2 1 0];  
>> eldraw2(ex1,ey1,plotpar);  
>> eldraw2(ex2,ey2,plotpar);  
>> eldraw2(ex3,ey3,plotpar);  
>> sfac=scalfact2(ex3,ey3,Ed(3,:),0.1);  
>> plotpar=[1 2 1];  
>> eldisp2(ex1,ey1,Ed(1,:),plotpar,sfac);  
>> eldisp2(ex2,ey2,Ed(2,:),plotpar,sfac);  
>> eldisp2(ex3,ey3,Ed(3,:),plotpar,sfac);  
>> axis([-1.5 7.5 -0.5 5.5]);  
>> pltscalb2(sfac,[1e-2 0.5 0]);  
>> axis([-1.5 7.5 -0.5 5.5]);  
>> title('displacements')
```

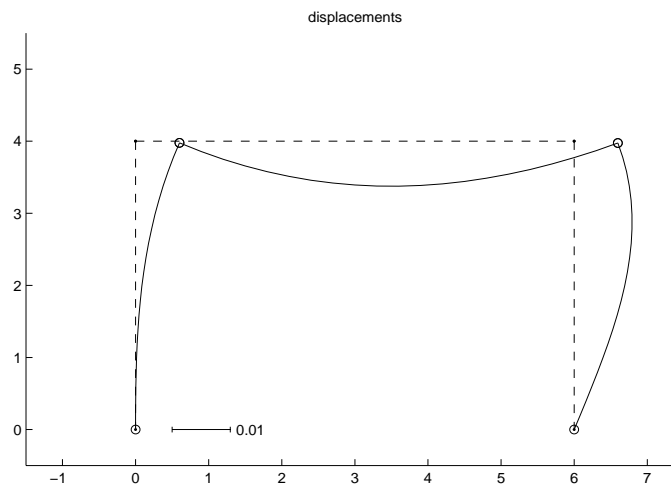
## Static analysis

exs6

```
>> figure(2)
>> plotpar=[2 1];
>> sfac=scalfact2(ex1,ey1,es1(:,1),0.2);
>> eldia2(ex1,ey1,es1(:,1),plotpar,sfac);
>> eldia2(ex2,ey2,es2(:,1),plotpar,sfac);
>> eldia2(ex3,ey3,es3(:,1),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[3e4 1.5 0]);
>> title('normal force')

>> figure(3)
>> sfac=scalfact2(ex3,ey3,es3(:,2),0.2);
>> eldia2(ex1,ey1,es1(:,2),plotpar,sfac);
>> eldia2(ex2,ey2,es2(:,2),plotpar,sfac);
>> eldia2(ex3,ey3,es3(:,2),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[3e4 0.5 0]);
>> title('shear force')

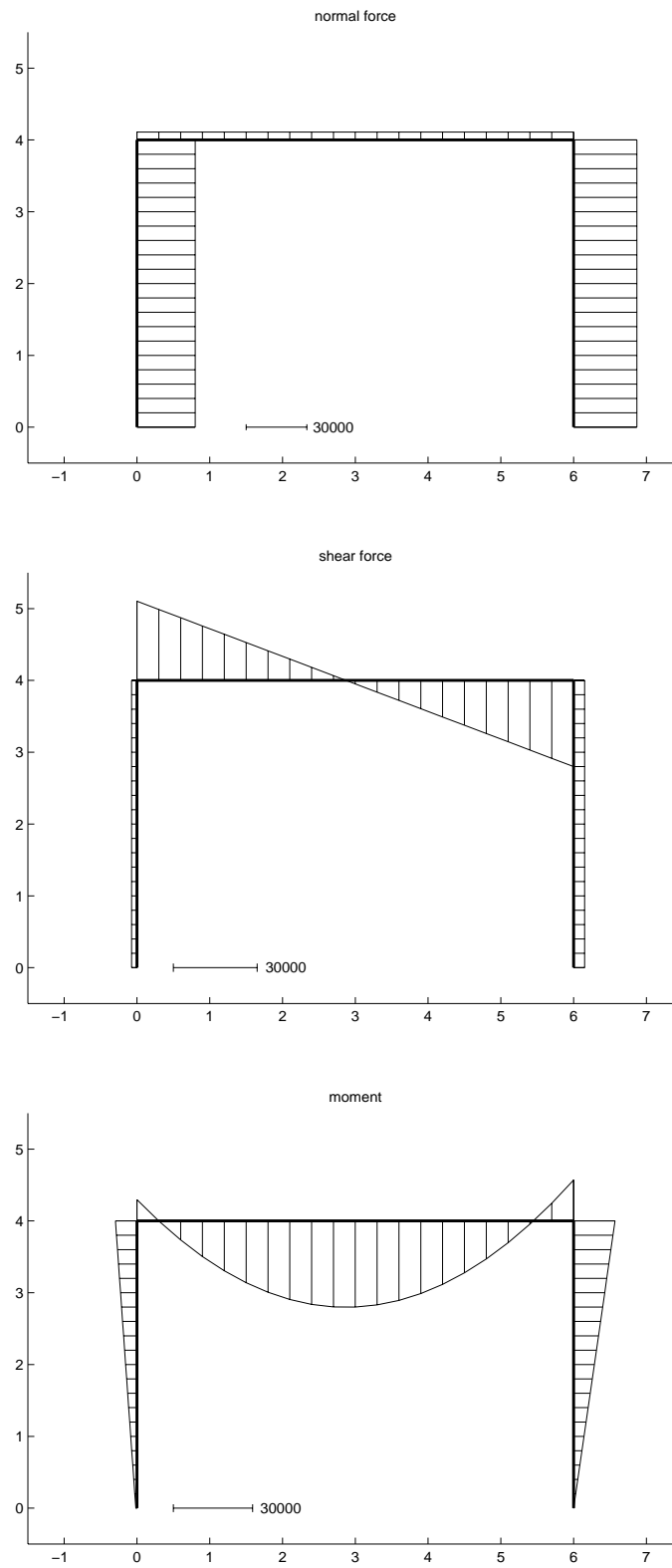
>> figure(4)
>> sfac=scalfact2(ex3,ey3,es3(:,3),0.2);
>> eldia2(ex1,ey1,es1(:,3),plotpar,sfac);
>> eldia2(ex2,ey2,es2(:,3),plotpar,sfac);
>> eldia2(ex3,ey3,es3(:,3),plotpar,sfac);
>> axis([-1.5 7.5 -0.5 5.5]);
>> pltscalb2(sfac,[3e4 0.5 0]);
>> title('moment')
```





exs6

Static analysis



EXAMPLES

9.3 – 24

## Static analysis

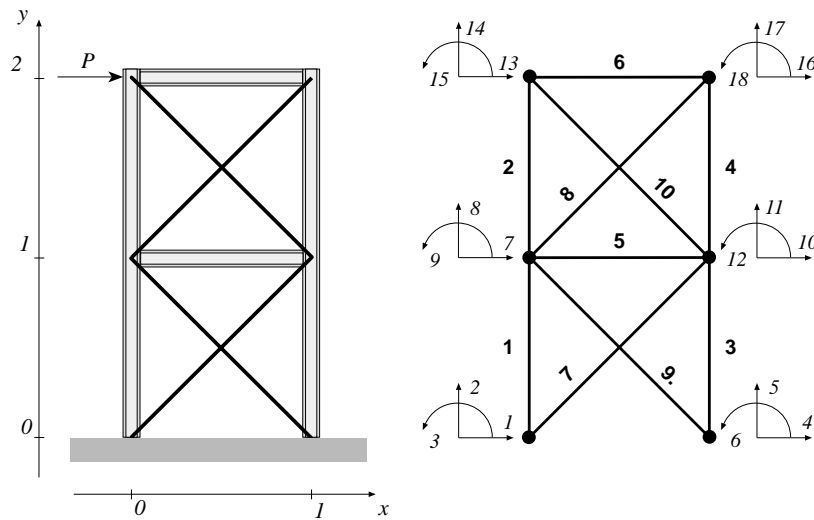
exs7

**Purpose:**

Set up a frame, consisting of both beams and bars, and illustrate the calculations by use of graphics functions.

**Description:**

A frame consists of horizontal and vertical beams, and is stabilized with diagonal bars.



The frame with its coordinates and loading is shown in the left figure, and the finite element model in the right. In the following, the statements for analysing the frame are given as an `.m`-file.

The matrices of the global system i.e. the stiffness matrix  $K$ , the load vector  $f$ , the coordinate matrix  $Coord$ , and the corresponding degrees of freedom matrix  $Dof$  are defined by

```
% ----- System matrices -----

K=zeros(18,18); f=zeros(18,1); f(13)=1;

Coord=[0 0;
        1 0;
        0 1;
        1 1;
        0 2;
        1 2];
```

```
Dof=[1 2 3;
      4 5 6;
      7 8 9;
      10 11 12;
      13 14 15;
      16 17 18];
```

The material properties, the topology, and the element coordinates for the beams and bars respectively, are defined by

```
% ----- Element properties, topology and coordinates -----
```

```
ep1=[1 1 1];
Edof1=[1 1 2 3 7 8 9;
        2 7 8 9 13 14 15;
        3 4 5 6 10 11 12;
        4 10 11 12 16 17 18;
        5 7 8 9 10 11 12;
        6 13 14 15 16 17 18];
[Ex1,Ey1]=coordxtr(Edof1,Coord,Dof,2);
```

```
ep2=[1 1];
Edof2=[7 1 2 10 11;
        8 7 8 16 17;
        9 7 8 4 5;
        10 13 14 10 11];
[Ex2,Ey2]=coordxtr(Edof2,Coord,Dof,2);
```

To check the model, the finite element mesh can be drawn.

