Laborübung FEM mit Matlab

Dr. Martin Kraska

28. März 2011
Sommersemester 2011 an der Fachhochschule Brandenburg

Inhaltsverzeichnis

Über dieses Skript

Das vorliegende Skript wurde mit LaTeX unter Verwendung des grafischen Editors LyX gesetzt. Unter Windows kommt der LyX-Installer mit der TeX-Distribution MikTeX. Für die Verwendung von LaTeX sprechen:

- Gute Satzqualität von mathematischen Formeln
- Einfache Numerierung und Verweise auf Formeln
- Einfache Literatur- und Zitatverwaltung mit Bibtex
- Konsequente Formatierung anhand der Dokumentstruktur, nicht anhand von direkten Vorgaben für das Aussehen
- Änderungsmanagement mit jedem Quellcodeverwaltungssystem (zum Beispiel Subversion)
- ausschließliche Benutzung kostenloser (legaler) Software

Nachteile bei der Verwendung von Latex (im Vergleich zu Word oder Powerpoint) sind

- Aufwand für den Einstieg ist höher
- weniger intuitive Plazierung und Bearbeitung von Grafiken

Typografische Konventionen

Im Interesse einer guten Lesbarkeit, eines einheitlichen Aussehens und in Anlehnung an einschlägige Normen werden bestimmte Formatierungen angestrebt.

In Formeln:

- Lateinische Buchstaben als Formelzeichen werden kursiv gesetzt: a, D
- Maßeinheiten und sprechende Indizes werden aufrecht gesetzt: $m_{\mathrm{gesamt}} = 100\,\mathrm{kg}$
- Maßeinheiten und Maßzahl werden durch ein kleines Leerzeichen getrennt: $200\,\mathrm{mm}$
- Vektoren werden durch unterstrichene Kleinbuchstaben dargestellt: $\underline{v}$

- Matrizen werden durch unterstrichene Großbuchstaben dargestellt: $\underline{M}$

In Bedienungsanleitungen:

- Objekte in Bedienungsoberflächen (Menüs, Fenster, Boxen) werden **fett** gesetzt
- Text in der Eingabezeile wird in `Schreibmaschinenschrift` gesetzt

In normalem Text:

- Englische Begriffe oder Dateinamen werden *kursiv* gesetzt

1. Einführung

1.1. Ziel des Kurses

Die Laborübungen mit Matlab sind Teil des Kurses „Finite Elemente Methode - Grundlagen und Anwendungen". Dabei sollen die Studenten mathematische Zusammenhänge aus der Vorlesung mit Matlab spielerisch nachvollziehen. An einfachen Stabfachwerken und ebenen Scheibenproblemen werden Begriffe veranschaulicht, die dem Anwender beim Umgang mit professioneller FEM-Software für die Lösung mechanischer Probleme begegnen. Die verwendeten Begriffe und Formeln sind im Skript „FEM Grundlagen" erläutert.

Matlab steht für Matrix Laboratory. Das ist eine Software, mit der man zunächst einmal interaktiv mit Matrizen rechnen kann. Dabei ist es zunächst egal, was das für Matrizen sind. Matrizen haben vielfältige Anwendungen und Interpretationen, Beispiele sind:

- Messdaten. Jede Messung entspricht einer Zeile, die Messzeit und die einzelnen Kanäle bilden die Spalten. Matrizenoperationen dienen dann der Berechnung abgeleiteter Größen, der Datenfilterung usw.
- Pixelbilder. Jeder Eintrag entspricht einem Farbwert, die Zeilen und Spalten bilden das zweidimensionale Bild. Matrizenoperationen sind dann bestimmte Bildauswertungen, z.B. Kantenfindung, Weichzeichnung, Schärfung, Drehungen usw.
- Eigenschaften linearer mechanischer oder elektrischer Systeme wie Steifigkeit, Masse, Impedanz.

Matlab ist durch die mächtigen Funktionen für den Umgang mit Daten und zur Erzeugung von Bildern eine sehr leistungsfähige Programmiersprache. Sie bietet dem Ingenieur wesentlich mehr als die klassische Programmierung mit C, wie man sie aus dem Informatikkurs kennen mag. Der Kurs soll den Einstieg vermitteln und ein wenig Appetit machen.

Sollte einmal Matlab nicht zur Verfügung stehen, kann man das freie Programm Octave verwenden. Es ist mit Matlab weitgehend kompatibel, bietet aber deutlich weniger Bedienkomfort.

1.2. Alternativen zu Matlab

Es gibt einen freien Matlab-Clone namens **Octave**. Das ist eine Open Source Entwicklung mit dem Anspruch, Matlab-kompatibel zu sein, erreicht aber nicht dessen Funktionsumfang

und ist auch wesentlich weniger komfortabel zu bedienen. Für Studenten und Wissenschaftler mit schmalem Budget ist es aber eine ernst zu nehmende Alternative. Da zunächst nicht klar war, ob Matlab an der FH Brandenburg für die Laborübungen zur Verfügung steht, wurden die Beispiele zunächst mit Octave entwickelt und sollten unter beiden Systemen unverändert funktionieren.

Ein weiteres vergleichbares Produkt ist das ebenfalls frei verfügbare **Scilab**. Dieses kommt im Bedienkomfort näher an Matlab heran, ist aber wesentlich weniger Matlab-kompatibel als Octave.

Matlab, Octave und Scilab sind vor allem für das Rechnen mit Matrizen optimiert. Eine andere Klasse von Mathematikprogrammen sind die sogenannten Computer-Algebra-Systeme (CAS), die auch symbolisch rechnen können. Beispiele für solche CAS sind **Mathematica**, **Maxima**, oder **Maple**, sowie **Scientific Workplace**.

Eine dritte Variante ist **MathCad**, ein Programm dass eigentlich für den Ingenieur zugeschnitten ist und zum Beispiel eine sehr intuitive Einheitenverwaltung bietet.

Nicht verschwiegen werden soll, dass man auch mit **Excel** rechnen könnte, wenn man mit Visual Basic for Applications (VBA) programmieren kann. Hier gibt es aber spürbare Beschränkungen für die Größe der handhabbaren Datenmengen. Das fällt besonders bei der Verarbeitung von Messdaten (Zeitreihen) auf.

2. Funktionen in Octave/Matlab

Die folgende Übersicht orientiert sich an den Bedürfnissen des FEM-Kurses und enthält nur Funktionen, dies sowohl in Octave als auch in Matlab unterstützt werden.

2.1. Vektor- und Matrixoperationen

Operation	Befehl in Matlab
Matrix erzeugen	`A=[1 2 3; 4 5 6; 7 8 9]`
$n \times n$ Einheitsmatrix erzeugen	`I=eye(n)`
$m \times n$ Nullmatrix erzeugen	`M=zeros(m,n)`
Spaltenvektor erzeugen	`a=[1;2;3]`
Zeilenvektor erzeugen	`b=[1 2 3]`
Matrixgröße	`size(A)`
Zeilenzahl	`size(A,1)`
Spaltenzahl	`size(A,2)`
Transponierte Matrix $\underline{\mathrm{A}}^T$	`A'`
Summe $\underline{\mathrm{A}} + \underline{\mathrm{B}}$	`A+B`
Produkt mit Skalar $\underline{\mathrm{A}}q$	`A*q`
Matrixprodukt $\underline{\mathrm{AB}}$	`A*B`
Inverse $\underline{\mathrm{A}}^{-1}$	`inv(A), A^-1`
Lösung von $\underline{\mathrm{Ax}} = \underline{\mathrm{b}}$	`x = A\b, x = A^-1*b`
Determinante	`det(A)`

Tabelle 1: Vektor- und Matrixoperationen

Matrizen oder Vektoren können nebeneinander oder übereinander zu neuen Matrizen oder Vektoren zusammengesetzt werden, genau wie des mit einfachen Zahlen geht (die intern als 1×1 Matrizen betrachtet werden). Die Größen müssen allerdings passen, so dass sich ein rechteckiges Gebiet ergibt.

`[A B]` stellt die Matrizen A und B nebeinander in eine neue Matrix.

`[A;B]` stellt die Matrizen A und B übereinander in eine neue Matrix.

2.2. Bereiche und Zugriff auf Einträge

Matrixelemente können mit Skalaren oder Vektoren indiziert werden. Skalare Indizes greifen ein einzelnes Element aus der Matrix heraus. Vektoren greifen einen Satz von Zeilen oder Spalten heraus. Dieser Satz muss nicht zusammenhängend sein.

Innerhalb der Indexklammer einer Matrix kann das Schlüsselwort `end` verwendet werden. Dieses bezeichnet den höchsten gültigen Index an der jeweiligen Stelle. Die folgenden Zugriffe auf einen Spaltenvektor der Länge n sind gleichwertig

```
v
v(:)
v(1:n)
v(1:end)
v(:,1)
v(1:n,1)
v(1:end,1)
```

Operation	Befehl in Matlab
Bereich (Indexvektor) $i = 1 \dots n$	`i=1:n`
Element A_{ij}	`A(i,j)`
Zeile i der Matrix A	`A(i,:)`
	`A(i,1:end)`
Spalte j der Matrix A	`A(:,j)`
	`A(1:end,j)`
3. und 4. Spalte der 1. und 3. Zeile	`A([1 3],[3 4])`
14. Element bei spaltenweise fortlaufender Zählung (*linear index*)	`A(14)`

Tabelle 2: Zugriff auf Matrixelemente

Sehr wichtig bei der schleifenlosen Programmierung ist der Befehl

```
reshape(A,m,n)
```

Er wandelt die Matrix A in eine $m \times n$ Matrix um. Dabei werden zunächst alle Spalten hintereinander in einen Spaltenvektor gelegt und dann n neue Spalten der Länge m daraus entnommen. Will man dies zeilenweise machen, kann man die Matrix vorher transponieren.

2.3. Datenlesen und -schreiben, Anzeigen

Matrizen können in Textfiles mit Leerzeichen als Spaltentrenner gespeichert und gelesen werden.

Operation	Befehl in Matlab
Matrix A schreiben	`save -ascii filename A`
Matrix A lesen	`A=load(filename)`
Matrix A als Tabelle anzeigen	`disp(A)`
Matrix A grafisch anzeigen	`imagesc(A)`
Belegung der Matrix A grafisch anzeigen	`spy(A)`

Tabelle 3: Befehle für Speichern, Lesen und Anzeige von Matrizen

2.4. Grafik

```
plot(X,Y,format)
```

Die Spalten von Y werden über den Spalten von X dargestellt. Dieser Befehl wird für die Darstellung von Fachwerkstäben oder Elementumrissen bei Scheibenelementen verwendet. Das Argument `format` definiert das Linienformat mit den Merkmalen

Linienart		Farbe		Symbol	
'-'	**durchgehend**	'r'	rot	'+'	Plus
'--'	Strich	'g'	grün	'o'	Kreis
':'	punktiert	'b'	**blau**	'*'	Stern
'-.'	Strichpunkt	'c'	cyan	'.'	Punkt
		'm'	magenta	'x'	Kreuz
		'y'	gelb	's'	Quadrat □
		'k'	schwarz	'd'	Quadrat ◇
		'w'	weiß	'^'	Dreieck △
				'v'	Dreieck ▽
				'>'	Dreieck ▷
				'<'	Dreieck ◁

Tabelle 4: Zeichen für das Linienformat in Grafikbefehlen. Fett: Standardwerte.

```
axis off equal
```

Gleicher Maßstab für x- und y-Achsen und Ausblenden der Achsen

```
contourf(X,Y,Z)
```

Die Argumente sind Matrizen, die ein Gitter definieren, dessen z-Werte als Farbverlauf dargestellt werden.

```
quiver(X,Y,U,V)
```

Pfeile, deren Anfangspunkte durch `X` und `Y` und deren Länge durch `U` und `V` definiert sind. Damit könnte man äußere Lasten, Randbedingungen oder Eigenspannungen darstellen, wie beispielsweise in Abbildung 3.

