Berechnung von Eigenfrequenzen mit FELyX

Tacoma Narrows Bridge, USA, 7. November 1940 [10]

Diplomarbeit
WS 2002 / 2003

Autor:
Boris Meier <meierbo@student.ethz.ch>

Betreuer:
Marc Wintermantel <wintermantel@imes.mavt.ethz.ch>

Version 3
7. Februar 2003

Zentrum für Strukturtechnologien
03-050

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

IMES
Institut für
Mechanische Systeme

Zusammenfassung

Das C++-Programm *FELyX* soll dahingehend erweitert werden, dass Eigenfrequenzen und Eigenschwingungsformen von Strukturen berechnet werden können. Dafür ist das allgemeine Eigenwertproblem $\mathbf{K}\mathbf{u}_i = \omega_i^2\mathbf{M}\mathbf{u}_i$ zu lösen, wobei ω_i die Eigenfrequenzen und $\mathbf{u}_i$ die dazugehörigen Eigenvektoren, die die Eigenschwingungsformen beschreiben, bedeuten.

Die Generierung der Steifigkeitsmatrix $\mathbf{K}$ ist in *FELyX* bereits implementiert, da sie auch für statische Analysen benötigt wird. Die Bildung der Massenmatrix $\mathbf{M}$ ist zu programmieren. Sie geschieht ähnlich wie die Bildung von $\mathbf{K}$.

$\mathbf{K}$ und $\mathbf{M}$ sind grosse Matrizen, deren Ordnung gleich der Anzahl Freiheitsgrade der in finite Elemente unterteilten Struktur ist. Sie weisen eine Bandstruktur auf, so dass die meisten Einträge Null sind. Dies erlaubt ein effizientes Speichern der Matrizen und ermöglicht schnelle Rechenalgorithmen.

In einem ersten Versuch wird das allgemeine Eigenwertproblem auf das spezielle Eigenwertproblem $\mathbf{C}\mathbf{u}_i = \omega_i^2\mathbf{u}_i$ reduziert. Ein spezielles Eigenwertproblem ist einfacher zu lösen. Es gibt OpenSource Algorithmen dazu. Bei der Reduktion geht jedoch die Bandstruktur der Matrizen $\mathbf{K}$ und $\mathbf{M}$ verloren, so dass der Speicheraufwand für $\mathbf{C}$ sehr gross wird.

In einem weiteren Versuch wird die Methode der simultanen Vektoriteration implementiert. Sie eignet sich, um die p kleinsten Eigenwerte und die dazugehörigen Eigenvektoren des allgemeinen Eigenwertproblems zu berechnen. Es werden dabei gleichzeitig p verschiedene Vektoren iteriert, bis sie gegen die gesuchten Eigenvektoren konvergieren. Diese Methode erweist sich in der Implementierung als ineffizient.

Zum Ziel führt schliesslich das Verfahren von Lanczos mit dem inversen Eigenwertproblem. Damit können die p kleinsten Eigenwerte und ihre Eigenvektoren effizient bestimmt werden. Es werden typischerweise 8 bis 20 Lanczos Schritte ausgeführt, wobei in jedem Schritt ein Gleichungssystem $\mathbf{K}\mathbf{u} = \mathbf{h}$ nach $\mathbf{u}$ gelöst werden muss. Da die Matrix $\mathbf{K}$ konstant bleibt kann dies so geschehen, dass $\mathbf{K}$ anfangs einmal zerlegt und in jedem Schritt dann nur noch einmal rückwärts eingesetzt wird. Der gesamte Rechenaufwand hält sich so in Grenzen.