```
eldraw2(Ex1,Ey1,[1 3 1]);
eldraw2(Ex2,Ey2,[1 2 1]);
```

The element stiffness matrices are generated and assembled in two loops, one for the beams and one for the bars. The element stiffness functions `beam2e` and `bar2e` use the element coordinate matrices `ex` and `ey`. These matrices are extracted from the global coordinates `Coord` by the function `coordxtr` above.

## Static analysis

exs7

```
% ----- Create and assemble element matrices -----
```

```
for i=1:6
    Ke=beam2e(Ex1(i,:),Ey1(i,:),ep1);
    K=assem(Edof1(i,:),K,Ke);
end
```

```
for i=1:4
    Ke=bar2e(Ex2(i,:),Ey2(i,:),ep2);
    K=assem(Edof2(i,:),K,Ke);
end
```

The global system of equations is solved considering the boundary conditions in `bc`,

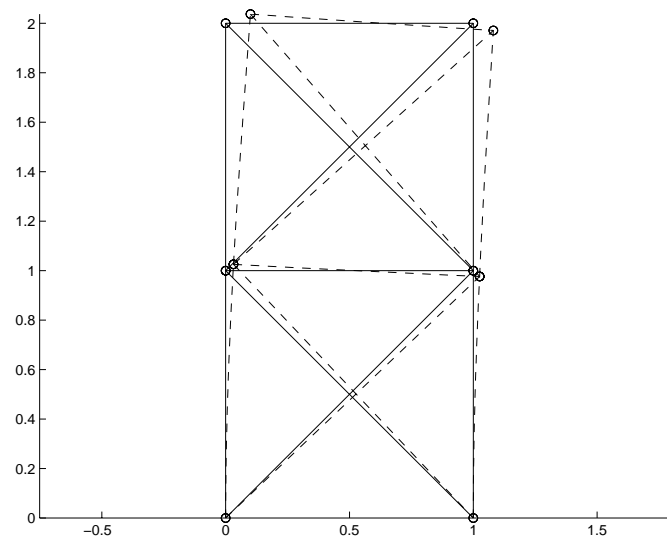
```
% ----- Solve equation system -----
```

```
bc= [1 0; 2 0; 3 0; 4 0; 5 0; 6 0]; [a,r]=solveq(K,f,bc);
```

and the deformed frame is displayed by the function `eldisp2`, where the displacements are scaled by the variable `sfac`.

```
Ed1=extract(Edof1,a);
Ed2=extract(Edof2,a);
```

```
[sfac]=scalfact2(Ex1,Ey1,Ed1,0.1);
eldisp2(Ex1,Ey1,Ed1,[2 1 1],sfac);
eldisp2(Ex2,Ey2,Ed2,[2 1 1],sfac);
```

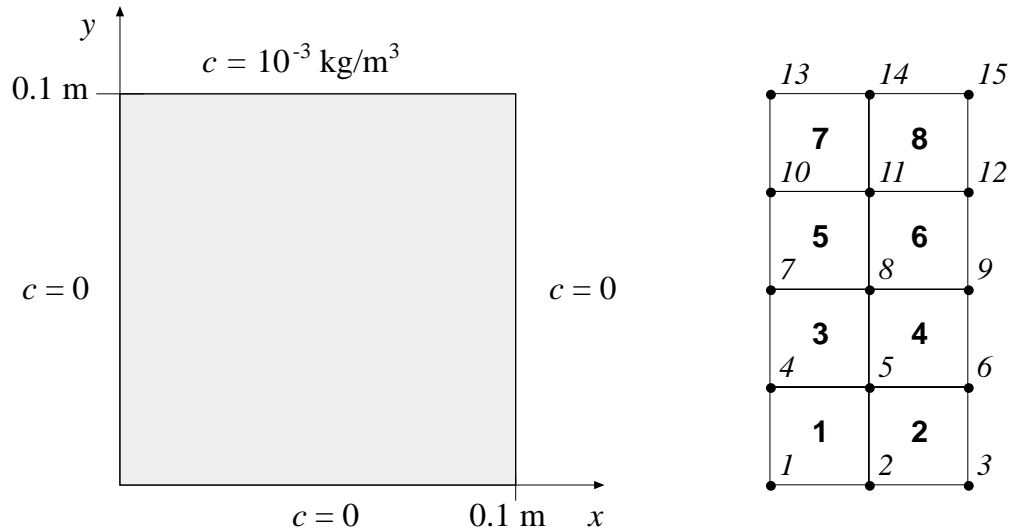


exs8

Static analysis

**Purpose:**

Analysis of two dimensional diffusion.

**Description:****Description:**

Consider a filter paper of square shape. Three sides are in contact with pure water and the fourth side is in contact with a solution of concentration  $c = 1.0 \cdot 10^{-3} \text{ kg/m}^3$ . The length of each side is 0.100 m. Using symmetry, only half of the paper has to be analyzed. The paper and the corresponding finite element mesh are shown. The following boundary conditions are applied

$$c(0, y) = c(x, 0) = c(0.1, y) = 0$$

$$c(x, 0.1) = 10^{-3}$$

The element topology is defined by the topology matrix

```
>> Edof=[1   1   2   5   4
          2   2   3   6   5
          3   4   5   8   7
          4   5   6   9   8
          5   7   8  11  10
          6   8   9  12  11
          7  10  11  14  13
          8  11  12  15  14];
```

## Static analysis

exs8

The system matrices, i.e. the stiffness matrix  $\mathbf{K}$  and the load vector  $\mathbf{f}$ , are defined by

```
>> K=zeros(15);    f=zeros(15,1);
```

Because of the same geometry, orientation, and constitutive matrix for all elements, only one element stiffness matrix  $\mathbf{K}_e$  has to be computed. This is done by the function `flw2qe`.

```
>> ep=1;          D=[1 0; 0 1];

>> ex=[0 0.025 0.025 0];    ey=[0 0 0.025 0.025];

>> Ke=flw2qe(ex,ey,ep,D)

>> Ke =
```

```
    0.7500    -0.2500    -0.2500    -0.2500
   -0.2500     0.7500    -0.2500    -0.2500
   -0.2500    -0.2500     0.7500    -0.2500
   -0.2500    -0.2500    -0.2500     0.7500
```

Based on the topology information, the global stiffness matrix is generated by assembling this element stiffness matrix  $\mathbf{K}_e$  in the global stiffness matrix  $\mathbf{K}$

```
>> K=assem(Edof,K,Ke);
```

Finally, the solution is calculated by defining the boundary conditions `bc` and solving the system of equations. The boundary condition at dof 13 is set to  $0.5 \cdot 10^{-3}$  as an average of the concentrations at the neighbouring boundaries. Concentrations `a` and unknown boundary flows `r` are computed by the function `solveq`.