2.5. Zeichenketten

Zeichenketten sind Zeilenvektoren aus Buchstaben.

Operation	Befehl in Matlab
Zeichenkette erzeugen	`A='ein Text'`
Zahl in Zeichenkette umwandeln	`A=num2str(n)`
Zeichenketten nebeneinander anfügen	`C=[A B]`

Tabelle 5: Zeichenkettenoperationen

3. FEM für Stabfachwerke

3.1. Systembeschreibung

In der FEM wird die Geometrie durch Elemente und Knoten beschrieben. An Knoten sind erklärt:

- Knotennummer
- Anfangslage, beschrieben durch die Knotenkoordinaten
- Freiheitsgrade (Bewegungsmöglichkeiten)
- Bindungen
- Äußere Kräfte

Die Struktur wird in Elemente aufgeteilt, das können je nach Art der Struktur Linienelemente (Stäbe, Balken), Flächenelemente (Scheiben, Schalen) oder Volumenelemente sein.

Bei Stabfachwerken wird jeder Stab durch ein Element dargestellt, welches 2 Knoten verbindet. Jedes Element ist gekennzeichnet durch

- Stabnummer
- Knotennummer am Stabanfang
- Knotennummer am Stabende
- Querschnittsfläche
- Elastizitätsmodul
- Dichte (wenn Eigengewicht oder Schwingungen betrachtet werden sollen)

Dichte, Querschnittsfläche und Elastizitätsmodul sind in der Regel für viele Stäbe eines Fachwerks identisch, daher ist es üblich, Gruppen zu definieren, die den Stäben durch Nummern zugeordnet werden.

Wir definieren das System über die folgende Tabellen

Knotentabelle Knotennummern mit den zugehörigen Koordinaten. Die Spalten bedeuten:

1. Knotennummer
2. x-Koordinate
3. y-Koordinate

Elementtabelle Stabnummern mit der Gruppennummer und den Knotennummern für Anfang und Ende

1. Stabnummer (Elementnummer)
2. Eigenschaftsnummer
3. Knotennummer Stabanfang
4. Knotennummer Stabende

Eigenschaftstabelle Materialdaten für die Stabgruppen (Querschnittsfläche, E-Modul, Dichte)

1. Materialgruppennummer
2. Querschnittsfläche
3. E-Modul
4. Dichte

Lasttabelle Knotennummern mit zugehörigen Lastkomponenten. Nur die Knoten mit Lasten müssen eingetragen werden. Sollen keine äußeren Lasten aufgebracht werden, ist die Lasttabelle trotzdem zu definieren (sie ist dann leer, Wert: `[]`.

1. Knotennummer
2. Koordinate (1 entspricht x, 2 entspricht y)
3. Zahlenwert der Komponente

Bindungstabelle Knotennummern mit zugehörigen Bindungen (Verschiebungsrandbedingungen). Nur die Knoten mit Bindungen müssen eingetragen werden.

1. Knotennummer
2. Koordinate (1 entspricht x, 2 entspricht y)
3. Vorgeschriebene Verschiebung (normalerweise 0, es können aber auch von Null verschiedene Verschiebungen aufgeprägt werden)

Beispiel

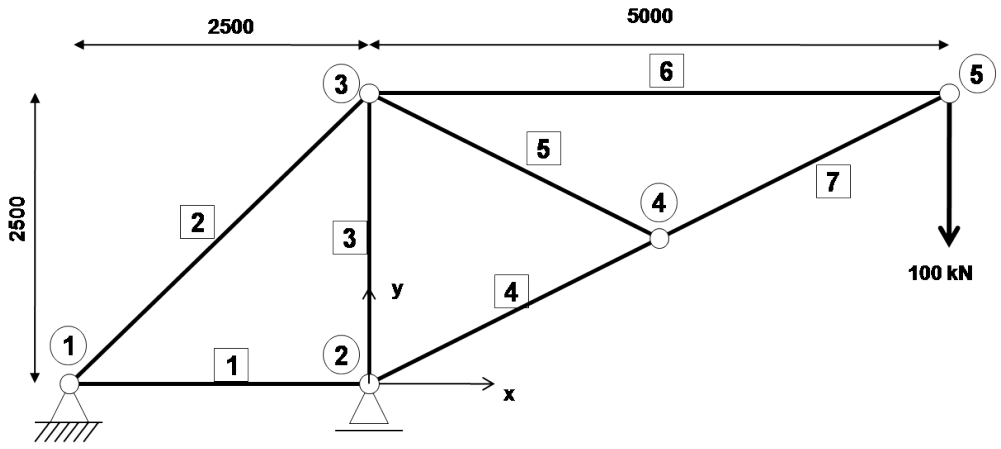


Abbildung 1: Fachwerk mit Knoten- und Stabnummern

Listing 1: Fachwerk1.m

```
% Stabfachwerk Beispiel
name='Fachwerk 1';
% Knoten:
% Nr x y
Knoten=[...
1 -2500 0;...
2 0 0;...
3 0 2500;...
4 2500 1250;...
5 5000 2500];
% Nr Gruppennummer Anfangsknoten Endknoten
```

```
Element=[...
1 1 1 2;...
2 1 1 3;...
3 1 2 3;...
4 1 2 4;...
5 1 3 4;...
6 1 3 5;...
7 1 4 5];
% Nr A E rho
Eigenschaft=[1 1000 210000 7.85e-9];
% Knoten Richtung Wert
Last=[5 2 -100000];
% Knoten Richtung Wert
Bindung=[...
1 1 0;...
1 2 0;...
2 2 0];
% Anzeigesteuerung
Uscale=10;
Fscale=0.01;
```

3.2. Das Programm StabFEM.m

Das Modell wird durch die folgenden Variablen definiert:

`name`	Text für Dateinamen und Beschriftung von Bildern
`Knoten`	Knotentabelle
`Element`	Element-/Stabtabelle
`Eigenschaft`	Eigenschaftstabelle
`Last`	Lasttabelle
`Bindung`	Bindungstabelle
`Uscale`	Faktor für Verschiebungsanzeige
`Fscale`	Faktor für Kraftanzeige

Daraus berechnet das Programm `StabFEM` die folgenden Werte

`a`	Zuordnungstabelle
`F`	Systemlastvektor
`frei`	Liste der Systemfreiheitsgrade ohne Bindungen
`N`	Schnittlasten
`ne`	Zahl der Elemente
`nk`	Zahl der Knoten
`S`	Systemsteifigkeitsmatrix

`U`	u-Werte für die Elementanzeige
`u`	Systemverschiebungsvektor
`unfrei`	Liste der Systemfreiheitsgrade mit Bindungen
`Uscale`	Vergrößerungsfaktor der Verschiebungsdarstellung
`V`	v-Werte für die Darstellung der Elementkanten
`X`	x-Werte für die Elementanzeige
`xg`	globaler Koordinatenvektor
`Y`	y-Werte für die Elementanzeige

Die folgenden Anweisungen sind in der Datei *StabFEM.m* enthalten.

Aus den Tabellen (Matrizen) werden intern der Systemkoordinatenvektor $\underline{x}^*$ und die Indexmatrix $\underline{a}$ gebildet. Zunächst werden die Dateien eingelesen und Knoten- und Elementanzahl festgestellt. Die Knoten- und Stabnummern in Knoten- und Stabtabelle werden in unserer einfachen Programmversion ignoriert. Wir gehen davon aus, dass beide fortlaufend numeriert sind.

Zahl der Knoten n_k und Elemente n_e

```
nk=size(Knoten,1);
ne=size(Element,1);
```

Systemkoordinatenvektor. In ihm sind für alle Knoten der Reihe nach die Koordinaten x und y angeordnet.

```
tmp=Knoten(:,2:3)';
xg=tmp(:);
```

Die zweite und dritte Spalte der Knotentabelle enthält die x- und y-Koordinaten der Knoten. Die Matrix `Knoten(:,2:3)'` enthält in der ersten Zeile die x-Werte und in der zweiten Zeile die y-Werte. Nun muss diese Matrix nur noch spaltenweise ausgelesen werden. Genau dies passiert, wenn auf die Elemente mit einem einzigen Index statt mit einem Zeilen- und Spaltenindexpaar zugegriffen wird (lineare Indizierung).

Die Variable `tmp` wird nur benutzt, um Matlab-kompatibel zu bleiben. In Octave kann jeder Ausdruck indiziert werden, während in Matlab nur Variablen indiziert werden können. In Octave hätte man also auch schreiben können: `xg=Knoten(:,2:3)'(:)`

Indexmatrix Die Matrix `a` enthält zeilenweise für jeden Stab die Nummern der lokalen Freiheitsgrade im Systemverschiebungsvektor $\underline{u}^*$. Die Indizes werden aus den Knotennummern für Stabanfang (3. Spalte der Elementmatrix `Element`) und Stabende (4. Spalte in `Element`) gebildet

```
a=[2*Element(:,3:4)-1  2*Element(:,3:4)];
```

Mit `a(2,[1 3])` beschafft man sich beispielsweise die Indizes der Freiheitsgrade des Stabanfangs für den Stab 2. Die Koordinaten dieses Punktes bekommt man mit

```
> xg(a(2,[1 3]))
ans =
  -2500
      0
```

Grafische Darstellung des Systems Dafür wird der plot-Befehl in der Form `plot(X,Y,format)` benutzt. Dieser Befehl zeichnet für jede Matrixspalte in `X` und `Y` eine Linie, wobei `X` die x-Koordinaten und `Y` die y-Koordinaten der Linienpunkte liefert. Jede Zeile der Matrizen entspricht einem Punkt. Das Argument `format` beschreibt die Linienart.

Zunächst beschaffen wir die beiden Matrizen:

```
X=xg(a(:,[1 3])');
Y=xg(a(:,[2 4])');
```

Mit `axis equal` bekommen beide Achsen den gleichen Maßstab.

`num2str(v)` wandelt die Elemente eines Spaltenvektors in einen Spaltenvektor aus Zeichenketten (gleichbedeutend mit einer Matrix aus Zeichen) um.

Der Befehl `text(x,y,t)` dient der Plazierung von Text im Bild. `x` und `y` sind Spaltenvektoren, die die Position des Textes angeben, `t` ist eine Matrix aus Zeichen, die zeilenweise den darzustellenden Text enthält oder eine Liste (*cell array*) mit Zeichenketten (*strings*), die den Text enthalten.

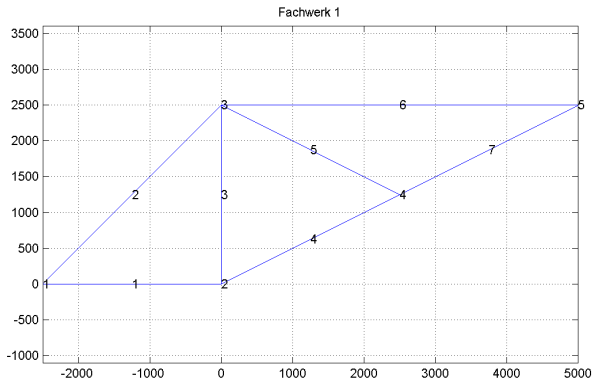


Abbildung 2: Geometrieplot

Die Knotenbeschriftung wird an die Position der Knoten geschrieben, die Elementbeschriftung wird in die Mitte der Stäbe geschrieben.

Mit dem Modellnamen (Variable `name`) wird das Diagramm beschriftet. Damit Unterstriche `_` nicht als Steuerzeichen für die Tiefstellung in Formeln interpretiert wird, muss der LaTeX-Interpreter von Matlab ausgeschaltet werden (Option `'interpreter'` auf den Wert `'none'`).