Der gefundene Rechenablauf ist effizient und kann mit kommerziellen Eigensolvern mithalten. Die Generierung der Massenmatrix könnte hingegen noch beschleunigt werden, wenn $\mathbf{K}$ und $\mathbf{M}$ gleichzeitig und nicht nacheinander aufgebaut würden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Eigenschwingungen

Die wohl einfachste Form einer Eigenschwingung ist ein Feder-Masse-System mit einer Masse und einer Feder (Abb. 1.1). Versetzt man die Masse in Bewegung, verschiebt sich ihre kinetische Energie in Federenergie, bis die Masse in extremaler Position stillsteht. Danach wird die Federenergie wieder in kinetische Energie umgewandelt. Diese Energieumwandlung ist die Basis jeder Schwingung. Falls das System nicht von aussen angeregt wird, verkleinert sich die Amplitude der Schwingung mit der Zeit aufgrund von Dämpfung. Diese tritt in der Realität immer auf und ist eine Folge der Reibung zwischen den Atomen. Bei der beschriebenen Energieumwandlung wird also ein Teil der Engergie immer auch in Wärme umgewandelt.

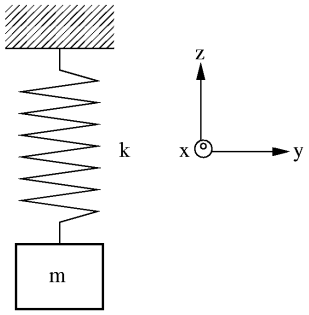

Abbildung 1.1: Einfaches Feder-Masse-System

Jeder Körper, jede Struktur ist eigentlich nichts anderes als ein komplexes System aus Federn und Massen (und Dämpfern). Auf unterster Ebene können Atmokerne durch Massepunkte und Elementarkräfte durch Federkräfte modelliert werden. Soweit geht man in der Ingenieurspraxis allerdings nicht. Sondern man unterteilt eine Struktur in kleine Einheiten, in finitie Elemente. Diese Elemente haben eine Masse und auf sie wirken Kräfte, was wiederum eine Art Feder-Masse-System bedeutet.

Überlässt man das angeregte System aus Abb. 1.1 sich selber, schwingt es mit einer klar definierten Frequenz weiter, der Eigenfrequenz f des Systems. Sie berechnet sich aus dem Impulssatz und ergibt $f = \sqrt{m/k}$. Hängt man an die Masse eine zweite Feder mit einer weiteren Masse, hat das neue System zwei Eigenfrequenzen, sofern man nur Schwingungen in z-Richtung betrachtet.

Betrachtet man alle Schwingungen, die das System aus Abb. 1.1 ausführen kann, kommt man auf sechs Eigenschwingungsformen. Die Masse kann nämlich auch in x- oder y-Richtung schwingen (pendeln) und um alle drei Achsenrichtungen rotieren. Beim System mit zwei Massen ergeben sich so total zwölf Eigenschwingungsformen mit je einer Eigenfrequenz. Jede beliebige Schwingung dieses Systems ist eine Superposition der Eigenschwingungsformen.

Die Anzahl Schwingungsformen und Eigenfrequenzen eines beliebigen Systems ist also die Summe der Massen multipliziert mit der Anzahl ihrer jeweiligen Freiheitsgrade. Für einen in finite Elemtente unterteilten Körper ergeben sich so exorbitant viele Eigenfrequenzen. Relevant sind in der Praxis jedoch nur die tiefsten.

1.2 Praktische Relevanz der Eigenfrequenzen

Regt man das Feder-Masse-System aus Abb. 1.1 mit einer der Eigenfrequenz ähnlichen Frequenz an, gerät das System in Resonanz. Die Amplitude des Schwingung kann sehr gross werden. In der Praxis kann dies gefährlich oder zumindest unangenehm sein.

Die Eigenfrequenzen eines Eisenbahnwagens zum Beispiel sollen nicht durch Unwuchten, die in jedem Rad auftreten, angeregt werden können. Bei der Fahrt mit einer bestimmten Geschwindigkeit würde so der Wagen zu schwingen beginnen, was für die Passagiere unangenehm wäre. Damit eine Eigenfrequenz nicht wesentlich angeregt wird, muss sie mindestens etwa vier mal höher sein als die Frequenz der Anregung [2].