```
>> bc=[1 0;2 0;3 0;4 0;7 0;10 0;13 0.5e-3;14 1e-3;15 1e-3];

>> [a,r]=solveq(K,f,bc);
```

The element flows `q` are calculated from element concentration `Ed`

```
>> Ed=extract(Edof,a);

>> for i=1:8
    Es=flw2qs(ex,ey,ep,D,Ed(i,:));
end
```

exs8

Static analysis

Results

a=	r=
1.0e-003 *	1.0e-003 *
0	-0.0165
0	-0.0565
0	-0.0399
0	-0.0777
0.0662	0.0000
0.0935	0
0	-0.2143
0.1786	0.0000
0.2500	0.0000
0	-0.6366
0.4338	0.0000
0.5494	-0.0000
0.5000	0.0165
1.0000	0.7707
1.0000	0.2542

Es =	
-0.0013	-0.0013
-0.0005	-0.0032
-0.0049	-0.0022
-0.0020	-0.0054
-0.0122	-0.0051
-0.0037	-0.0111
-0.0187	-0.0213
-0.0023	-0.0203

## Static analysis

exs8

The following .m-file shows an alternative set of commands to perform the diffusion analysis of exs8. By use of global coordinates, an FE-mesh is generated. Also plots with flux-vectors and contour lines are created.

```
% ----- System matrices -----

K=zeros(15); f=zeros(15,1);
Coord=[0      0      ; 0.025 0      ; 0.05  0
        0      0.025; 0.025 0.025; 0.05  0.025
        0      0.05  ; 0.025 0.05  ; 0.05  0.05
        0      0.075; 0.025 0.075; 0.05  0.075
        0      0.1   ; 0.025 0.1   ; 0.05  0.1   ];
Dof=[1; 2; 3
     4; 5; 6
     7; 8; 9
    10;11;12
    13;14;15];

% ----- Element properties, topology and coordinates -----

ep=1; D=[1 0;0 1];
Edof=[1  1  2  5  4
      2  2  3  6  5
      3  4  5  8  7
      4  5  6  9  8
      5  7  8 11 10
      6  8  9 12 11
      7 10 11 14 13
      8 11 12 15 14];
[Ex,Ey]=coor dxtr(Edof,Coord,Dof,4);

% ----- Generate FE-mesh -----

eldraw2(Ex,Ey,[1 3 0],Edof(:,1));
pause; clf;

% ----- Create and assemble element matrices -----

for i=1:8
    Ke=flw2qe(Ex(i,:),Ey(i,:),ep,D);
    K=assem(Edof(i,:),K,Ke);
end;

% ----- Solve equation system -----

bc=[1 0;2 0;3 0;4 0;7 0;10 0;13 0.5e-3;14 1e-3;15 1e-3];
```



exs8

Static analysis

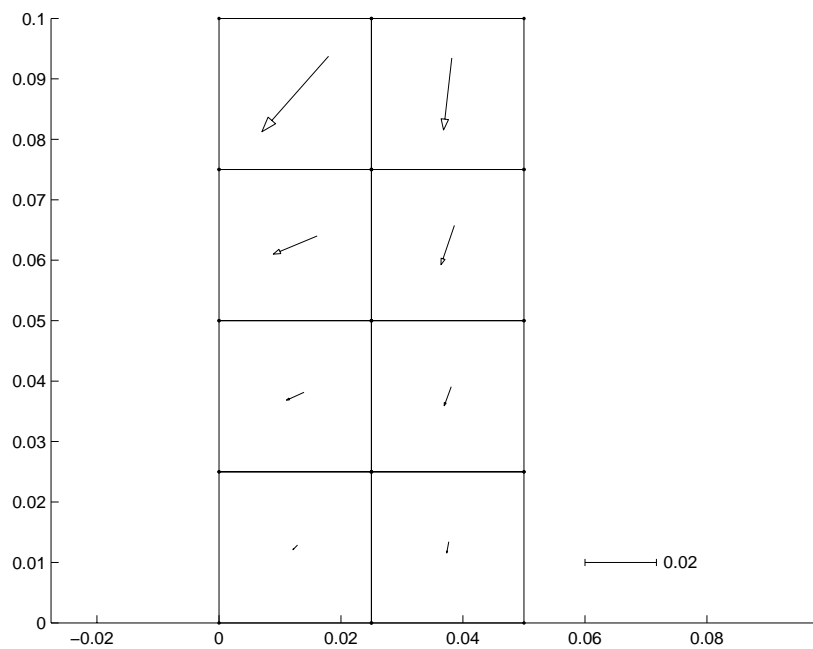
---

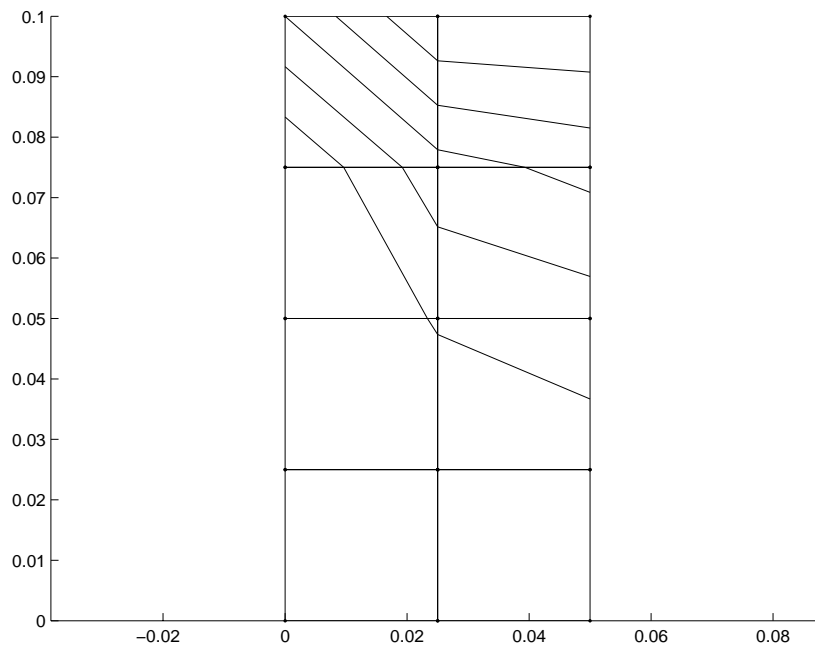
```
[a,r]=solveq(K,f,bc)

% ----- Compute element flux vectors -----

Ed=extract(Edof,a);
for i=1:8
    Es(i,:)=flw2qs(Ex(i,:),Ey(i,:),ep,D,Ed(i,:))
end

% ----- Draw flux vectors and contour lines -----
sfac=scalfact2(Ex,Ey,Es,0.5);
eldraw2(Ex,Ey,[1,3,0]);
elflux2(Ex,Ey,Es,[1,4],sfac);
pltscalb2(sfac,[2e-2 0.06 0.01],4);
pause; clf;
eldraw2(Ex,Ey,[1,3,0]);
eliso2(Ex,Ey,Ed,5,[1,4]);
```

*Flux vectors*

*Contour lines*

Two comments concerning the contour lines:

In the upper left corner, the contour lines should physically have met at the corner point. However, the drawing of the contour lines for the **plane** element follows the numerical approximation along the element boundaries, i.e. a linear variation. A finer element mesh will bring the contour lines closer to the corner point.

Along the symmetry line, the contour lines should physically be perpendicular to the boundary. This will also be improved with a finer element mesh.

With the MATLAB functions `colormap` and `fill` a color plot of the concentrations can be obtained.