Da Zeichenketten Zeilenvektoren aus Zeichen sind, können sie wie andere Zeilenvektoren auch durch Nebeneinanderstellen zusammengefügt werden,

```
plot(X,Y,'b'); grid on; axis equal
text(Knoten(:,2),Knoten(:,3),num2str(Knoten(:,1)));
text(mean(X),mean(Y),num2str(Element(:,1)));
title(name,'interpreter','none');
print('-dpng', [name '_geo.png']);
snapnow;
```

Der Befehl `snapnow` ist für die Protokollerzeugung gedacht. Er bewirkt, dass das soeben erzeugte Bild mit ins Protokoll (siehe auch Abschnitt A.1) aufgenommen wird.

Aufbau und Lösung des Gesamtsystems Zunächst wird die Systemsteifigkeit $\underline{S}^*$(Variable `S`) initialisiert (als Nullmatrix). In der Schleife werden zunächst die Stabeigenschaften (`EA`) und der Indexvektor `ae` extrahiert. `Se` ist die Elementsteifigkeitsmatrix $\underline{S}_e$.

```
S=zeros(2*nk,2*nk);
for i=1:ne
    EA=prod(Eigenschaft(Element(i,2),2:3));
    ae=a(i,:);
    ve=xg(ae([2 4]))-xg(ae([1 3]));
    l=sqrt(ve'*ve);
    vn=ve/l;
    Veu=[-vn(1) vn(1) -vn(2) vn(2)]/l;
    Se=EA*l*Veu'*Veu;
    % Einbau in globale Matrix
    S(ae,ae)=S(ae,ae)+Se;
end
```

Die letzte Zeile `S(ae,ae)=S(ae,ae)+Se;` in der Schleife ist ein Beispiel für die Eleganz, die die Verwendung von Indexvektoren ermöglicht. Wenn man dieses Mittel nicht hätte, müsste man schreiben:

```
for i=1:4
    for j=1:4
        S(a(i),a(j))=S(a(i),a(j))+Se(i,j);
    end
end
```

Globaler Lastvektor Er wird zunächst als Nullvektor initialisiert. Seine Größe wird dabei vom globalen Koordinatenvektor `xg` übernommen. Die Lasttabelle enthält in der ersten Spalte `Last(:,1)` die Knotennummer, in der zweiten Spalte die Koordinatennummer (1 für x und 2 für y) und in der dritten Spalte den Wert. Aus Knoten- und Koordinatennummer wird der Index berechnet und dem entsprechenden Eintrag der vorgegebene Wert zugewiesen. Das alles findet wieder in einer impliziten Schleife statt, die über alle Zeilen in der Lasttabelle läuft.

Das funktioniert nur, wenn die Lasttabelle nicht leer ist.

```
F=0*xg;
if ~isempty(Last), F((Last(:,1)-1)*2+Last(:,2))=Last(:,3); end
```

Einbau der Randbedingungen Der Indexvektor `unfrei` listet alle globalen Freiheitsgrade auf, für die Randverschiebungen vorgegeben sind (Einträge in der Bindungstabelle). Dazu komplementär listet `frei` alle diejenigen globalen Freiheitsgrade, für die keine Randbedingungen vorgegeben sind.

Die Einträge in `unfrei` werden aus der Bindungstabelle `Bindung` bestimmt. Die Einträge von `frei` sind dann die Differenz der Menge (englisch: *set*) aller Freiheitsgrade (dargestellt durch den Vektor `1:2*nk`) und der Menge aller gebundenen Freiheitsgrade (Vektor `unfrei`).

```
unfrei=(Bindung(:,1)-1)*2+Bindung(:,2);
frei=setdiff(1:2*nk,unfrei);
```

Bestimmung des Systemverschiebungsvektors Der Systemverschiebungsvektor `u` wird zunächst mit Nullen initialisiert und mit den gegebenen Randwerten vorbelegt. Die unbekannten Komponenten werden als Lösung des reduzierten Gleichungssystems ermittelt.

```
u=0*xg);
u(unfrei)=Bindung(:,3);
u(frei)=S(frei,frei)\(F(frei)-S(frei,unfrei)*u(unfrei));
```

Der Zugriff auf Teilmengen von Vektoren und Matrizen mittels Indexvektoren (hier `frei` und `unfrei`) ist eine ganz zentrale Eigenschaft von Matlab und Octave. Schauen Sie sich die Wirkungsweise an, indem Sie folgende Ausdrücke anzeigen lassen:

```
unfrei
frei
u
u(unfrei)
u(frei)
S
S(frei,frei)
```

Auflagerreaktionen Die Auflagerreaktionen werden mit dem nun vollständig bekannten Verschiebungen und der Systemsteifigkeitsmatrix berechnet.

```
F(unfrei)=S(unfrei,:)*u;
```

Anzeige des verformten Systems und der Lasten Die Lage des verformten Systems ist die Summe aus Anfangslage und Verschiebung. Damit man die Verschiebung

erkennt, wird sie in der Regel vergrößert werden müssen. Dafür dient der Faktor `Uscale` . Um im gleichen Bild auch die äußeren Lasten und Auflagerreaktionen darstellen zu können, wird ein Skalierungsfaktor `Fscale` eingeführt, der mit der Kraft multipliziert eine Länge ergibt.

Obwohl diese Faktoren eigentlich nur der Programmsteuerung dienen, ist es sinnvoll, sie bei der Systembeschreibung mit anzugeben, denn welche Werte sinnvoll sind, ist systemabhängig. Die folgenden Zeilen prüfen, ob die Skalierungsfaktoren definiert wurden und stellen anderenfalls Standardwerte (*defaults*) bereit.

```
if exist('Uscale','var')~=1, Uscale=100; end
if exist('Fscale','var')~=1, Fscale=0.001; end
```

`xp` enthält die skalierte verformte Lage der Knoten. Die Koordinatenmatrizen `x` und `y` werden aus `xp` genauso gebildet, wie die Matrizen `X` und `Y` aus dem Systemkoordinatenvektor gebildet wurden.

```
xp=xg+Uscale*u;
x=xp(a(:,1:2)');
y=xp(a(:,3:4)');
Fx=Fscale*F(1:2:end);
Fy=Fscale*F(2:2:end);
```

Zunächst werden die unverformte Lage in blau und die verformte Lage in rot dargestellt. Anschließend werden die Pfeile für die Lasten und Auflagerreaktionen ergänzt. Dies leistet der Befehl `quiver(x,y,vx,vy)`, dessen erste beide Argumente die Position der Anfangspunkte und dessen drittes und viertes Argument die x- und y-Komponenten der Pfeillänge enthalten. Die Pfeillänge wird automatisch skaliert.

```
plot(X,Y,'b',x,y,'r');grid on;axis equal
hold on
quiver(xp(1:2:end),xp(2:2:end),F(1:2:end),F(2:2:end),'k');
hold off
title(name);
print('-dpng', [name '_disp.png']);
```

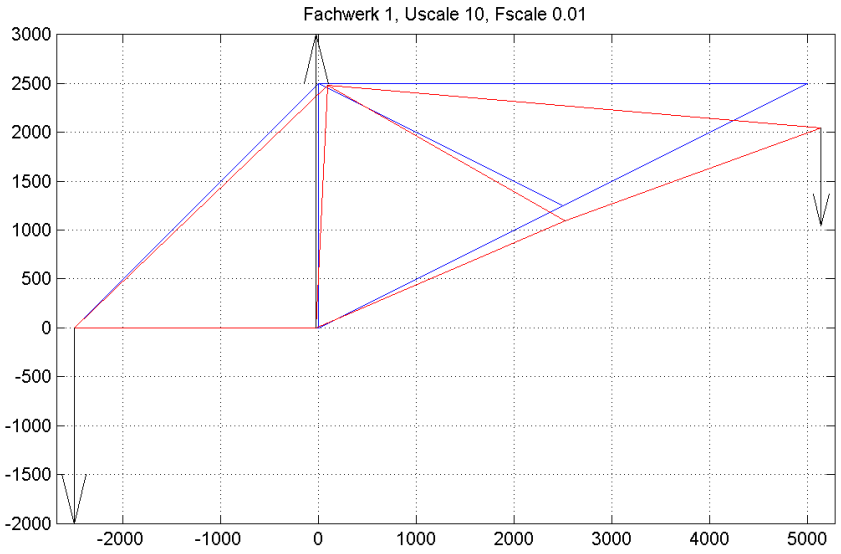


Abbildung 3: Verformte Lage mit Lasten und Auflagerreaktionen

Berechnung der Schnittlasten Die Schnittlasten werden mit den Stabeigenschaften und dem Verschiebungs-Verzerrungs-Operator aus dem Verschiebungszustand der Elemente bestimmt. Die Rechnung ist ähnlich derjenigen für die Bestimmung der Elementsteifigkeitsmatrix. In professionellen FEM-Programmen wird der Aufbau der Steifigkeitsmatrix und die Bestimmung der Schnittlasten bei gegebenen Verschiebungen daher auch oft im selben Unterprogramm erledigt. Damit verwendet man einen Teil des Codes doppelt.

```
N=zeros(ne,1);
for i=1:ne
    ae=a(i,:);
    EA=prod(Eigenschaft(Element(i,2),2:3));
    ve=xg(ae([2 4]))-xg(ae([1 3]));
```

```
    l=sqrt(ve'*ve);
    vn=ve/l;
    Veu=[-vn(1) vn(1) -vn(2) vn(2)]/l;
    N(i)=EA*Veu*u(ae);
end
```

4. Scheibenelement

Für Scheibenelemente wollen wir zunächst kein komplettes FEM-Programm schreiben, sondern die Eigenschaften linearer 4-Knoten-Elemente genauer untersuchen. Die Lage solcher Elemente ist gegeben durch den Element-Koordinatenvektor

$$\underline{\mathrm{x}}_e = \left[\begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & y_1 & y_2 & y_3 & y_4 \end{array} \right]^T$$

und den Element-Verschiebungsvektor

$$\underline{\mathrm{u}}_e = \left[\begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & v_1 & v_2 & v_3 & v_4 \end{array} \right]^T .$$

4.1. Interpolation und Ansatzfunktionen

Hauptaufgabe der Ansatzfunktionen ist es, Werte, die an den Knoten eines Elements bekannt sind (z.B. Lage und Verschiebung) innerhalb des Elements zu interpolieren. Damit wird die an den einzelnen Punkten definierte Größe auf das ganze Gebiet übertragen.

Zunächst sehen wir, wie man sich solche Ansatzfunktionen beschaffen kann. Ein Element habe n_k Knoten, deren und Lage im Elementkoordinatensystem r, s gegeben ist. Jedem Knoten wird eine Ansatzfunktion $h_k(r, s)$ zugeordnet, so dass sie an diesem Knoten den Wert 1 und sonst den Wert 0 hat. Das ist die entscheidende Bedingung, die die Ansatzfunktionen erfüllen müssen. Außerdem müssen die Funktionen hinreichend glatt (stetig) sein.

Für jede Ansatzfunktion können also n_k Gleichungen aufgestellt werden. Die Werte an den Knoten des Elements sind ja bekannt. Die Ansatzfunktionen können also je n_k freie Koeffizienten enthalten. Ein Polynom-Ansatz in r und s kann also zum Beispiel bei 4 Knoten die folgenden Terme enthalten

$$h_k(r, s) = a_{k1} + a_{k2} r + a_{k3} s + a_{k4} rs$$

oder bei 8 Knoten:

$$h_k(r, s) = a_{k1} + a_{k2} r + a_{k3} s + a_{k4} rs + a_{k5} r^2 + a_{k6} s^2 + a_{k7} r^2 s + a_{k8} rs^2$$

Die weitere Rechnung ist nur für besonders neugierige Studenten von Wert. Alle anderen können sie überspringen.