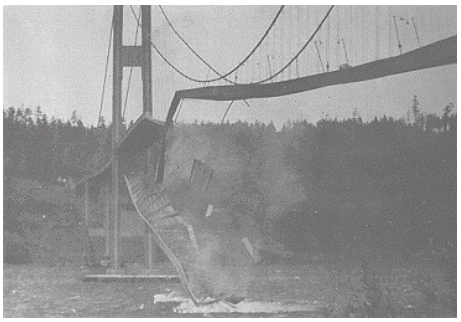

Abbildung 1.2: Tacoma Narrows Bridge [10]

Welch dramatische Folgen eine Vernachlässigung der Betrachtung der Eigenfrequenzen haben kann, zeigt der berühmte Fall der Tacoma Narrows Bridge (Abb. 1.2), welche am 7. November 1940 in Folge starker Schwingungen einstürzte. Bei der Planung war die Ablösung von Wirbeln bei Wind nicht genügend berücksichtigt worden. Die Windgeschwindigkeit an jenem Tag versetzte die Brücke in Resonanzschwingung, welche schliesslich zum katastrophalen Versagen führte.

1.3 Eigenfrequenzoptimierung

Das Ziel der Eigenfrequenzoptimierung einer Struktur ist es meist, die tiefsten Eigenfrequenzen so hoch wie möglich zu machen. Dies kann zum Beispiel gelingen, indem man

die Struktur steifer macht oder ein Material mit höherem Elastizitätsmodul wählt.

Um eine Optimierung zu illustrieren, dient folgendes Beispiel. Es wird eine fachwerkartige Brückenstruktur betrachtet (Abb. 1.3 - 1.5). Die drei Konstruktionsvorschläge haben je das gleiche Gewicht, das heisst die Stäbe haben bei den Vorschlägen verschieden grosse Querschnitte. Der obere und untere Biegebalken sei aber gleich.

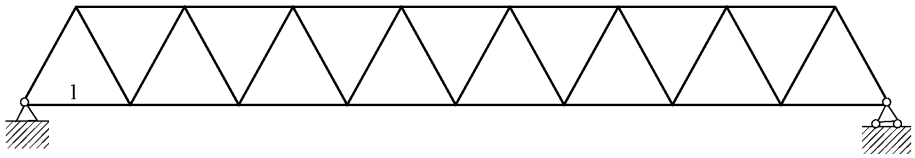

Abbildung 1.3: Struktur 1

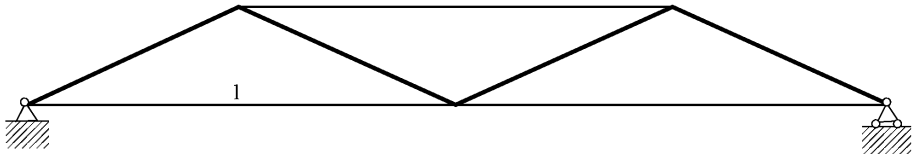

Abbildung 1.4: Struktur 2

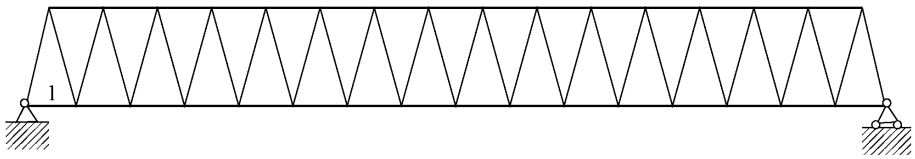

Abbildung 1.5: Struktur 3

Die Spannweite der Brćke sei 8 m, der Abstand zwischen den Biegebalken $\sqrt{3}$ m. Das Material hat eine Dichte ρ von 2700 kg/m^3 und einen E-Modul von 70 GPa. Die Balken haben eine Querschnittsflćhe von 16 cm^2 und ein Flächenträgheitsmoment I von $2{\cdot}10^{-6}$ m^4. Die unterschiedlichen Parameter der drei Strukturen sind in Tabelle 1.1 zusammengefasst, wobei l die horizontale Distanz zwischen zwei Fachwerkknoten und GL die Gesamtlänge der Stäbe bedeutet.