```
colormap('jet')
fill(Ex',Ey',Ed')
axis equal
```



## 9.4 Dynamic analysis

This section concerns linear dynamic finite element calculations. The examples illustrate some basic features in dynamics such as modal analysis and time stepping procedures.

Dynamic analysis	
exd1	Modal analysis of frame
exd2	Transient analysis
exd3	Reduced system transient analysis
exd4	Time varying boundary condition

**Note:** The examples listed above are supplied as .m-files under the directory `examples`. The example files are named according to the table.

## exd1

## Dynamic analysis

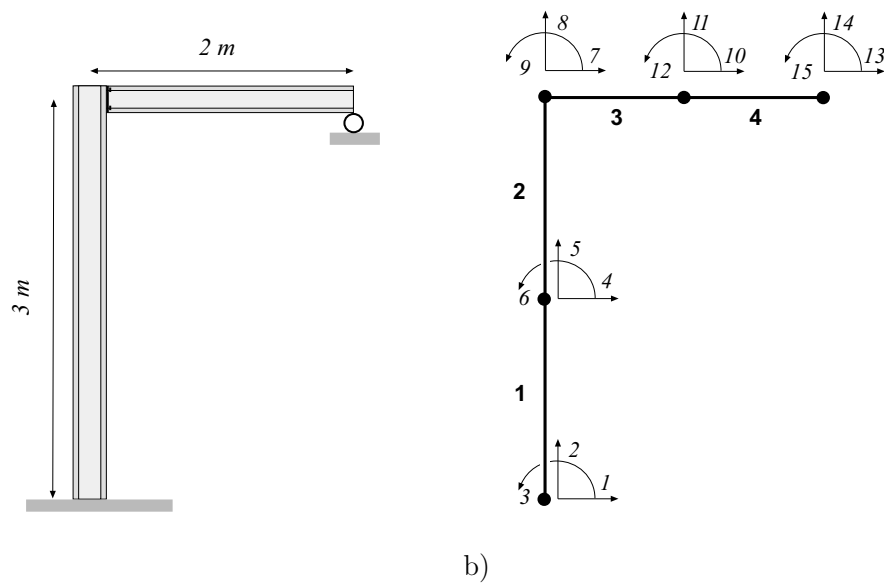
**Purpose:**

Set up the finite element model and perform eigenvalue analysis for a simple frame structure.

**Description:**

Consider the two dimensional frame shown below. A vertical beam is fixed at its lower end, and connected to a horizontal beam at its upper end. The horizontal beam is simply supported at the right end. The length of the vertical beam is 3 m and of the horizontal beam 2 m. The following data apply to the beams

	vertical beam	horizontal beam
Young's modulus (N/m <sup>2</sup> )	$3 \cdot 10^{10}$	$3 \cdot 10^{10}$
Cross section area (m <sup>2</sup> )	$0.1030 \cdot 10^{-2}$	$0.0764 \cdot 10^{-2}$
Moment of inertia (m <sup>4</sup> )	$0.171 \cdot 10^{-5}$	$0.0801 \cdot 10^{-5}$
Density (kg/m <sup>3</sup> )	2500	2500



The structure is divided into 4 elements. The numbering of elements and degrees-of-freedom are apparent from the figure. The following .m-file defines the finite element model.

```
% --- material data -----
E=3e10;          rho=2500;
Av=0.1030e-2;    Iv=0.0171e-4;          % IPE100
Ah=0.0764e-2;    Ih=0.00801e-4;        % IPE80
epv=[E Av Iv rho*Av]; eph=[E Ah Ih rho*Ah];
```

## Dynamic analysis

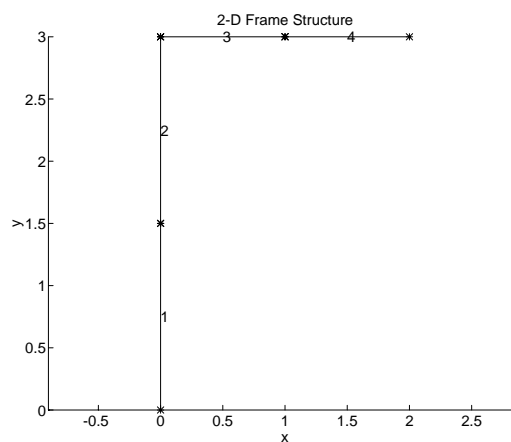
exd1

---

```
% --- topology -----
Edof=[1   1  2  3  4  5  6
      2   4  5  6  7  8  9
      3   7  8  9 10 11 12
      4  10 11 12 13 14 15];
% --- list of coordinates -----
Coord=[0 0; 0 1.5; 0 3; 1 3; 2 3];
% --- list of degrees-of-freedom -----
Dof=[1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15];
% --- generate element matrices, assemble in global matrices -
K=zeros(15); M=zeros(15);
[Ex,Ey]=coordxtr(Edof,Coord,Dof,2);
for i=1:2
    [k,m,c]=beam2d(Ex(i,:),Ey(i,:),epv);
    K=assem(Edof(i,:),K,k); M=assem(Edof(i,:),M,m);
end
for i=3:4
    [k,m,c]=beam2d(Ex(i,:),Ey(i,:),eph);
    K=assem(Edof(i,:),K,k); M=assem(Edof(i,:),M,m);
end
```

The finite element mesh is plotted, using the following commands

```
clf;
eldraw2(Ex,Ey,[1 2 2],Edof);
grid; title('2D Frame Structure');
pause;
```



*Finite element mesh*

## exd1

## Dynamic analysis

A standard procedure in dynamic analysis is eigenvalue analysis. This is accomplished by the following set of commands.

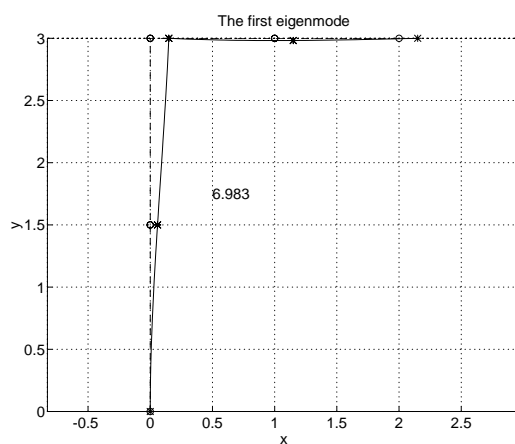
```
b=[1 2 3 14]';  
[La,Egv]=eigen(K,M,b);  
Freq=sqrt(La)/(2*pi);
```

Note that the boundary condition matrix, **b**, only lists the degrees-of-freedom that are zero. The results of these commands are the eigenvalues, stored in **La**, and the eigenvectors, stored in **Egv**. The corresponding frequencies in Hz are calculated and stored in the column matrix **Freq**.

```
Freq = [6.9826 43.0756 66.5772 162.7453 230.2709 295.6136  
        426.2271 697.7628 877.2765 955.9809 1751.3]T
```

The eigenvectors can be plotted by entering the commands below.

```
figure(1), clf, grid, title('The first eigenmode'),  
eldraw2(Ex,Ey,[2 3 1]);  
Edb=extract(Edof,Egv(:,1)); eldisp2(Ex,Ey,Edb,[1 2 2]);  
FreqText=num2str(Freq(1)); text(.5,1.75,FreqText);  
pause;
```



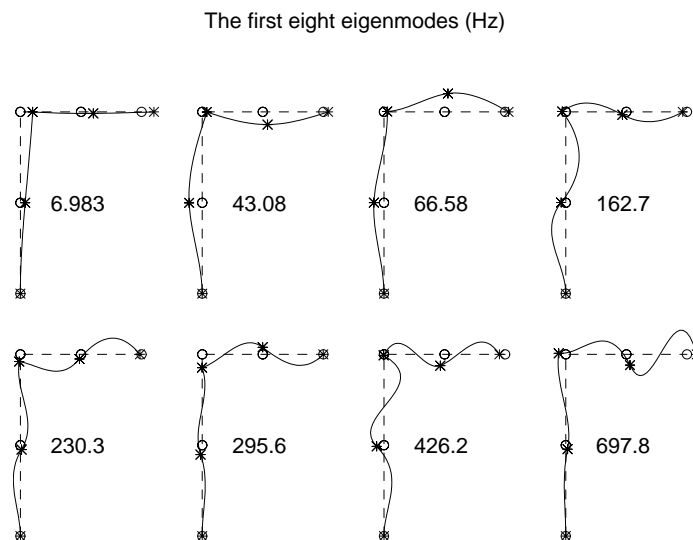
*The first eigenmode, 6.98 Hz*

## Dynamic analysis

exd1

An attractive way of displaying the eigenmodes is shown in the figure below. The result is accomplished by translating the different eigenmodes in the  $x$ -direction, see the Ext-matrix defined below, and in the  $y$ -direction, see the Eyt-matrix.