Nun seien die Koordinaten r und s der Knoten in den Vektoren $\underline{\mathrm{r}}$ und $\underline{\mathrm{s}}$ gegeben, z.B. für ein Vier-Knoten-Element

$$\underline{\mathrm{r}} = \left[\begin{array}{c} -1 \\ 1 \\ 1 \\ -1 \end{array} \right], \quad \underline{\mathrm{s}} = \left[\begin{array}{c} -1 \\ -1 \\ 1 \\ 1 \end{array} \right].$$

Dann können beispielsweise die Koeffizienten der Ansatzfunktion $h_1$aus dem folgenden Gleichungssystem bestimmt werden:

$$\begin{aligned} a_{11} - a_{12} - a_{13} + a_{14} &= 1 \\ a_{11} + a_{12} - a_{13} - a_{14} &= 0 \\ a_{11} + a_{12} + a_{13} + a_{14} &= 0 \\ a_{11} - a_{12} + a_{13} - a_{14} &= 0 \end{aligned}$$

Das lässt sich für alle Ansatzfunktionen gleichzeitig aufschreiben. Die unbekannten Koeffizienten der Ansatzfunktionen werden in einer Matrix $\underline{\mathrm{A}}$ zusammengefasst, die zeilenweise den Knoten k und spaltenweise den Termen 1, r, s und rs des Ansatzes zugeordnet ist. Die Einheitsmatrix auf der rechten Seite bildet in jeder Spalte die Bedingung ab, dass die k-te Ansatzfunktion nur am Knoten k den Wert 1 haben darf.

$$\begin{bmatrix} 1 & r_1 & s_1 & r_1 s_1 \\ 1 & r_2 & s_2 & r_2 s_2 \\ 1 & r_3 & s_3 & r_3 s_3 \\ 1 & r_4 & s_4 & r_4 s_4 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \underline{\mathrm{MA}}^T = \underline{\mathrm{I}}$$

In Matlab ist dieses Gleichungssystem kurz und schmerzlos aufgestellt und gelöst:

```
r=[-1 1 1 -1]';
s=[-1 -1 1 1]';
M=[ones(4,1) r s r.*s];
A=(M\eye(4))'
A =
   0.25000  -0.25000  -0.25000   0.25000
   0.25000   0.25000  -0.25000  -0.25000
   0.25000   0.25000   0.25000   0.25000
   0.25000  -0.25000   0.25000  -0.25000
```

Damit können die Ansatzfunktionen geschrieben werden

$$\begin{bmatrix} h_1(r,s) \\ h_2(r,s) \\ h_3(r,s) \\ h_4(r,s) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ r \\ s \\ rs \end{bmatrix} \tag{1}$$

Wenn nun α_k die an den Knoten gegebenen Werte irgendeiner Größe (Verschiebungskomponenten oder Koordinatenwerte) sind, dann ist der Wert an der Stelle r, s im Inneren des Elements gleich der Summe aus den Produkten der Knotenwerte mit den jeweiligen Ansatzfunktionen:

$$\alpha(r,s) = \sum_{k=1}^{4} h_k(r,s)\alpha_k$$

oder in Matrixform geschrieben:

$$\begin{aligned}\alpha(r,s) &= \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{bmatrix} \begin{bmatrix} h_1(r,s) \\ h_2(r,s) \\ h_3(r,s) \\ h_4(r,s) \end{bmatrix} \\ &= \frac{1}{4} \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ r \\ s \\ rs \end{bmatrix}\end{aligned}$$

4.2. Darstellung der Ansatzfunktionen

Im linearen Viereckelement lauten die Ansatzfunktionen

$$\begin{aligned} h_1(r,s) &= \frac{1}{4}(1-r)(1-s) \\ h_2(r,s) &= \frac{1}{4}(1+r)(1-s) \\ h_3(r,s) &= \frac{1}{4}(1+r)(1+s) \\ h_4(r,s) &= \frac{1}{4}(1-r)(1+s) \end{aligned}$$

Für die Darstellung benutzen wir die Funktion `ezsurf`, die eine Funktion zweier Variable als eingefärbte Fläche darstellen kann. Um alle vier Funktionen in einem Bild darzustellen, benutzen wir die Funktion `subplot`. Um von rechts unten auf die Flächen zu schauen, ändern wir noch die Blickrichtung.

Listing 2: Ansatz.m

```
% Plot der Ansatzfunktionen eines linearen Viereck-Elements
clf;
v=[20,50];
subplot(2,2,1);
ezsurf(@(r,s) 0.25*(1-r).*(1+s),[-1 1],20);
set(gca,'view',v);
subplot(2,2,2);
ezsurf(@(r,s) 0.25*(1+r).*(1+s),[-1 1],20);
set(gca,'view',v);
subplot(2,2,3);
ezsurf(@(r,s) 0.25*(1-r).*(1-s),[-1 1],20);
set(gca,'view',v);
subplot(2,2,4);
ezsurf(@(r,s) 0.25*(1+r).*(1-s),[-1 1],20);
set(gca,'view',v);
print -dpng Ansatz.png
```

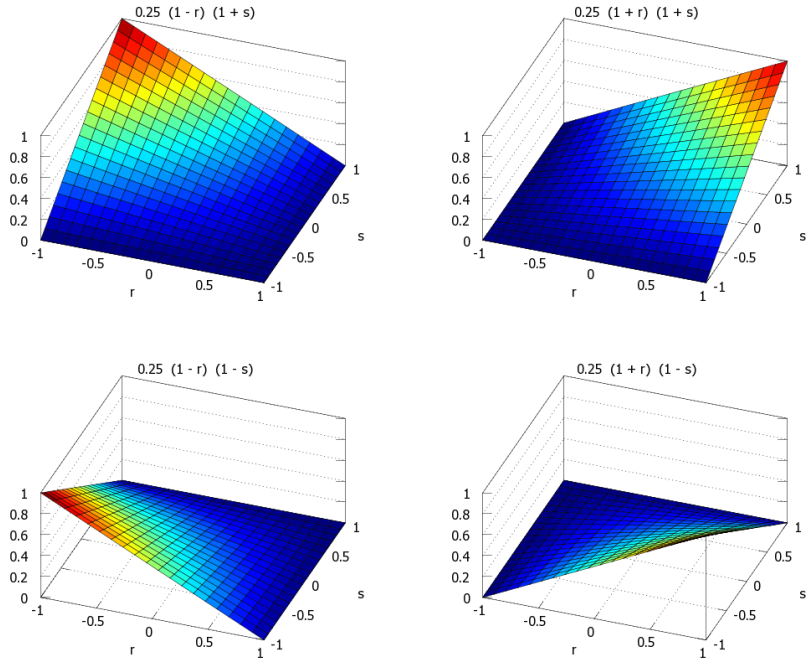


Abbildung 4: Ansatzfunktionen für das lineare 4-Knoten-Scheibenelement

Die Plotbefehle für Funktionen sind etwas hinterhältig, denn sie rufen die Funktion mit vektoriellen Argumenten auf, in denen alle benötigten Punkte enthalten sind. Dadurch wird auch bei engmaschigen Gittern nur ein Funktionsaufruf benötigt. Wir müssen aber sicherstellen, dass die Rechenoperationen elementweise erfolgen, indem wir z.B. bei Produkten der Argumente die Operationen `.*` oder `./` oder `.^` verwenden. Anderenfalls wird die entsprechende Matrixoperation ausgeführt, was beim Funktionsplotten aber nur in Ausnahmefällen sinnvoll ist.

Ansonsten sind die `ez`-Funktionen extrem komfortabel, da sie aus der Funktionsdefinition den Titel und die Achsenbeschriftungen erzeugen.

Der @-Operator steht für eine „anonyme" Funktion, die dort verwendet wird, wo sie definiert wird, also hier zum Beispiel im Plotten. Man gibt einfach die Argumente und die Formel wie oben gezeigt an.

4.3. Ansatzfunktion als Unterprogramm mit Argumentvektoren

Für die spätere Berechung und Anzeige von Feldgrößen (Verschiebungen, Verzerrungen, Spannungen) im Elementinneren müssen wir deren Werte auf einem rechteckigen Werteraster für die Elementkoordinaten r und s bereitstellen. Die folgende Funktion nimmt als Argumente$r$ und s in Form von Spaltenvektoren mit n Elementen (diese Zahl ist beliebig) und liefert eine $n \times 4$ Matrix, deren Spalten den Werten der Ansatzfunktionen $h_1 \ldots h_4$ an den gegebenen Stellen entsprechen.

Listing 3: h41.m

```
function h=h41(r,s)
% lineare Ansatzfunktionen für ein 4-Knotenelement
% Vektorisierte Version, r und s können Zeilenvektoren sein.
a=0.25*[ 1 -1 -1  1;...
         1  1 -1 -1;...
         1  1  1  1;...
         1 -1  1 -1];
h=[ones(size(r)) r s r.*s]*a';
```

Um alle Ansatzfunktionen auf einmal zu berechnen, haben wir die Gleichung (1) benutzt.

4.4. Verschiebungszustand im Element

Mit Hilfe der Ansatzfunktionen und des Elementkoordinatenvektors $\underline{x}_e$ sowie des Elementverschiebungsvektors $\underline{u}_e$ werden nun die Verläufe der Ortskoordinaten und der Verschiebungen im Elementinneren dargestellt. Die Verläufe der Ortskoordinaten sind natürlich trivial, diese Darstellung soll nur zeigen, dass die Interpolation korrekt funktioniert.

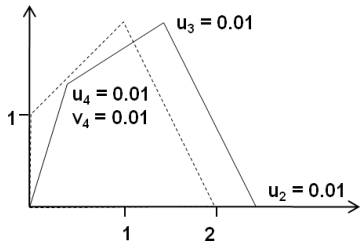


Abbildung 5: Beispielgeometrie

Als Beispiel verwenden wir das oben gezeigt e schiefwinklige Viereckselement, damit man auch was sieht. Wir definieren zunächst die Elementvektoren $\underline{x}_e$ und $\underline{u}_e$:

Listing 4: Viereck1.m

```
name='Viereck1';
xe=[0 2 1 0 0 0 2 1]';
ue=[0 0.01 0.01 0.01 0 0 0 0.01]';
E=210000;
nu=0.3;
```

Die Zeile

Listing 5: Scheibe.m

```
[r,s]=meshgrid(-1:0.1:1,-1:0.1:1);
```

stellt das Punkteraster für die Elementkoordinaten r und s bereit, an dem dann die Feldgrößen ausgewertet werden.

Die Form des Rasters (Zeilenzahl, Spaltenzahl) wird dann auf die Koordinaten und Verschiebungen übertragen, die Werte werden anschließend überschrieben.

Für die Auswertung der Ansatzfunktion und deren Multiplikation mit den Knotenwerten wird das Raster linear indiziert (`r(:),s(:)`), also als fortlaufender Spaltenvektor betrachtet. Das Ergebnis ist automatisch in der richtigen Form, weil wir sie ja vorher von `r` übernommen haben.

Listing 6: Scheibe.m

```
x=r;y=r;u=r;v=r;
x(:)=h41(r(:),s(:))*xe(1:4);
y(:)=h41(r(:),s(:))*xe(5:8);
u(:)=h41(r(:),s(:))*ue(1:4);
v(:)=h41(r(:),s(:))*ue(5:8);
```

Mit den nun folgenden Zeilen wird Abbildung 6 erzeugt. Die Funktion `contourf` erzeugt farbige Höhenlinienplots. Die Achsen werden auf gleichen Maßstab gesetzt und ihre Anzeige abgeschaltet. Der Titel muss anders als bei den `ez-` (Easy)-Plotfunktionen hier extra angegeben werden.