Struktur	l [m]	GL [m]	A [cm^2]	I [10^{-6} m^4]	f_1 [Hz]
1	2	16.00	16.00	2.00	54.33
2	4	10.58	24.19	3.02	29.67
3	1	57.69	4.44	0.55	31.68

Tabelle 1.1: Parameter der Strukturen

Die berechneten tiefsten Eigenfrequenzen f_1 der Strukturen sind ebenfalls in Tabelle 1.1 ersichtlich. Die Struktur 1 hat die höchste primäre Eigenfrequenz. Das ist insofern erstaunlich, als dass die Struktur ein “Kompromiss” zwischen den Strukturen 2 und 3 ist. Intuitiv erkennt man, dass es einen “optimalen Kompromiss” zwischen 2 und 3 geben muss. Abbildung 1.6 zeigt den qualitativen Verlauf der tiefsten Eigenfrequenz in Abhängigkeit von l.

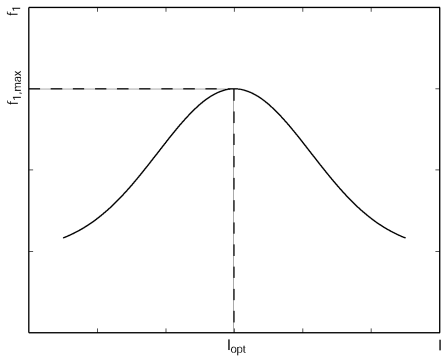

Abbildung 1.6: Tiefste Eigenfrequenz in Abängigkeit von l

Dieses Beispiel ist eine typische einfache Optimierungsaufgabe mit einzigem Parameter l. Komplizierte Optimierungsaufgaben haben mehrer Paramter. Es gibt verschiedene Optimierungsstrategien, wie zum Beispiel *Genetische Algorithmen* oder *Simulated Annealing*. Optimierung ist aber nicht Aufgabe dieser Arbeit, sondern die effiziente Berechnung von Eigenfrequenzen.

Kapitel 2

Grundsätzliche Überlegungen

Die Bestimmung der Eigenfrequenzen eines Bauteils unter Verwendung finiter Elemente soll am einfachen Beispiel eines einseitig eingespannten Balkens (Abb. 2.1) mit Elementlänge l und Querschnittsfläche b^2 untersucht werden. Für diese zweidimensionale Aufgabe werden Beam3-Elemente verwendet. Das Vorgehen für komplexere Strukturen ist analog.
Die Parameter des Beispielproblems sind in Tabelle 2.1 zusammengefasst.

L	4	m
b	4	cm
ρ	2700	kg/m^3
E	$8 \cdot 10^{10}$	Pa

Tabelle 2.1: Parameter des Problems

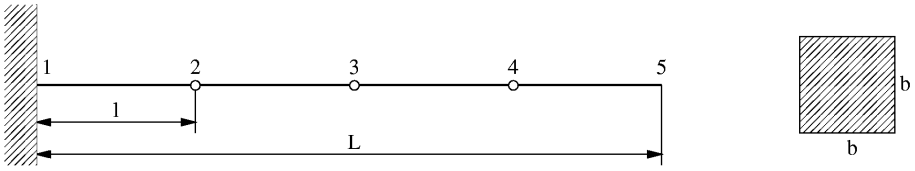

Abbildung 2.1: Einseitig eingespannter Balken mit 4 Elementen

2.1 Die Steifigkeitsmatrix

Die Steifigkeitselementmatrix $\mathbf{k}$ für ein Beam3-Element ist [9]

$$\mathbf{k} = \frac{E}{l^3} \begin{bmatrix} Al^2 & 0 & 0 & -Al^2 & 0 & 0 \\ 0 & 12I & 6Il & 0 & -12I & 6Il \\ 0 & 6Il & 4Il^2 & 0 & -6Il & 2Il^2 \\ -Al^2 & 0 & 0 & Al^2 & 0 & 0 \\ 0 & -12I & -6Il & 0 & 12I & -6Il \\ 0 & 6Il & 2Il^2 & 0 & -6Il & 4Il^2 \end{bmatrix} \tag{2.1}$$

Setzt man die Parameter des Problems ein, erhält man

$$\mathbf{k} = 10^6 \cdot \begin{bmatrix} 128 & 0 & 0 & -128 & 0 & 0 \\ 0 & 0.205 & 0.102 & 0 & -0.205 & 0.102 \\ 0 & 0.102 & 0.0683 & 0 & -0.102 & 0.0341 \\ -128 & 0 & 0 & 128 & 0 & 0 \\ 0 & -0.205 & -0.102 & 0 & 0.205 & -0.102 \\ 0 & 0.102 & 0.0341 & 0 & -0.102 & 0.0683 \end{bmatrix} \tag{2.2}$$