```
clf, axis('equal'), hold on, axis off
sfac=0.5;
title('The first eight eigenmodes (Hz)' )
for i=1:4;
    Ext=Ex+(i-1)*3;          eldraw2(Ext,Ey,[2 3 1]);
    Edb=extract(Edof,Egv(:,i));
    eldisp2(Ext,Ey,Edb,[1 2 2],sfac);
    FreqText=num2str(Freq(i)); text(3*(i-1)+.5,1.5,FreqText);
end;
Eyt=Ey-4;
for i=5:8;
    Ext=Ex+(i-5)*3;          eldraw2(Ext,Eyt,[2 3 1]);
    Edb=extract(Edof,Egv(:,i));
    eldisp2(Ext,Eyt,Edb,[1 2 2],sfac);
    FreqText=num2str(Freq(i)); text(3*(i-5)+.5,-2.5,FreqText);
end
```



*The first eight eigenmodes. Frequencies are given in Hz.*



**exd2****Dynamic analysis**

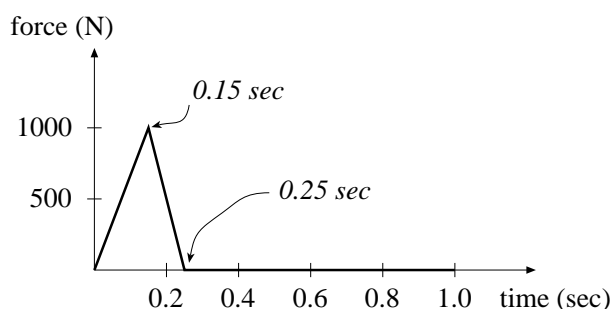
---

**Purpose:**

The frame structure defined in **exd1** is exposed in this example to a transient load. The structural response is determined by a time stepping procedure.

**Description:**

The structure is exposed to a transient load, impacting on the center of the vertical beam in horizontal direction, i.e. at the 4th degree-of-freedom. The time history of the load is shown below. The result shall be displayed as time history plots of the 4th degree-of-freedom and the 11th degree-of-freedom. At time  $t = 0$  the frame is at rest. The timestep is chosen as  $\Delta t = 0.001$  seconds and the integration is performed for  $T = 1.0$  second. At every 0.1 second the deformed shape of the whole structure shall be displayed.



*Time history of the impact load*

The load is generated using the **gfunc**-function. The time integration is performed by the **step2**-function. Because there is no damping present, the **C**-matrix is entered as **[]**.

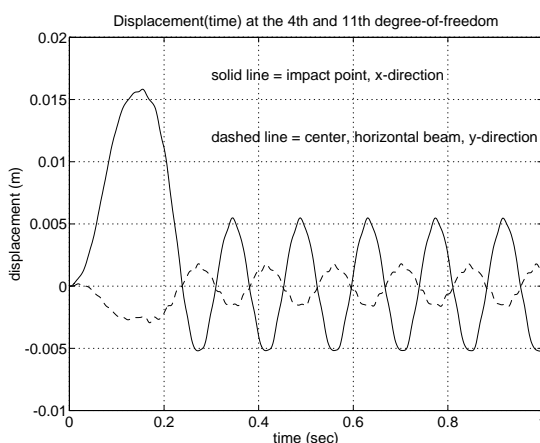
```
dt=0.005;    T=1;
% --- the load -----
G=[0 0; 0.15 1; 0.25 0; T 0];    [t,g]=gfunc(G,dt);
f=zeros(15, length(g));    f(4,:)=1000*g;
% --- boundary condition, initial condition -----
bc=[1 0; 2 0; 3 0; 14 0];
d0=zeros(15,1);    v0=zeros(15,1);
% --- output parameters -----
ntimes=[0.1:0.1:1];    nhist=[4 11];
% --- time integration parameters -----
ip=[dt T 0.25 0.5 10 2 ntimes nhist];
% --- time integration -----
k=sparse(K);    m=sparse(M);
[Dsnap,D,V,A]=step2(k,[],m,d0,v0,ip,f,bc);
```

The requested time history plots are generated by the following commands

## Dynamic analysis

exd2

```
figure(1), plot(t,D(1,:),'-',t,D(2,:), '--')
grid, xlabel('time (sec)'), ylabel('displacement (m)')
title('Displacement(time) for the 4th and 11th...
      ' degree-of-freedom')
text(0.3,0.009,'solid line = impact point, x-direction')
text(0.3,0.007,'dashed line = center, horizontal beam,...
      ' y-direction')
```



*Time history at DOF 4 and DOF 11.*

The deformed shapes at time increment 0.1 sec are stored in **Dsnap**. They are visualized by the following commands:

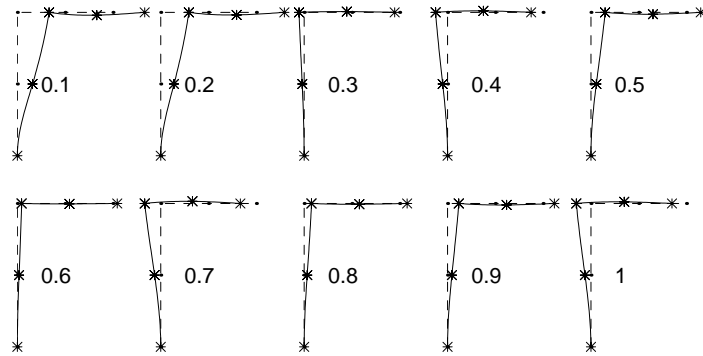
```
figure(2),clf, axis('equal'), hold on, axis off
sfac=25;
title('Snapshots (sec), magnification = 25');
for i=1:5;
    Ext=Ex+(i-1)*3;          eldraw2(Ext,Ey,[2 3 0]);
    Edb=extract(Edof,Dsnap(:,i));
    eldisp2(Ext,Ey,Edb,[1 2 2],sfac);
    Time=num2str(ntimes(i));  text(3*(i-1)+.5,1.5,Time);
end;

Eyt=Ey-4;
for i=6:10;
    Ext=Ex+(i-6)*3;          eldraw2(Ext,Eyt,[2 3 0]);
    Edb=extract(Edof,Dsnap(:,i));
    eldisp2(Ext,Eyt,Edb,[1 2 2],sfac);
    Time=num2str(ntimes(i));  text(3*(i-6)+.5,-2.5,Time);
end
```

exd2

Dynamic analysis

Snapshots (sec), magnification = 25

*Snapshots of the deformed geometry for every 0.1 sec.*

## Dynamic analysis

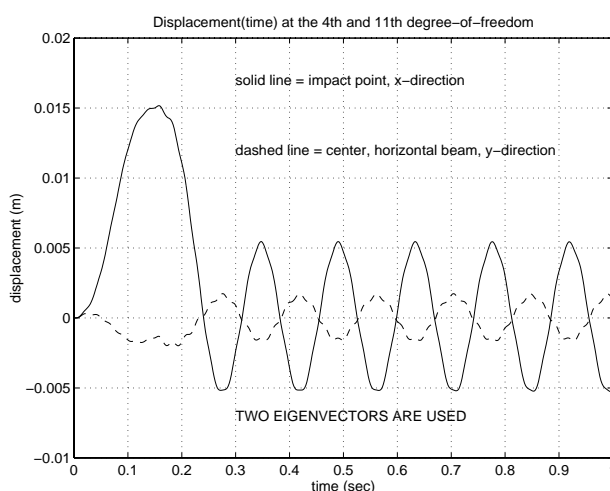
exd3

**Purpose:**

This example concerns reduced system analysis for the frame structure defined in exd1. Transient analysis on modal coordinates is performed for the reduced system.

**Description:**

In the previous example the transient analysis was based on the original finite element model. Transient analysis can also be employed on some type of reduced system, commonly a subset of the eigenvectors. The commands below pick out the first two eigenvectors for a subsequent time integration, see constant `nev`. The result in the figure below shall be compared to the result in exd2.



*Time history at DOF 4 and DOF 11 using two eigenvectors.*

```
dt=0.002;    T=1;    nev=2;
% --- the load -----
G=[0 0; 0.15 1; 0.25 0; T 0];    [t,g]=gfunc(G,dt);
f=zeros(15, length(g));    f(4,:)=9000*g;
fr=sparse([1:1:nev]' Egv(:,1:nev)']*f);
% --- reduced system matrices -----
kr=sparse(diag(diag(Egv(:,1:nev)')*K*Egv(:,1:nev))));
mr=sparse(diag(diag(Egv(:,1:nev)')*M*Egv(:,1:nev))));
% --- initial condition -----
dr0=zeros(nev,1);    vr0=zeros(nev,1);
% --- output parameters -----
ntimes=[0.1:0.1:1];    nhistr=[1:1:nev];
% --- time integration parameters -----
ip=[dt T 0.25 0.5 10 nev ntimes nhistr];
% --- time integration -----
[Dsnapr,Dr,Vr,Ar]=step2(kr,[],mr,dr0,vr0,ip,fr,[]);
% --- mapping back to original coordinate system -----
```

exd3

Dynamic analysis

---