Listing 7: Scheibe.m

```
subplot(2,2,1);
    contourf(x,y,x); axis equal off; colorbar
    title 'Koordinate x'
subplot(2,2,2);
    contourf(x,y,y); axis equal off; colorbar
    title 'Koordinate y'
subplot(2,2,3);
    contourf(x,y,u); axis equal off; colorbar
    title 'Verschiebung u'
subplot(2,2,4);
    contourf(x,y,v); axis equal off; colorbar
    title 'Verschiebung v'
print('-dpng', [name '_Interpolation.png']);
```

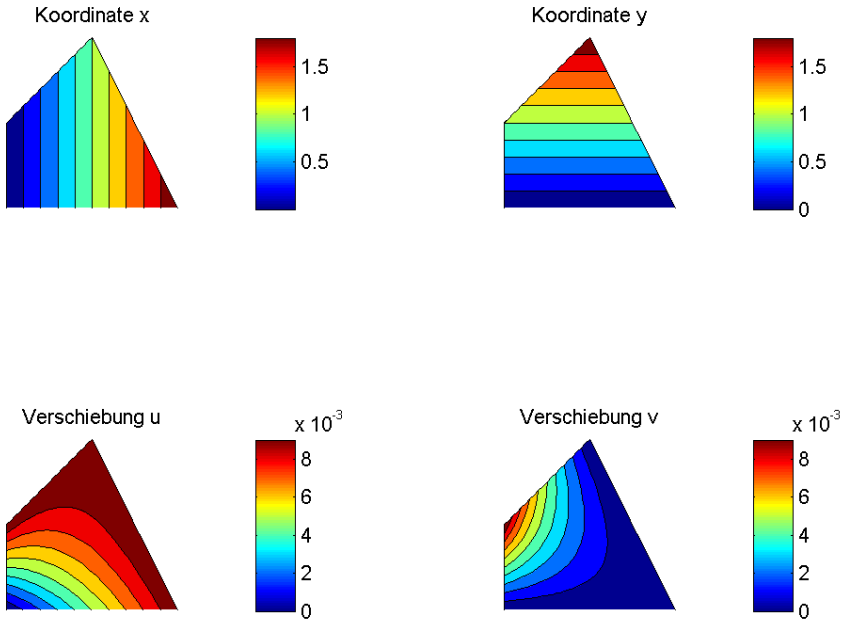


Abbildung 6: Interpolation im linearen 4-Knoten-Scheibenelement mit der in Listing 4 definierten Geometrie und Verschiebung.

4.5. Ableitungen der Ansatzfunktionen

Für die Berechnung der Verzerrung benötigen wir die Ableitungen der Ansatzfunktionen nach den Elementkoordinaten. Wir wollen sie grafisch dargestellen. Das Sonderzeichen ∂ in den Diagrammtiteln wird mit dem Text `\partial` erzeugt. Das ist Latex-Code. Latex ist ein Satzsystem für Mathematik-lastige Texte und wird auch benutzt, um dieses Skript zu setzen.

$$\begin{bmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} -1+s & 1-s & 1+s & -1-s \\ -1+r & -1-r_1 & 1+r & 1-r \end{bmatrix} \tag{2}$$

Listing 8: Plot_dhdr.m

```
[r,s]=meshgrid(-1:0.1:1,-1:0.1:1);
dhdr=0.25*[-1+s(:) 1-s(:) 1+s(:) -1-s(:)...
           -1+r(:) -1-r(:) 1+r(:) 1-r(:)];
clf;
for j=1:4
    subplot(2,4,j),
        contourf(r,s,reshape(dhdr(:,j),size(r)));
        caxis([-0.5 0.5]);axis equal;
        colorbar('southoutside');
```

```
        title(['\partial_h_' num2str(j) '/\partial_r']);
    subplot(2,4,j+4),
        contourf(r,s,reshape(dhdr(:,j+4),size(r)));
        caxis([-0.5 0.5]);axis equal;
        colorbar('southoutside');
        title(['\partial_h_' num2str(j) '/\partial_s']);
end
print -dpng dhdr.png
```

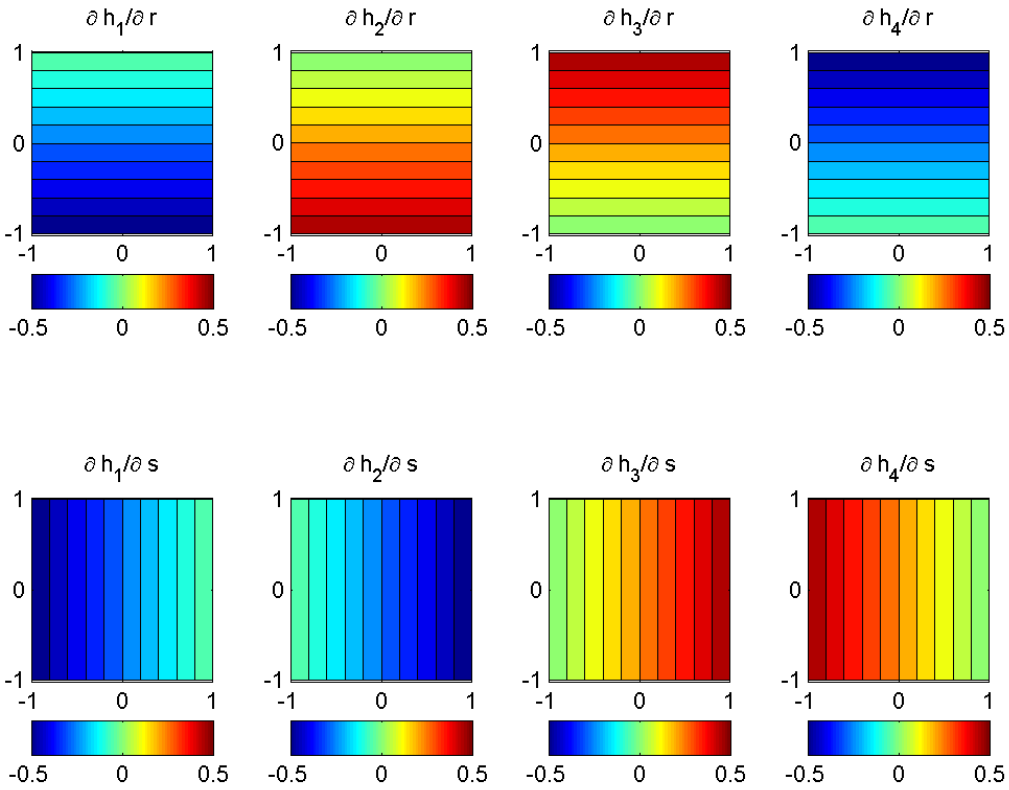


Abbildung 7: Ableitungen der Ansatzfunktionen nach r und s

4.6. Jacobideterminante

Zunächst wird eine Funktion definiert, die die Jacobideterminante berechnet. Als Argumente benötigt sie die Element-Knotenkoordinaten und die Werte der Elementkoordinaten r und s, an denen sie ausgewertet wird.

$$\underline{\mathrm{J}} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} = \frac{1}{4} \begin{bmatrix} -1+s & 1-s & 1+s & -1-s \\ -1+r & -1-r & 1+r & 1-r \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} \tag{3}$$

Listing 9: Jacobi.m

```
function J=Jacobi(xe,r,s)
J=r;
for i=1:length(r(:))
        dhr=0.25*[-1+s(i) 1-s(i) 1+s(i) -1-s(i);...
                  -1+r(i) -1-r(i) 1+r(i) 1-r(i)];
        J(i)=det(dhr*[xe(1:4) xe(5:8)]);
end
```

Das folgende Listing zeigt das Skript für die Berechnung von Abbildung 8 auf der nächsten Seite. Das Netz der Elementkoordinaten in der realen Lage (linkes Bild) wird mit `plot` erzeugt, der Farbplot mit `contourf`. Knotenkoordinaten und -verschiebungen werden aus der Datei *Viereck1.m* auf Seite 20 eingelesen.

Listing 10: Jacobiplot.m

```
Viereck1;
[r,s]=meshgrid(-1:0.1:1,-1:0.1:1);
J=Jacobi(xe,r,s);
x=r;y=s;
x=h41(r(:),s(:))*xe(1:4);
y=h41(r(:),s(:))*xe(5:8);
clf;
subplot(1,2,1);
    plot(x,y,'k',x',y','k'); axis equal
    title 'Koordinatenlinien r und s'
subplot(1,2,2)
    contourf(x,y,reshape(J(:),size(r)));
    axis equal off; colorbar('southoutside')
    title(['Jacobideterminante']);
print -dpng detJ.png
```

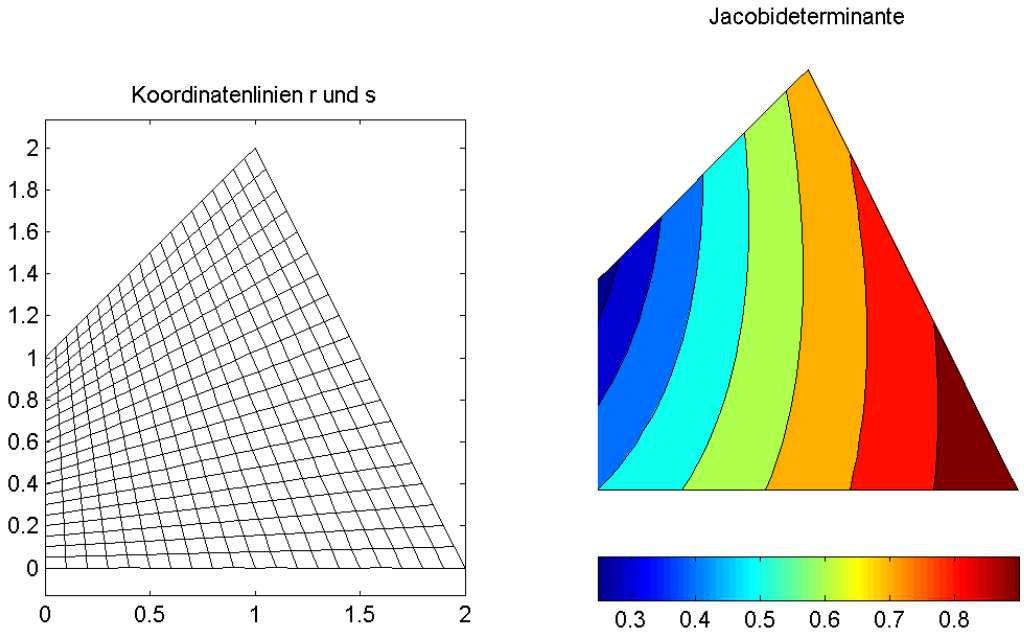


Abbildung 8: Jacobideterminante

4.7. Verzerrungen

Für die Berechnung der Verzerrungen ε_x, ε_y und γ_{xy} steht die Funktion `Verzerrung` zur Verfügung. Sie benötigt als Argumente den Element-Knotenkoordinatenvektor $\underline{x}_e$, den Elementverschiebungsvektor $\underline{u}_e$ sowie die Werte der Elementkoordinaten r und s, an denen sie ausgewertet wird. r und s können als Matrix vorliegen, dann werden die Verzerrungskomponenten ebenfalls als Matrix zurückgegeben.

Dafür müssen zunächst die Ableitungen `dhdr` der Ansatzfunktionen nach den Elementkoordinaten r und s gemäß (2) und die Jacobimatrix `J` gemäß (3) berechnet werden. Damit können dann die Ortsableitungen der Ansatzfunktionen `d` berechnet werden:

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \end{bmatrix} = \underline{J}^{-1} \frac{1}{4} \begin{bmatrix} -1+s & 1-s & 1+s & -1-s \\ -1+r & -1-r_1 & 1+r & 1-r \end{bmatrix}$$

Diese werden umsortiert und bilden den Element-Verschiebungs-Verzerrungsoperator `deu`,

der den Element-Verschiebungsvektor $\underline{u}_e$ (ue) auf die Verzerrungen e abbildet.