Die globale Steifigkeitsmatrix setzt sich aus den Elementsteifigkeitsmatrizen zusammen, welche sich gegenseitig überlappen (Abb. 2.2). Die Überlappung beruht darauf, dass ein Knoten jeweils zu zwei Elemten gehört. Weil der Balken sich im Knoten 1 weder verschieben noch verdrehen kann, werden die ersten drei Zeilen und Spalten der globalen Steifigkeitsmatrix, welche den drei Freiheitsgraden des ersten Knoten entsprchen, gestrichen. Damit wird die Matrix **K** positiv definit.

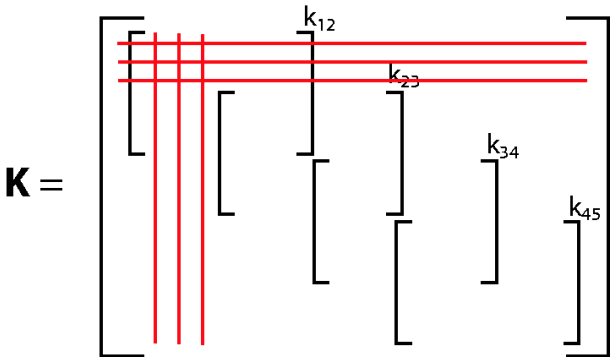

Abbildung 2.2: Bildung der globalen Steifigkeitsmatrix

Gleichung 2.3 zeigt die globale Steifigkeitsmatrix. Sie ist symmetrisch mit deutlicher Bandstruktur. Dank diesen Eigenschaften kann die Matrix effizient gespeichert werden, was bei Matrizen höherer Ordnung wichtig wird. Ebenfalls erlaubt die Speicherung im Enveloppe-Format effiziente Rechenalgorithmen.

$$\mathbf{K} = 10^6 \cdot \begin{bmatrix} 260 & 0 & 0 & -130 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.41 & 0 & 0 & -0.2 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.14 & 0 & -0.1 & 0.034 & 0 & 0 & 0 & 0 & 0 & 0 \\ -130 & 0 & 0 & 260 & 0 & 0 & -130 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.2 & -0.1 & 0 & 0.41 & 0 & 0 & -0.2 & 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0.034 & 0 & 0 & 0.14 & 0 & -0.1 & 0.034 & 0 & 0 & 0 \\ 0 & 0 & 0 & -130 & 0 & 0 & 260 & 0 & 0 & -130 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.2 & -0.1 & 0 & 0.41 & 0 & 0 & -0.2 & 0.1 \\ 0 & 0 & 0 & 0 & 0.1 & 0.034 & 0 & 0 & 0.14 & 0 & -0.1 & 0.034 \\ 0 & 0 & 0 & 0 & 0 & 0 & -130 & 0 & 0 & 130 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.2 & -0.1 & 0 & 0.2 & -0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.034 & 0 & -0.1 & 0.068 \end{bmatrix} \tag{2.3}$$

2.2 Die Massensmatrix

Die Massenelementmatrix **m** für ein Beam3-Element ist in Abschnitt 3.3 definiert. Setzt man die Parameter aus Tabelle 2.1 ein, erhält man

$$\mathbf{m} = \begin{bmatrix} 1.44 & 0 & 0 & 0.72 & 0 & 0 \\ 0 & 1.6 & 0.226 & 0 & 0.555 & -0.134 \\ 0 & 0.226 & 0.0411 & 0 & 0.134 & -0.0309 \\ 0.72 & 0 & 0 & 1.44 & 0 & 0 \\ 0 & 0.555 & 0.134 & 0 & 1.6 & -0.226 \\ 0 & -0.134 & -0.0309 & 0 & -0.226 & 0.0411 \end{bmatrix} \tag{2.4}$$