```
DsnapR=Egv(:,1:nev)*Dsnapr;          DR=Egv(nhist,1:nev)*Dr;
% --- plot time history for two DOF:s -----
figure(1), plot(t,DR(1,:),'-',t,DR(2,:), '--')
axis([0    1.0000   -0.0100    0.0200])
grid, xlabel('time (sec)'), ylabel('displacement (m)')
title('Displacement(time) at the 4th and 11th...
      ' degree-of-freedom')
text(0.3,0.017,'solid line = impact point, x-direction')
text(0.3,0.012,'dashed line = center, horizontal beam,...
      ' y-direction')
text(0.3,-0.007,'2 EIGENVECTORS ARE USED')
```

## Dynamic analysis

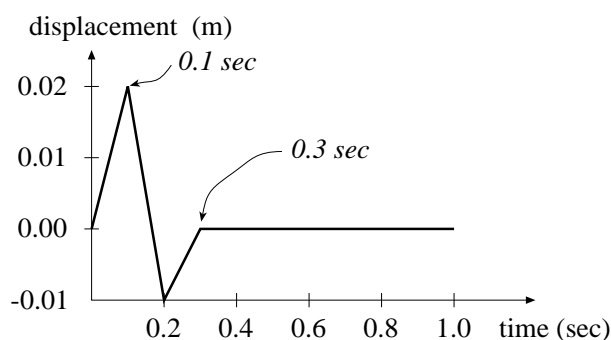
exd4

**Purpose:**

This example deals with a time varying boundary condition and time integration for the frame structure defined in `exd1`.

**Description:**

Suppose that the support of the vertical beam is moving in the horizontal direction. The commands below prepare the model for time integration. Note that the structure of the boundary condition matrix `bc` differs from the structure of the load matrix `f` defined in `exd2`.



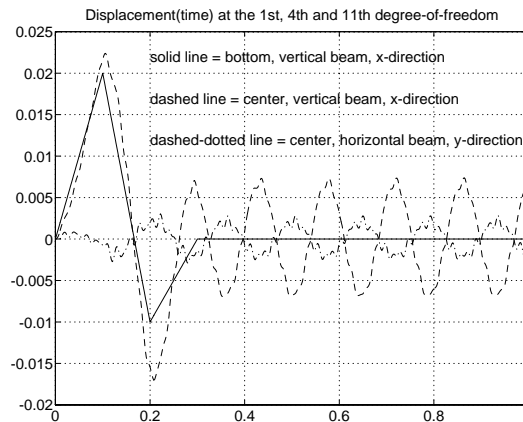
*Time dependent boundary condition at the support, DOF 1.*

```
dt=0.002;    T=1;
% --- boundary condition, initial condition -----
G=[0 0; 0.1 0.02; 0.2 -0.01; 0.3 0.0; T 0]; [t,g]=gfunc(G,dt);
bc=zeros(4, 1 + length(g));
bc(1,:)= [1 g]; bc(2,1)=2; bc(3,1)=3; bc(4,1)=14;
d0=zeros(15,1);          v0=zeros(15,1);
% --- output parameters -----
ntimes=[0.1:0.1:1];      nhist=[1 4 11];
% --- time integration parameters -----
ip=[dt T 0.25 0.5 10 3 ntimes nhist];
% --- time integration -----
k=sparse(K);              m=sparse(M);
[Dsnap,D,V,A]=step2(k,[],m,d0,v0,ip,[],bc);
% --- plot time history for two DOF:s -----
figure(1), plot(t,D(1,:),'-',t,D(2,:), '--',t,D(3,:), '-. ');
grid, xlabel('time (sec)'), ylabel('displacement (m)')
title('Displacement(time) at the 1st, 4th and 11th...
      ' degree-of-freedom')
text(0.2,0.022,'solid line = bottom, vertical beam,'...
      ' x-direction')
text(0.2,0.017,'dashed line = center, vertical beam,'...
      ' x-direction')
```

exd4

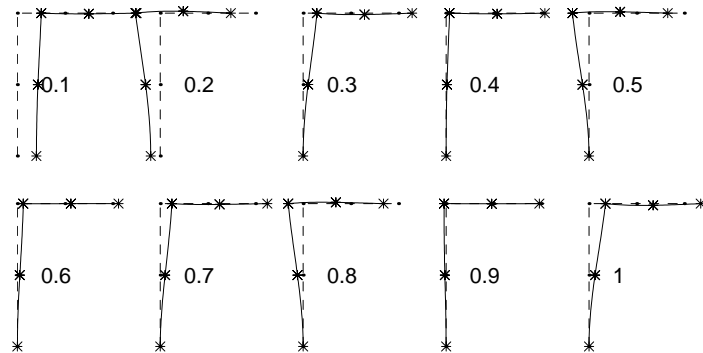
Dynamic analysis

```
text(0.2,0.012,'dashed-dotted line = center,...  
      ' horizontal beam, y-direction')  
% --- plot displacement for some time increments -----  
figure(2),clf, axis('equal'), hold on, axis off  
sfac=20;  
title('Snapshots (sec), magnification = 20'); for i=1:5;  
    Ext=Ex+(i-1)*3;          eldraw2(Ext,Ey,[2 3 0]);  
    Edb=extract(Edof,Dsnap(:,i));  
    eldisp2(Ext,Ey,Edb,[1 2 2],sfac);  
    Time=num2str(ntimes(i)); text(3*(i-1)+.5,1.5,Time);  
end;  
Eyt=Ey-4;  
for i=6:10;  
    Ext=Ex+(i-6)*3;          eldraw2(Ext,Eyt,[2 3 0]);  
    Edb=extract(Edof,Dsnap(:,i));  
    eldisp2(Ext,Eyt,Edb,[1 2 2],sfac);  
    Time=num2str(ntimes(i)); text(3*(i-6)+.5,-2.5,Time);  
end
```



*Time history at DOF 1, DOF 4 and DOF 11.*

Snapshots (sec), magnification = 20

*Snapshots of the deformed geometry for every 0.1 sec.*





---

## 9.5 Nonlinear analysis

This section illustrates some nonlinear finite element calculations.

Nonlinear analysis	
exN1	Second order theory analysis of a frame
exN2	Buckling analysis of a frame

**Note:** The examples listed above are supplied as .m-files under the directory `examples`. The example files are named according to the table.

exn1

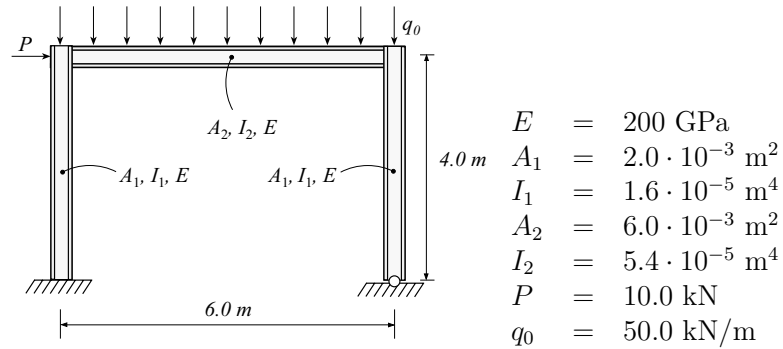
Nonlinear analysis

**Purpose:**

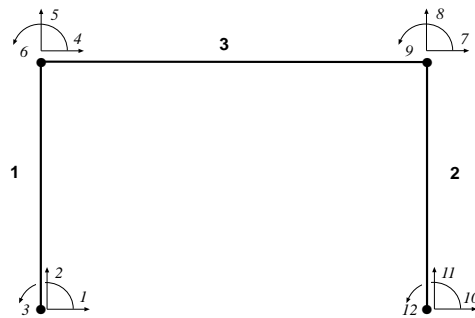
Analysis of a plane frame using second order theory.

**Description:**

The frame of **exs6** is analysed again, but it is now subjected to a load five times larger than in **exs6**. Second order theory is used.



The element model consisting of three beam elements and twelve degrees of freedom is repeated here.



The following .m-file defines the finite element model.

```
% ----- Topology -----

Edof=[1  4  5  6  1  2  3 ;
       2  7  8  9 10 11 12;
       3  4  5  6  7  8  9];
```

## Nonlinear analysis

exn1

---

```
% ----- Element properties and global coordinates -----
```

```
E=200e9;
A1=2e-3;      A2=6e-3;
I1=1.6e-5;    I2=5.4e-5;
ep1=[E A1 I1]; ep3=[E A2 I2];
eq3=[-50e3];
```

```
Ex=[0 0;6 6;0 6];  Ey=[4 0;4 0;4 4];
```

The beam element function of the second order theory `beam2g` requires a normal force as input variable. In the first iteration this normal force is chosen to zero. This means that the first iteration is equivalent to a linear first order analysis using `beam2e`. Since the normal forces are not known initially, an iterative procedure has to be applied, where the normal forces  $N$  are updated according to the results of the former iteration. The iterations continue until the difference in normal force of the two last iteration steps is less than an accepted error `eps`,  $(N-N_0)/N_0 < \text{eps}$ . The small value given to the initial normal force  $N(1)$  is to avoid division by zero in the second convergence check. If  $N$  does not converge in 20 steps the analysis is interrupted.