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} h_{1,x} & h_{2,x} & h_{3,x} & h_{4,x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_{1,y} & h_{2,y} & h_{3,y} & h_{4,y} \\ h_{1,y} & h_{2,y} & h_{3,y} & h_{4,y} & h_{1,x} & h_{2,x} & h_{3,x} & h_{4,x} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

Die Funktion Verzerrung führt diese Berechnung in einer Schleife über alle gewünschten Punkte im Element durch. Das versucht man normalerweise in Matlab zu vermeiden, aber die dafür notwendigen Operationen mit mehrdimensionalen Matrizen wären zu unübersichtlich geworden. So können wir im Programmtext die Formeln noch gut wiedererkennen.

Listing 11: Verzerrung.m

```
function [ex,ey,gxy]=Verzerrung(xe,ue,r,s)
ex=r; % Form der Koordinatenmatrix übernehmen
ey=r;
gxy=r;
for i=1:length(r(:))
    dhdr=0.25*[-1+s(i) 1-s(i) 1+s(i) -1-s(i);...
               -1+r(i) -1-r(i) 1+r(i) 1-r(i)];
    J=dhdr*[xe(1:4) xe(5:8)];
    d=J^-1*dhdr;
    deu=[d(1,:)  0 0 0 0;...
         0 0 0 0 d(2,:);...
         d(2,:)  d(1,:) ];
    e=deu*ue;
    ex(i)=e(1); % Ergebniswerte einsortieren
    ey(i)=e(2);
    gxy(i)=e(3);
end
```

In der Fortsetzung des Skripts *Scheibe.m* werden die Verzerrungen berechnet und dargestellt.

Listing 12: Scheibe.m

```
[ex,ey,gxy]=Verzerrung(xe,ue,r,s);
clf;
% Anfangskontur
lx=xe([1:4 1]);
ly=xe([5:8 5]);
% Darstellung
subplot(2,2,1);
    contourf(x+u,y+v,ex); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Längsdehnung \epsilon_x'
```

```
subplot(2,2,2);
    contourf(x+u,y+v,ey); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Längsdehnung \epsilon_y'
subplot(2,2,3);
    contourf(x+u,y+v,gxy); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Schubverzerrung \gamma_{xy}'
print('-dpng', [name '_Verzerrung.png']);
```

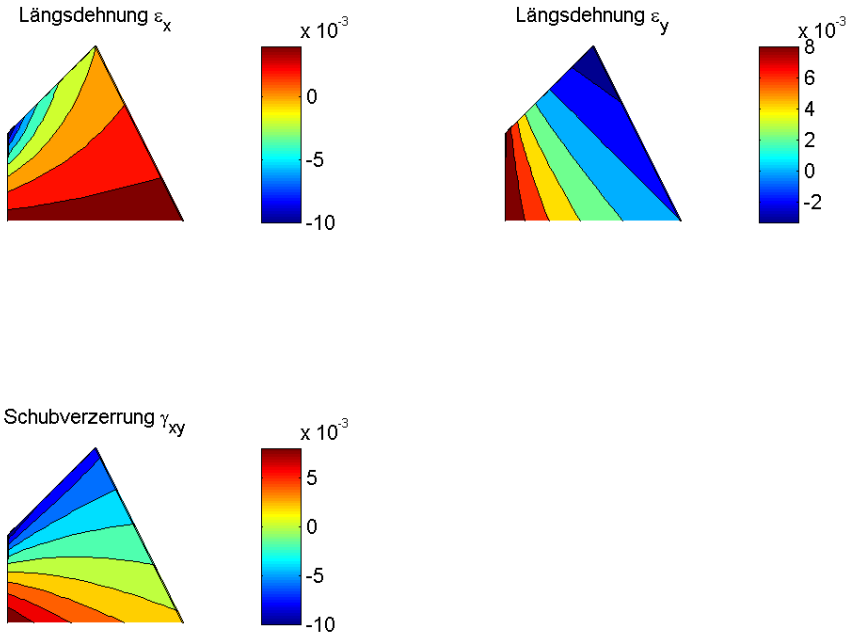


Abbildung 9: Verzerrungsfeld im Scheibenelement

4.8. Spannungen

Bei Scheibenelementen werden der ebene Spannungszustand (ESZ) und der ebene Verzerrungszustand (EVZ) unterschieden. Für die Spannungsberechnung muss daher neben den elastischen Konstanten E (E-Modul) und ν (Querkontraktionszahl) angegeben werden, welche der beiden Varianten vorliegt.

4.8.1. Ebener Spannungszustand

Die Dehnung in Dickenrichtung kann mit der Querkontraktionszahl ν aus den Längsdehnungen berechnet werden:

$$\varepsilon_z = -\nu\left(\varepsilon_x + \varepsilon_y\right).$$

Spannung:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix}.$$

Diese Rechnungen für den ESZ werden in der Funktion `ESZ` durchgeführt:

Listing 13: ESZ.m

```
function [sx,sy,txy,ez]=ESZ(E,nu,ex,ey,gxy)
sx=E/(1-nu^2)*(ex-nu*ey);
sy=E/(1-nu^2)*(ey-nu*ex);
txy=E/(2+2*nu)*gxy;
ez=-nu*(ex+ey);
```

Aufgerufen wird die Funktion im Skript *Scheibe.m*, dort wird auch das folgende Bild erzeugt.

Listing 14: Scheibe.m

```
[sx,sy,txy,ez]=ESZ(E,nu,ex,ey,gxy);
% Darstellung
subplot(2,2,1);
    contourf(x+u,y+v,sx); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Längsspannung \sigma_x'
subplot(2,2,2);
    contourf(x+u,y+v,sy); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Längsspannung \sigma_y'
subplot(2,2,3);
    contourf(x+u,y+v,txy); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Schubspannung \tau_{xy}'
subplot(2,2,4);
    contourf(x+u,y+v,ez); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Dickendehnung \epsilon_z'
print('-dpng', [name '_ESZ.png']);
```

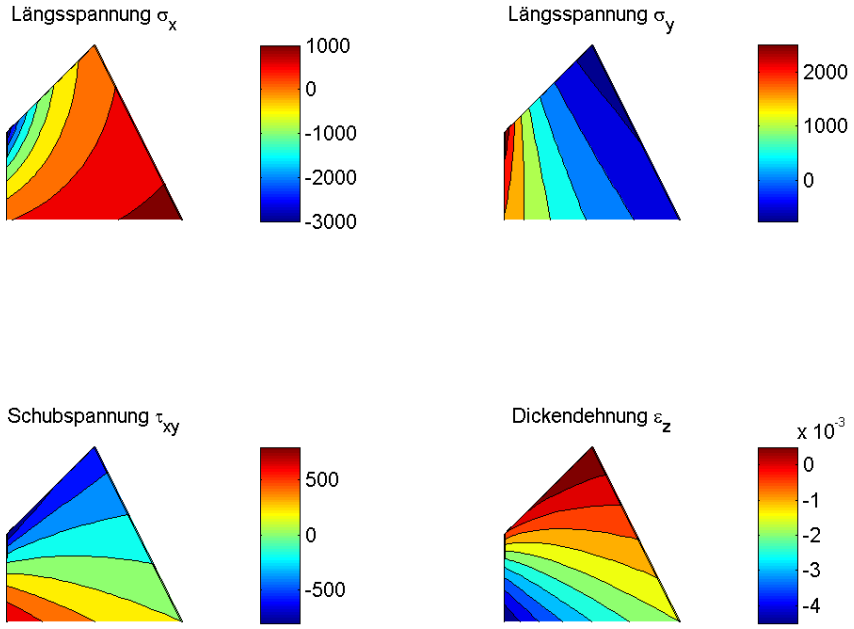


Abbildung 10: Ebener Spannungszustand im Scheibenelement

4.8.2. Ebener Verzerrungszustand

Die Materialgleichungen für den ebenen Verzerrungszustand lauten

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} 2G+\lambda & \lambda & 0 \\ \lambda & 2G+\lambda & 0 \\ \lambda & \lambda & 0 \\ 0 & 0 & G \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix}$$

mit den Laméschen Konstanten

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = G = \frac{E}{2(1+\nu)}$$

Diese Rechnungen werden in der Funktion `EVZ` durchgeführt:

Listing 15: EVZ.m

```
function [sx,sy,txy,sz]=EVZ(E,nu,ex,ey,gxy)
l=E*nu/((1+nu)*(1-2*nu));
m=E/(2+2*nu);
sx=(2*m+l)*ex+l*ey;
sy=(2*m+l)*ey+l*ex;
sz=l*ex+l*ey;
txy=m*gxy;
```

Aufgerufen wird die Funktion im Skript *Scheibe.m*, dort wird auch das folgende Bild erzeugt.

Listing 16: Scheibe.m

```
[sx,sy,txy,sz]=EVZ(E,nu,ex,ey,gxy);
% Darstellung
subplot(2,2,1);
    contourf(x+u,y+v,sx); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Längsspannung \sigma_x'
subplot(2,2,2);
    contourf(x+u,y+v,sy); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Längsspannung \sigma_y'
subplot(2,2,3);
    contourf(x+u,y+v,txy); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Schubspannung \tau_{xy}'
subplot(2,2,4);
    contourf(x+u,y+v,sz); axis equal off; colorbar
    line(lx,ly,'color','k');
    title 'Dickenspannung \sigma_z'
print('-dpng', [name '_EVZ.png']);
```

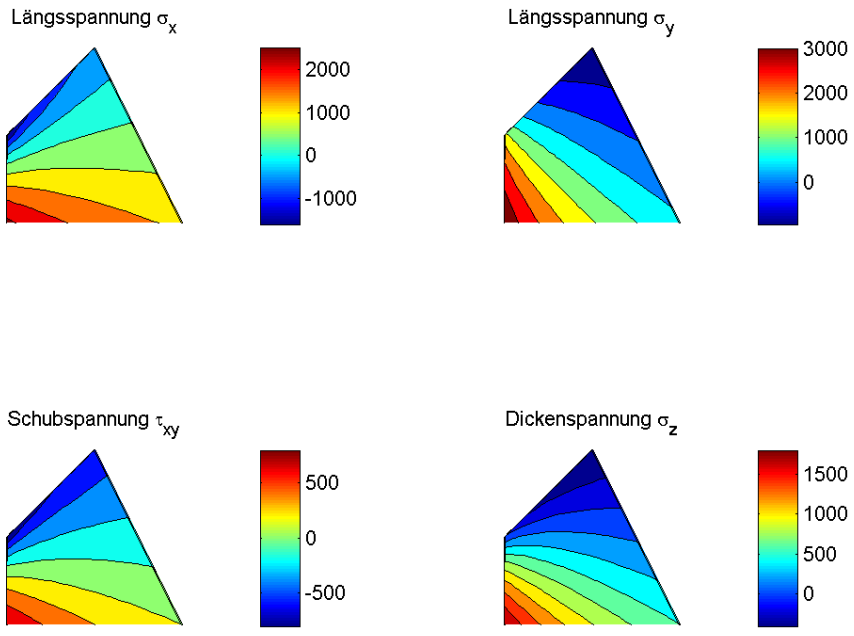


Abbildung 11: Ebener Verzerrungszustand im Scheibenelement

4.9. Gauss-Integration

Die folgende Funktion stellt die Stützstellen und die Gewichte der Gauss-Integrationsformel für die Ordnungen 1 und 2 bereit.