Die Bildung der globalen Massenmatrix **M** geschieht genau analog zur Bildung der globalen Steifigkeitsmarix. Sie ist ebenfalls symmetrisch und positiv definit und kann in der gleichen Hüllenform wie **K** gespeichert werden.

$$\mathbf{M} = \begin{bmatrix} 2.9 & 0 & 0 & 0.72 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.2 & 0 & 0 & 0.56 & -0.13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.082 & 0 & 0.13 & -0.031 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.72 & 0 & 0 & 2.9 & 0 & 0 & 0.72 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.56 & 0.13 & 0 & 3.2 & 0 & 0 & 0.56 & -0.13 & 0 & 0 & 0 \\ 0 & -0.13 & -0.031 & 0 & 0 & 0.082 & 0 & 0.13 & -0.031 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.72 & 0 & 0 & 2.9 & 0 & 0 & 0.72 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.56 & 0.13 & 0 & 3.2 & 0 & 0 & 0.56 & -0.13 \\ 0 & 0 & 0 & 0 & -0.13 & -0.031 & 0 & 0 & 0.082 & 0 & 0.13 & -0.031 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.72 & 0 & 0 & 1.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.56 & 0.13 & 0 & 1.6 & -0.23 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.13 & -0.031 & 0 & -0.23 & 0.041 \end{bmatrix} \tag{2.5}$$

2.3 Eigenfrequenzen

Die globale Steifigkeits- und Massenmatrix erlauben nun die Eigenfrequenzen und Eigenschwingungsformen des Balkens zu berechnen. Gleichung 2.6 beschreibt den Zusammenhang zwischen den erwähnten Matrizen und den Eigenfrequenzen bzw. Eigenschwingungsformen. Zu lösen ist ein allgemeines Eigenwertproblem.

$$\mathbf{K}\mathbf{u}_i = \omega_i^2 \mathbf{M}\mathbf{u}_i \tag{2.6}$$

Multipliziert man die Gleichung beidseitig mit $\mathbf{M}^{-1}$ erhält man das spezielle Eigenwertproblem in Gleichung 2.7.

$$(\mathbf{M}^{-1}\mathbf{K})\mathbf{u}_i = \omega_i^2 \mathbf{u}_i \tag{2.7}$$

Die Eigenfrequenzen ω_i des Systems erhält man also, indem man aus den Eigenwerten der Matrix $\mathbf{M}^{-1}\mathbf{K}$ die Wurzeln zieht. Da die Matrizen **K** und **M** positiv definit sind, sind alle Eigenwerte positiv.

$$\omega_i^2 = \mathrm{eig}(\mathbf{M}^{-1}\mathbf{K}) \tag{2.8}$$

Als tiefste zwei Eigenfrequenzen erhält man

$$\omega_1 = 13.8127 \qquad \omega_2 = 86.6605 \tag{2.9}$$

Die Einheit ist rad/s. Will man die Eigenfrequenzen in Hertz ausdrücken, wie sie z.B. *Ansys* als Resultat ausgibt, sind die Eigenkreisfrequenzen ω_i in Gleichung 2.9 noch durch 2π zu dividieren.

$$f_i = \frac{\omega_i}{2\pi} \tag{2.10}$$

Vergleicht man die berechneten Eigenfrequenzen mit den exakten Werten [2] in Gleichungen 2.11, stellt man vor allem für die tiefste Eigenfrequenz eine hohe Genauigkeit fest. In je mehr finite Elemente der Balken unterteilt wird, desto weiter nähert sich das Ergebnis den exakten Werten an.

$$\omega_1 = 3.516\sqrt{\frac{EI}{mL^3}} = 13.8122 \qquad \omega_2 = 22.03\sqrt{\frac{EI}{mL^3}} = 86.5420 \tag{2.11}$$

2.4 Eigenschwingungsformen

Der zum Eigenwert ω_i^2 gehörende Eigenvektor $\mathbf{u}_i$ beschreibt die Form der Eigenschwingung. Beim Balkenproblem stehen in den Eigenvektoren die Verschiebungen der Knoten in horizontale und vertikale Richtung (u bzw. v) sowie die Verdrehung gegenüber der Horizontalen (α).