```
% ----- Initial values for the iteration -----
```

```
eps=0.0001;      % Error norm
N=[0.01 0 0];    % Initial normal forces
N0=[1 1 1];      % Normal forces of the initial former iteration
n=0;             % Iteration counter
```

```
% ----- Iteration procedure -----
```

```
while(abs((N(1)-N0(1))/N0(1)) > eps)
    n=n+1;

    K=zeros(12,12);
    f=zeros(12,1);
    f(4)=10e3;

    Ke1=beam2g(Ex(1,:),Ey(1,:),ep1,N(1));
    Ke2=beam2g(Ex(2,:),Ey(2,:),ep1,N(2));
    [Ke3,fe3]=beam2g(Ex(3,:),Ey(3,:),ep3,N(3),eq3);

    K=assem(Edof(1,:),K,Ke1);
    K=assem(Edof(2,:),K,Ke2);
    [K,f]=assem(Edof(3,:),K,Ke3,f,fe3);
```

**exn1**

**Nonlinear analysis**

---

```

bc=[1 0;2 0;3 0;10 0;11 0];
[a,r]=solveq(K,f,bc)

Ed=extract(Edof,a);

es1=beam2gs(Ex(1,:),Ey(1,:),ep1,Ed(1,:),N(1))
es2=beam2gs(Ex(2,:),Ey(2,:),ep1,Ed(2,:),N(2))
es3=beam2gs(Ex(3,:),Ey(3,:),ep3,Ed(3,:),N(3),eq3)

N0=N;
N=[es1(1,1) es2(1,1) es3(1,1)];

if (n>20)
    disp('The solution doesn''t converge')
    return
end
end
end

```

Displacements and element forces from the linear elastic analysis and from the second order theory analysis respectively:

a =	a =
0	0
0	0
0	0
0.0377	0.0452
-0.0014	-0.0014
-0.0269	-0.0281
0.0376	0.0451
-0.0016	-0.0016
0.0233	0.0239
0	0
0	0
-0.0258	-0.0296

**Nonlinear analysis****exn1**

es1 =

1.0e+005 \*

-1.4370	0.0963	0.4076
-1.4370	0.0963	0.0223

es1 =

1.0e+005 \*

-1.4241	0.0817	0.3428
-1.4241	0.0817	0.0803

es2 =

1.0e+005 \*

-1.5630	-0.1963	-0.7854
-1.5630	-0.1963	0.0000

es2 =

1.0e+005 \*

-1.5759	-0.1817	-0.7980
-1.5759	-0.1817	-0.0000

es3 =

1.0e+005 \*

-0.1963	-1.4370	-0.4076
-0.1963	1.5630	-0.7854

es3 =

1.0e+005 \*

-0.1817	-1.4241	-0.3428
-0.1817	1.5759	-0.7980

Using the second order theory, the horizontal displacement of the upper left corner of the frame increases from 37.7 to 45.2 mm. The moment in the lower left corner increases from 2.2 kNm to 8.0 kNm.

exn2

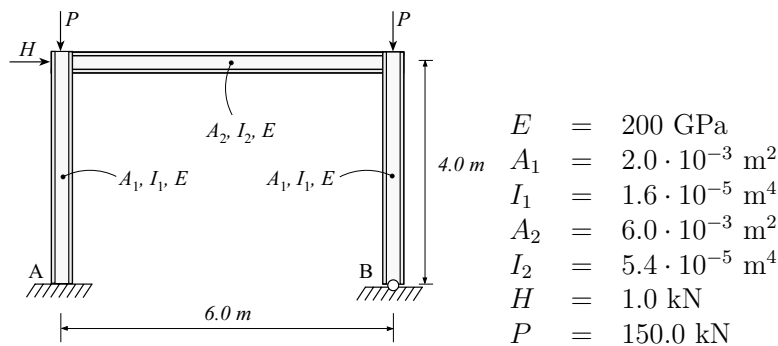
Nonlinear analysis

**Purpose:**

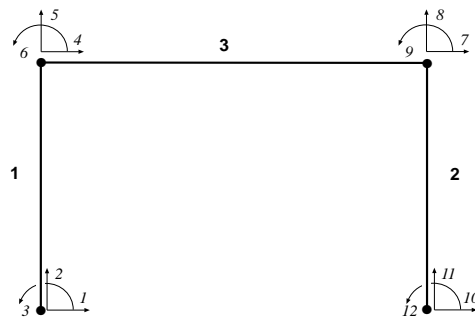
Buckling analysis of a plane frame.

**Description:**

The frame of exn1 is in this example analysed with respect to security against buckling for a case when all loads are increased proportionally. The initial load distribution is increased by a loading factor **alpha** until buckling occurs, i.e. the determinant of the stiffness matrix **K** passes zero, or the solution does not converge. For each value of **alpha** a second order theory calculation of type **exn1** is performed. The horizontal displacement  $a_4$  and the moment  $M_A$  are plotted against **alpha**. The shape of the buckling mode is also plotted using the last computed displacement vector before buckling occurs.



The element model consists of three beam elements and twelve degrees of freedom.



The following .m-file defines the finite element model.

```
% ----- Topology -----

Edof=[1  4  5  6  1  2  3 ;
      2  7  8  9 10 11 12;
      3  4  5  6  7  8  9];
```

## Nonlinear analysis

exn2

```
% ----- Element properties and global coordinates -----

E=200e9;
A1=2e-3;    A2=6e-3;
I1=1.6e-5;  I2=5.4e-5;
ep1=[E A1 I1]; ep3=[E A2 I2];

Ex=[0 0;6 6;0 6]; Ey=[4 0;4 0;4 4];

% ----- Initial loads -----

f0=zeros(12,1);
f0(4)=1e3;    f0(5)=-150e3;    f0(8)=-150e3;

% ----- Increase loads until det(K)=0 -----

j=0;
for alpha=1:0.2:10
    j=j+1;
    N=[0.01 0 0];
    N0=[1 1 1];

% ----- Iteration for convergence -----

    eps=0.0001;
    n=0;
    while(abs((N(1)-N0(1))/N0(1))>eps)
        n=n+1;

        K=zeros(12,12);
        f=f0*alpha;
        Ke1=beam2g(Ex(1,:),Ey(1,:),ep1,N(1));
        Ke2=beam2g(Ex(2,:),Ey(2,:),ep1,N(2));
        Ke3=beam2g(Ex(3,:),Ey(3,:),ep3,N(3));

        K=assem(Edof(1,:),K,Ke1);
        K=assem(Edof(2,:),K,Ke2);
        K=assem(Edof(3,:),K,Ke3);

        bc=[1 0;2 0;3 0;10 0;11 0];
        [a,r]=solveq(K,f,bc);

        Ed=extract(Edof,a);
```



exn2

Nonlinear analysis

---

```
es1=beam2gs(Ex(1,:),Ey(1,:),ep1,Ed(1,:),N(1));
es2=beam2gs(Ex(2,:),Ey(2,:),ep1,Ed(2,:),N(2));
es3=beam2gs(Ex(3,:),Ey(3,:),ep3,Ed(3,:),N(3));

NO=N;
N=[es1(1,1),es2(1,1),es3(1,1)];

if(n>20)
    disp(['Alpha= ',num2str(alpha), ...
        ': The solution doesn't converge.'])
    break
end
end

% ----- Check determinant for buckling -----

Kred=red(K,bc(:,1));
if (det(Kred)<=0)
    disp(['Alpha= ',num2str(alpha), ...
        ': Determinant <= 0, buckling load passed.'])
    break
end
if(n>20)
    break
end
disp(['Alpha= ',num2str(alpha),' is OK! ', int2str(n), ...
    ' iterations are performed.'])
disp([' '])

% ----- Save values for plotting of results -----

deform(j)=a(4);
M(j)=r(3);
loadfact(j)=alpha;
bmode=a;
end
```

The following text strings are produced by the .m-file.