Listing 17: Gauss.m

```
function [r,s,w]=Gauss(n)
if n==1
    r=0;s=0;w=4;
elseif n==2
    a=3^(-0.5);
    [r,s]=meshgrid([-a a],[-a a]);
        w=ones(2,2);
end
```

Wir wollen uns überzeugen, wie gut die Gauss-Integration auch bei verzerrten Elementgeometrien funktioniert. Dafür berechnen wir die Fläche von verschiedenen Viereckselementen als Integral über die Jacobideterminante in Elementkoordinaten.

```
[r,s,w]=Gauss(2)
xe=[0 1 1 0 0 0 1 1]'
J=Jacobi(xe,r,s)
w(:)'*J(:)
```

4.10. Elementsteifigkeitsmatrix

Die Elementsteifigkeitsmatrix für das 4-Knoten-Scheibenelement berechnet sich nach der Formel

$$\underline{S}_e = t \int_{-1}^{1} \int_{-1}^{1} \underline{V}_{\varepsilon u}^T \underline{CV}_{\varepsilon u} \det \underline{J} dr ds$$

Listing 18: Steifigkeit

```
function S=Steifigkeit(xe,t,E,nu,s,n)
if s==0 % ESZ
    C=E/(1-nu^2)*[1 nu 0; nu 1 0; 0 0 (1-nu)/2];
else % EVZ
    l=E*nu/((1+nu)*(1-2*nu));
    m=E/(2+2*nu);
    C=[2*m+l l 0; l 2*m+l 0; 0 0 m];
end
S=zeros(8,8);
[r,s,w]=Gauss(n);
for i=1:length(r(:))
    dhdr=0.25*[-1+s(i) 1-s(i) 1+s(i) -1-s(i);...
               -1+r(i) -1-r(i) 1+r(i) 1-r(i)];
    J=dhdr*[xe(1:4) xe(5:8)];
    d=J^-1*dhdr;
```

```
    veu=[d(1,:)   0  0  0  0;...
          0  0  0  0  d(2,:);...
          d(2,:)   d(1,:)  ];
        S=S+w(i)*veu'*C*veu*det(J);
end
S=S*t;
```

5. FEM für Scheiben

Nun haben wir die wesentlichen Bausteine für eine Scheibenelement untersucht und wollen ein kleines FEM-Programm für Scheibenprobleme erstellen, mit dem man schon richtig rechnen kann.

Eine Rechnung läuft in drei Schritten ab.

1. Einlesen der Systembeschreibung. Diese wird als Matlab-Skript erstellt, ein Beispiel zeigt Listing 19, die Variablen sind im folgenden Abschnitt erläutert.
2. Aufstellen und Lösen des Gleichungssystems durch Aufruf des Skripts `FEM`.
3. Nachlaufrechnung zur Anzeige von Ergebnissen mit dem Skript `Sichter`. Dafür wird zunächst die Variable `d` mit der Bezeichnung der anzuzeigenden Größe belegt, mögliche Werte sind

`'re'`	Elementkoordinate r
`'se'`	Elementkoordinate s
`'xe'`	Ortskoordinate x
`'ye'`	Ortskoordinate y
`'ue'`	Verschiebung u
`'ve'`	Verschiebung v
`'sx'`	Längsspannung σ_x
`'sy'`	Längsspannung σ_y
`'txy'`	Schubspannung τ_{xy}
`'ex'`	Längsdehnung ε_x
`'sy'`	Längsdehnung ε_y
`'txy'`	Schubverzerrung γ_{xy}

5.1. Beispiel

Die oben genannten Schritte werden an einem Beispielsystem demonstriert, wie es im Grundlagenskript ausführlich erläutert ist. Das System ist in Listing 19 definiert

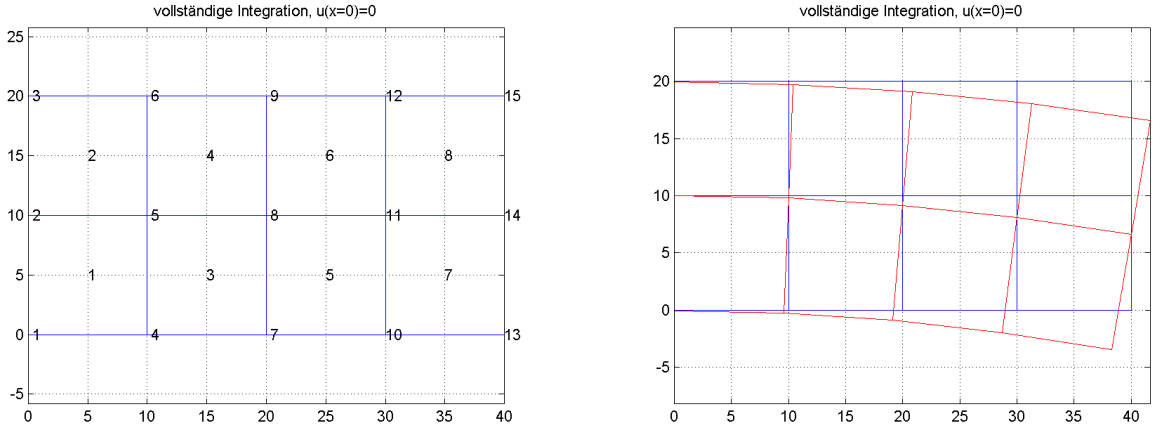
```
>> ESZ_2x4_M_n2_b3
>> FEM
>> d='sx'
>> Sichter
```

Das Skript FEM erzeugt die beiden folgenden Bilder, das Skript Sichter erzeugt Abbildung

Listing 19: ESZ_2x4_M_n2_b3.m

```
name='vollständige Integration, u(x=0)=0'
% Nr  x  y
Knoten=[...
1  0   0;...
2  0   10;...
3  0   20;...
4  10   0;...
5  10   10;...
6  10   20;...
7  20   0;...
8  20   10;...
9  20   20;...
10  30   0;...
11  30   10;...
12  30   20;...
13  40   0;...
14  40   10;...
15  40   20];
% Nr  Gruppe  1  2  3  4
Element=[...
1  1  1  4  5  2;...
2  1  2  5  6  3;...
3  1  4  7  8  5;...
4  1  5  8  9  6;...
5  1  7  10  11  8;...
6  1  8  11  12  9;...
7  1  10  13  14  11;...
8  1  11  14  15  12];
% Nr  Dicke  E  nu  sz  n
Eigenschaft=[...
1  1  210000  0.3  0  2];
% Knoten  Richtung  Wert
Bindung=[...
1  1  0;...
2  2  0;...
2  1  0;...
3  1  0;...
];
% Element  Rand  Richtung  Wert1  Wert2
Randlast=[...
```

```
1  4  1  0   500;...
2  4  1  -500  0;...
7  2  1  -500  0;...
8  2  1  0   500];
```



(a) Knoten- und Elementnummern (b) Verschiebung (20fach überhöht)

Abbildung 12: Beispielsystem

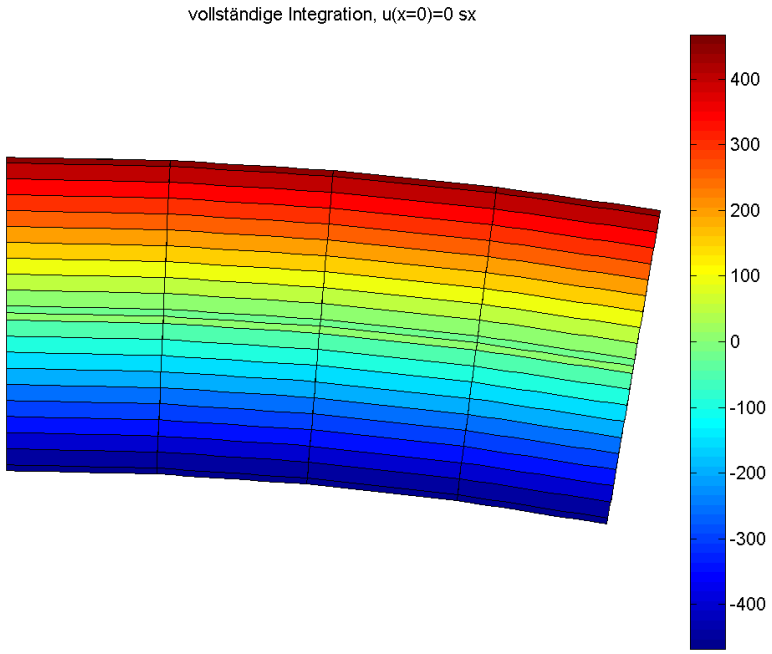


Abbildung 13: Aus den Verschiebungen berechnete Längsspannung σ_x für das Beispielsystem (keine extrapolierte und gemittelte Gausspunktwerte!)

5.2. Systembeschreibung

Für die Systembeschreibung sind folgende Variablen zu definieren;

`name` Bezeichnung des Systems, wird für die Diagrammbeschriftung und Erzeugung von Dateinamen verwendet.

`Knoten` Knotentabelle. Die Spalten bedeuten:

1. Knotennummer
2. x-Koordinate
3. y-Koordinate

`Element` Elementtabelle mit den Spalten

1. Elementnummer
2. Eigenschaftsnummer
3. Knotennummer Ecke 1
4. Knotennummer Ecke 2
5. Knotennummer Ecke 3
6. Knotennummer Ecke 4

`Eigenschaft` Eigenschaftstabelle mit den Spalten

1. Eigenschaftsnummer
2. Dicke
3. E-Modul
4. Querkontraktionszahl
5. 0 für ESZ, 1 für EVZ
6. Integrationsordnung (1 oder 2)

`Randlast` Randlasttabelle mit den Spalten

1. Elementnummer
2. Randnummer 1...4
3. Koordinate (1 entspricht x, 2 entspricht y)
4. Wert am Anfangsknoten
5. Wert am Endknoten

`Bindung` Bindungstabelle. Knotennummern mit zugehörigen Bindungen (Verschiebungsrandbedingungen). Nur die Knoten mit Bindungen müssen eingetragen werden.

1. Knotennummer
2. Koordinate (1 entspricht x, 2 entspricht y)
3. Vorgeschriebene Verschiebung (normalerweise 0, es können aber auch von Null verschiedene Verschiebungen aufgeprägt werden)

5.3. Das Skript FEM.m

Variablenliste

`a`	Zuordnungsmatrix der Elementfreiheitsgrade zu den Systemfreiheitsgraden, jede Zeile mit 8 Einträgen steht für ein Element.
`Bindung`	Bindungstabelle
`Ce`	Koordinaten der Elementmittelpunkte (Mittelwert der Eckknotenkoordinaten)
`Eigenschaft`	Elementgruppentabelle
`Elemente`	Elementtabelle
`F`	Systemlastvektor
`frei`	Liste der Systemfreiheitsgrade ohne Bindungen
`Knoten`	Knotentabelle
`name`	Frei wählbarer Name der Rechnung (Grundlage für Dateinamen und Diagrammbeschriftungen)
`ne`	Anzahl der Elemente
`nk`	Anzahl der Knoten
`nr`	Anzahl der Randlasten
`Randlast`	Randlasttabelle
`S`	Systemsteifigkeitsmatrix $\underline{S}^*$
`Se`	Elementsteifigkeitsmatrix $\underline{S}_e$
`U`	u-Werte für die Darstellung der Elementkanten
`u`	Systemverschiebungsvektor
`unfrei`	Liste der Systemfreiheitsgrade mit Bindungen
`Uscale`	Vergrößerungsfaktor der Verschiebungsdarstellung
`V`	v-Werte für die Darstellung der Elementkanten
`X`	x-Werte für die Darstellung der Elementkanten
`xg`	Globaler Koordinatenvektor $\underline{x}^*$
`Y`	y-Werte für die Darstellung der Elementkanten

Der erste Programmabschnitt bereitet die Daten auf und zeigt die Geometrie mit Knoten- und Elementnummern an (siehe Abbildung 12a). Besonders hingewiesen sei auf die Berechnung der Mittelwerte der Elementknotenkoordinaten als Position der Elementnummern.

Listing 20: FEM.m

```
% FEM 2D System
% Koordinatenvektor und Zuordnungsvektoren zu Elementen
nk=size(Knoten,1);
ne=size(Element,1);
nr=size(Randlast,1);
```

```
tmp=Knoten(:,2:3)';
xg=tmp(:);
a=[2*Element(:,3:6)-1   2*Element(:,3:6)];
% Anzeige der Elemente
X=xg(a(:,[1:4  1])');
Y=xg(a(:,[5:8  5])');
Ce=mean(reshape(Knoten(Element(:,3:6),2:3),[ne 4 2]),2);
clf;
plot(X,Y,'b');grid on;axis equal
text(Knoten(:,2),Knoten(:,3),num2str(Knoten(:,1)));
text(Ce(:,:,1),Ce(:,:,2),num2str(Element(:,1)));
title(name);
print('-dpng', [name '_geo.png']);
```

In einer Schleife über alle Elemente wird die Systemsteifigkeitsmatrix aufgebaut.