```
Alpha= 1 is OK! 3 iterations are performed.
Alpha= 1.2 is OK! 3 iterations are performed.
Alpha= 1.4 is OK! 3 iterations are performed.
Alpha= 1.6 is OK! 3 iterations are performed.
.
.
.
Alpha= 6 is OK! 3 iterations are performed.
Alpha= 6.2 is OK! 4 iterations are performed.
Alpha= 6.4 is OK! 4 iterations are performed.
Alpha= 6.6 is OK! 5 iterations are performed.
Alpha= 6.8: The solution doesn't converge.
Alpha= 6.8: Determinant <= 0, buckling load passed
```

The requested plots of the horizontal displacement, the moment  $M_A$ , and the shape of the buckling mode are generated by the following commands

```
% ----- Plot results -----

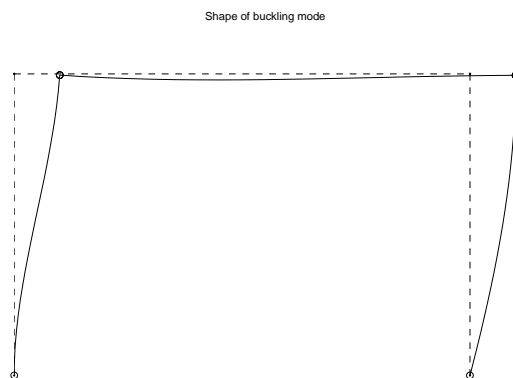
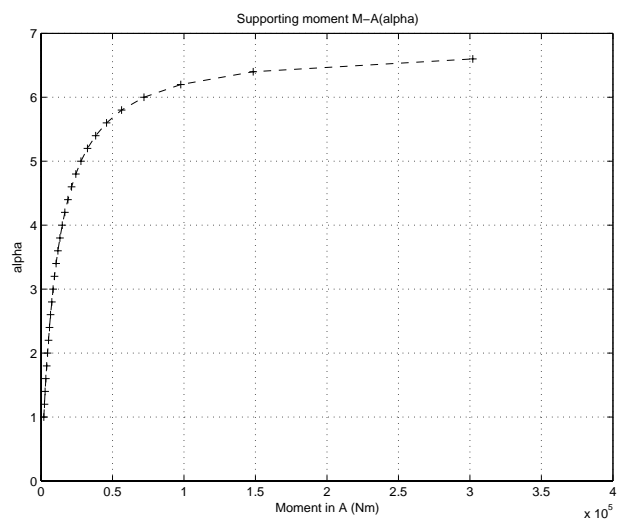
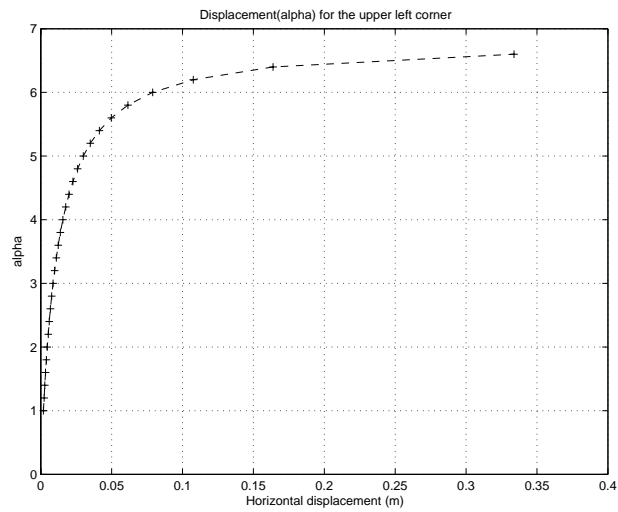
figure(1), clf
plot(deform(:),loadfact(:),'+',deform(:),loadfact(:),'--')
axis([0 0.4 0 7]), grid
xlabel('Horizontal displacement (m)'), ylabel('alpha')
title('Displacement(alpha) for the upper left corner')

figure(2), clf
plot(M(:),loadfact(:),'+',M(:),loadfact(:),'--')
axis([0 0.4e6 0 7]), grid
xlabel('Moment in A (Nm)'), ylabel('alpha')
title('Supporting moment M-A(alpha)')

figure(3), clf, axis off
eldraw2(Ex,Ey,[2,3,0]);
Ed1=extract(Edof,bmode);
sfac=eldisp2(Ex,Ey,Ed1);
eldisp2(Ex,Ey,Ed1,[1,1,1],sfac);
title('Shape of buckling mode')
```

exn2

Nonlinear analysis




---

EXAMPLES

9.5 – 10

## Index

abs .....	3 – 6	figure .....	8 – 11
assem .....	6.2 – 2	fill .....	8 – 12
axis .....	8 – 2	flw2i4e .....	5.4 – 8
bar2e .....	5.3 – 2	flw2i4s .....	5.4 – 10
bar2g .....	5.3 – 3	flw2i8e .....	5.4 – 11
bar2s .....	5.3 – 5	flw2i8s .....	5.4 – 13
bar3e .....	5.3 – 6	flw2qe .....	5.4 – 6
bar3s .....	5.3 – 7	flw2qs .....	5.4 – 7
beam2d .....	5.6 – 20	flw2te .....	5.4 – 3
beam2ds .....	5.6 – 22	flw2ts .....	5.4 – 5
beam2e .....	5.6 – 2	flw3i8e .....	5.4 – 14
beam2g .....	5.6 – 15	flw3i8s .....	5.4 – 16
beam2gs .....	5.6 – 18	for .....	7 – 3
beam2s .....	5.6 – 5	format .....	2 – 6
beam2t .....	5.6 – 7	freqresp .....	6.3 – 5
beam2ts .....	5.6 – 9	full .....	3 – 9
beam2w .....	5.6 – 11	function .....	7 – 5
beam2ws .....	5.6 – 13	gfunc .....	6.3 – 6
beam3e .....	5.6 – 24	grid .....	8 – 13
beam3s .....	5.6 – 27	help .....	2 – 7
clear .....	2 – 2	hold .....	8 – 14
clf .....	8 – 3	hooke .....	4 – 2
coordxtr .....	6.2 – 3	if .....	7 – 2
det .....	3 – 7	ifft .....	6.3 – 7
diag .....	3 – 8	insert .....	6.2 – 8
diary .....	2 – 3	inv .....	3 – 10
disp .....	2 – 4	length .....	3 – 11
dmises .....	4 – 4	load .....	2 – 8
dyna2f .....	6.3 – 3	max .....	3 – 12
dyna2 .....	6.3 – 2	min .....	3 – 13
echo .....	2 – 5	mises .....	4 – 3
eigen .....	6.2 – 5	ones .....	3 – 14
eldia2 .....	8 – 4	plani4e .....	5.5 – 20
eldisp2 .....	8 – 6	plani4f .....	5.5 – 25
eldraw2 .....	8 – 7	plani4s .....	5.5 – 23
elflux2 .....	8 – 8	plani8e .....	5.5 – 26
eliso2 .....	8 – 9	plani8f .....	5.5 – 31
elprinc2 .....	8 – 10	plani8s .....	5.5 – 29
extract .....	6.2 – 6	planqe .....	5.5 – 9
fft .....	6.3 – 4	planqs .....	5.5 – 11

## Index

planre .....	5.5 – 12	whos .....	2 – 13
planrs .....	5.5 – 15	xlabel .....	8 – 21
plantce .....	5.5 – 16	ylabel .....	8 – 21
plantcs .....	5.5 – 19	zeros .....	3 – 21
plante .....	5.5 – 4	zlabel .....	8 – 21
plantf .....	5.5 – 8		
plants .....	5.5 – 7		
platre .....	5.7 – 2		
platrS .....	5.7 – 5		
plot .....	8 – 15		
pltscalb2 .....	8 – 16		
print .....	8 – 17		
quit .....	2 – 9		
red .....	3 – 15		
ritz .....	6.3 – 8		
save .....	2 – 10		
scalfact2 .....	8 – 18		
script .....	7 – 6		
size .....	3 – 16		
sol8e .....	5.5 – 32		
sol8f .....	5.5 – 37		
sol8s .....	5.5 – 35		
solveq .....	6.2 – 9		
sparse .....	3 – 17		
spectra .....	6.3 – 9		
spring1e .....	5.2 – 4		
spring1s .....	5.2 – 5		
spy .....	3 – 18		
sqrt .....	3 – 19		
statcon .....	6.2 – 10		
step1 .....	6.3 – 10		
step2 .....	6.3 – 12		
sum .....	3 – 20		
sweep .....	6.3 – 14		
text .....	8 – 19		
title .....	8 – 20		
type .....	2 – 11		
what .....	2 – 12		
while .....	7 – 4		
who .....	2 – 13		