Listing 21: FEM.m

```
S=zeros(2*nk,2*nk);
for i=1:ne
    %Elementsteifigkeitsmatrix
    ae=a(i,:);
    p=Eigenschaft(Element(i,2),2:6);
    Se=Steifigkeit(xg(ae),p(1),p(2),p(3),p(4),p(5));
    % Einbau in globale Matrix
    S(ae,ae)=S(ae,ae)+Se;
end
```

In einer Schleife über alle Randlasten wird der Systemlastvektor aufgebaut.

Listing 22: FEM.m

```
F=0*xg;
% Randlasten
for i=1:nr
    r=Randlast(i,:);
    ie=r(1); % Element
    ir=r(2); % Rand
    ix=r(3); % Richtung
    gkx1=a(ie,ir);
    gkx2=a(ie,1+mod(ir,4));
    l=norm(xg([gkx2 gkx2+1])-xg([gkx1 gkx1+1]));
    ar=[gkx1 gkx2]+ix-1;
    t=Eigenschaft(Element(ie,2),2);
    F(ar)=F(ar)+l*t/6*[2 1; 1 2]*r(4:5)';
end
```

Einbau der Randbedingungen und Lösung des Gleichungssystems

Listing 23: FEM.m

```
unfrei=(Bindung(:,1)-1)*2+Bindung(:,2);
frei=setdiff(1:2*nk,unfrei);
% reduziertes Gleichungssystem lösen
u=0*xg;
u(unfrei)=Bindung(:,3);
u(frei)=S(frei,frei)\(F(frei)-S(frei,unfrei)*u(unfrei));
% Auflagerreaktionen berechnen
F(unfrei)=S(unfrei,:)*u;
```

Darstellung der Verschiebung, siehe Abbildung 12b.

Listing 24: FEM.m

```
Uscale=20;
U=u(a(:,[1:4 1])')*Uscale;
V=u(a(:,[5:8 5])')*Uscale;
clf;
plot(X,Y,'b',X+U,Y+V,'r');grid on;axis equal
title(name);
print('-dpng', [name '_disp.png']);
```

5.4. Das Skript Sichter.m

Das Skript `Sichter.m` führt die Nachlaufrechnung durch und erzeugt Farbverlaufbilder (*contour plots*). Die Größen werden aus dem Verschiebungszustand an einem engmaschigen Raster in den Elementen berechnet. Sie sind also keine extraplierte und an den Knoten gemittelte Gausspunktwerte! Ein Beispielbild (Längsspannung σ_x) zeigt Abbildung 13.

`vmin, vmax` Unter- und Obergrenze der auszugebenden Größe über alle Elemente. Sie werden für die Skalierung des Farbverlaufs benötigt, da jedes Element mit einem separaten `contourf`-Befehl geplottet wird.

Listing 25: Sichter.m

```
clf;
hold on
vmin=inf;
vmax=-inf;
for i=1:ne
    ae=a(i,:);
    p=Eigenschaft(Element(i,2),2:6);
    [re,se,xe,ye,ue,ve,sx,sy,txy,ex,ey,gxy]=...
        Nachlauf(xg(ae),u(ae),p(1),p(2),p(3),p(4),10);
    vd=eval(d);
    vmin=min(vmin,min(vd(:)));
    vmax=max(vmax,max(vd(:)));
    contourf(xe+ue*Uscale,ye+ve*Uscale,eval(d));
end
set(gca,'clim',[vmin vmax]);
```

```
xlim([min(X(:)+U(:)) max(X(:)+U(:))]);
ylim([min(Y(:)+V(:)) max(Y(:)+V(:))]);
title([name '_' d]);
colormap jet
colorbar;axis off equal;
hold off
```

Die Ergebnisgrößen eines Elements werden mit der Funktion Nachlauf berechnet, die in folgendem Listing dokumentiert ist. Zunächst wird das Koordinatenraster bereitgestellt und seine Form für alle Ergebnisse übernommen. Ortskoordinaten und Verschiebungen werden dann mit den Ansatzfunktionen aus den Knotenwerten interpoliert. Die Verzerrungen werden mit der in Listing 11 definierten Funktion `Verzerrung` berechnet. Bei der Spannungsberechnung wird zwischen EVZ und ESZ unterschieden. Die Argumente der Funktion sind

`xe`	Element-Koordinatenvektor $\underline{x}_e$
`ue`	Element-Verschiebungsvektor $\underline{u}_e$
`t`	Element-Dicke, wird aktuell nicht benötigt.
`E`	E-Modul
`nu`	Querkontraktionszahl
`fs`	0: ESZ, 1: EVZ
`n`	Integrationsordnung, wird aktuell nicht benötigt.

Listing 26: Nachlauf.m

```
function [r,s,x,y,u,v,sx,sy,txy,ex,ey,gxy]=Nachlauf(xe,ue,t,E,nu,fs,n)
[r,s]=meshgrid(-1:(2/n):1,-1:(2/n):1);
% Form der Koordinatenmatrix übernehmen
x=r;y=r;u=r;v=r;ex=r;ey=r;gxy=r;sx=r;sy=r;txy=r;
x(:)=h41(r(:),s(:))*xe(1:4);
y(:)=h41(r(:),s(:))*xe(5:8);
u(:)=h41(r(:),s(:))*ue(1:4);
v(:)=h41(r(:),s(:))*ue(5:8);
[ex,ey,gxy]=Verzerrung(xe,ue,r,s);
if fs==0 % ESZ
    C=E/(1-nu^2)*[1 nu 0; nu 1 0; 0 0 (1-nu)/2];
else % EVZ
    l=E*nu/((1+nu)*(1-2*nu));
    m=E/(2+2*nu);
    C=[2*m+l l 0; l 2*m+l 0; 0 0 m];
end
R=C*[ex(:) ey(:) gxy(:)]';
sx(:)=R(1,:);
sy(:)=R(2,:);
txy(:)=R(3,:);
```

A. Erste Schritte mit Matlab

Matlab ist auf den Rechnern im CAD-Labor IWZ 210 installiert. Aufgerufen wird es entweder über das Startmenü oder das Desktop Icon.

Die Benutzeroberfläche von Matlab heißt *Desktop*. Der Desktop besteht aus unter- und nebeneinander angeordneten Fenstern, deren Anordnung vom Nutzer sehr frei gestaltet werden kann.

Den Standardzustand stellt man her mit **Hauptmenu** > **Desktop** > **Desktop Layout** > **Default**. In diesem Zustand zeigt der Desktop die Fenster

- Command Window (Kommandozeile)
- Command History (Liste der bisher eingegebenen Kommandos)
- Workspace (Anzeige definierter Variablen)
- Current Directory (Inhalt des aktuellen Verzeichnisses)

Weitere häufig benutzte Fenster sind

- Editor (Bearbeiten von Kommandodateien, sogenannten .m-Files)
- Help (Dokumentation und Hilfe, in Englisch)

A.1. Arbeitsweise mit Matlab

Befehlseingabe an der Kommandozeile:

- Befehl eintippen und mit **Enter** abschließen.
- Steht am Ende ein Semikolon (;), wird der Befehl ausgeführt, aber das Ergebnis nicht angezeigt.
- Mit **Aufwärts-Pfeiltaste** holt man sich vorige Eingaben an die Kommandozeile zurück und kann sie dann editieren.

Befehlseingabe aus einer Datei

- Datei muss die Endung .m haben und im aktuellen Verzeichnis liegen.
- Dateiname ohne Endung an der Kommandozeile als Befehl ausführen. Alle Kommandos in der Datei werden dann abgearbeitet. Eventuelle Zwischenergebnisse werden angezeigt, wenn die zugehörigen Zeilen nicht mit ; abgeschlossen wurden.
- Zeilen, die mit % beginnen, sind Kommentare.
- Datei im Editor öffnen und ausführen mit Run (in der Werkzeugleiste des Editors

Befehlseingabe aus einer Datei mit Zellen

- Zellen (Cells) sind Abschnitte in einer Datei, die mit **%%** und einem Leerzeichen beginnen.
- Der Text nach **%%** und dem Leerzeichen wird als Überschrift interpretiert.
- Zeilen, die mit **%** beginnen, sind Kommentare.

- Die aktuelle Zelle (wo der Cursor steht) ist gelb hinterlegt.
- Die Befehle in der aktuellen Zelle können mit **Strg-Enter** ausgeführt werden.
- Ein Protokoll kann erstellt werden mit **Hauptmenü > File > Publish...**

B. Erste Schritte mit Octave

Mit den Stichwörtern „gnu octave" findet man im Internet schnell den passenden Installer. Octave ist auf Linux zu Hause und wird mit vielen Distributionen als Paket mitgeliefert. Auch für Windows gibt es einen Installer. Dieser installiert im Startmenü die folgenden Punkte:

- Dokumentation in HTML und PDF. Hier darf man keine Wunder erwarten, die Beschreibung beschränkt sich häufig auf die Angabe der Aufrufsyntax von Funktionen. Hier hilft der Zugriff auf eine Matlab-Dokumentation. Im Zweifelsfall muss man dann prüfen, ob die geplante Vorgehensweise in Octave realisierbar ist.
- Notepad++ als Editor. In der dem Autor vorliegenden Installation funktioniert der Startmenü-Eintrag nicht. Von der Octave-Konsole kann er jedoch mit dem Befehl `edit` aufgerufen werden.
- Octave, die interaktive Konsole (Kommandozeile). Sie kann auch über das Desktop-Icon aufgerufen werden.
- WGnuplot, das Grafik-Rückgrat von Octave.

B.1. Anpassungen nach der Installation unter Windows

Hat man beim Installieren alle Pakete angewählt, funktioniert die Grafik nicht. Das liegt am sogenannten „oct2mat-Problem", einem Paket, dass sich nicht mit der Grafik verträgt. An der Oktave-Konsole wird es abgeschaltet mit

```
pkg rebuild -noauto oct2mat
```

B.2. Dateien und Verzeichnisse

Nach dem Start ist das **aktuelle Verzeichnis** (anzuzeigen mit `pwd`) das Verzeichnis, in dem die ausführbare Octave-Datei liegt, beispielsweise *C:\Programme\Octave\3.2.4_gcc-4.4.0\bin*.

Octave ist auch das **Home-Verzeichnis** '~' bekannt, beispielsweise

```
> cd '~'
> pwd
ans = C:\Dokumente und Einstellungen\kraska
```

Dort kann man eine Datei *.octaverc* anlegen, die immer beim Start von Octave ausgeführt wird. Beispielsweise kann man in das Verzeichnis des aktuellen Projekts wechseln, indem man dort einträgt:

```
cd 'C:\Temp\FH Brandenburg Kraska\Octave'
```

Im Home-Verzeichnis findet sich auch die Datei *.octave_hist*, in der die Eingabe protokolliert wird.

B.3. Links

Einige Hinweise zur Installation und zum Test auf Mac und PC findet man unter http://cds130.org/Introduction_To_Octave.

Ein FEM-Programm, realisiert in Octave findet man unter https://staff.ti.bfh.ch/sha1/pwf/fem/FEMoctave/, einen Artikel dazu unter Paper

Eine Übersicht über Octave-Programme Dritter http://econpapers.repec.org/software/codoctave/

Eine Liste des Caltec zu kommerziellen und freien FEM-Programmen Caktec FEM-Liste

Online-Octave-Handbuch von 2008, das es auch in gedruckter Form gibt http://www.network-theory.co.uk/docs/octave3/index.html