$$\mathbf{u}_i = \begin{bmatrix} u_2 & v_2 & \alpha_2 & u_3 & v_3 & \alpha_3 & u_4 & v_4 & \alpha_4 & u_5 & v_5 & \alpha_5 \end{bmatrix}^T \tag{2.12}$$

Die zu den tiefsten zwei Eigenfrequenzen gehörenden Eigenvektoren sind

$$\begin{aligned} \mathbf{u}_1 &= \begin{bmatrix} 0 & 0.07 & 0.13 & 0 & 0.25 & 0.21 & 0 & 0.48 & 0.24 & 0 & 0.72 & 0.25 \end{bmatrix}^T \\ \mathbf{u}_2 &= \begin{bmatrix} 0 & -0.2 & -0.27 & 0 & -0.34 & 0.054 & 0 & -0.064 & 0.46 & 0 & 0.48 & 0.57 \end{bmatrix}^T \end{aligned} \tag{2.13}$$

In Abbildungen 2.3 und 2.4 sind die dazugehörigen Eigenschwingungsformen gezeichnet. In den Eigenvektoren (Gl. 2.14) erkennt man, dass die Verschiebungen der Knoten in horizontale Richtung sehr klein sind.

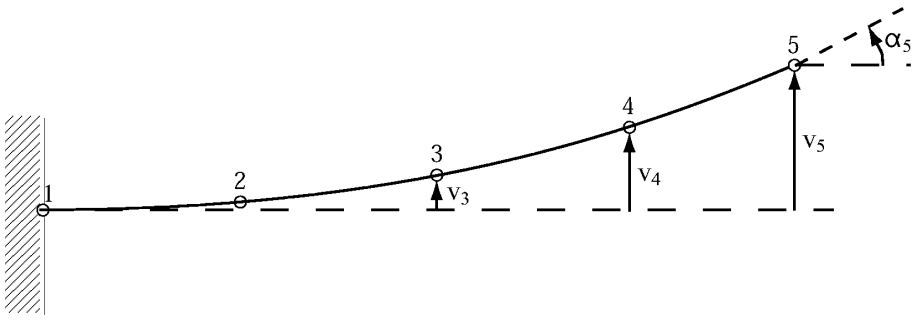

Abbildung 2.3: 1. Eigenschwingungsform des Balkens

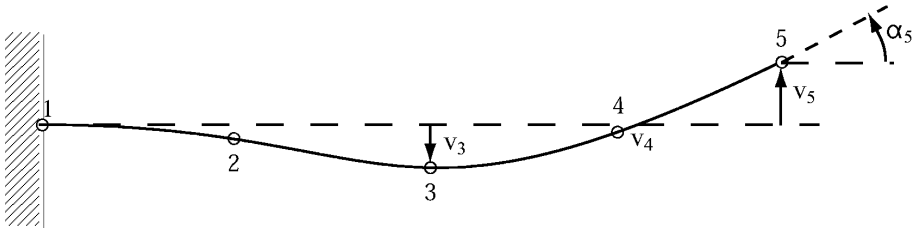

Abbildung 2.4: 2. Eigenschwingungsform des Balkens

Die Eigenvektoren können beliebig skaliert werden. Sie enthalten keine absoluten Werte, sondern beschreiben lediglich das *Verhältnis* der maximalen Auslekungen zueinander. Sie können bezüglich irgendeiner Norm normiert werden.

Kapitel 3

Massenmatrix

Dieses Kapitel behandelt das Aufstellen der Massenmatrix für die verschiedenen Elementtypen. Bei den einfacheren Elementtypen kann die Elementmassenmatrix in geschlossener Form angegeben werden. Bei komplexeren Elementtypen muss die Elementmassenmatrix numerisch an verschiedenen Integrationspunkten aufintegriert werden (Abschn. 3.11).

3.1 Link1

Das Link1-Element modelliert einen zweidimensionalen Zug- oder Druckstab. Es kann keine Biegelast aufnehmen.
Die Massenmatrix des Link1-Elements lässt sich geschlossen darstellen:

$$\mathbf{m} = \frac{1}{6} Al\rho \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \tag{3.1}$$

3.2 Link8

Das Link8-Element modelliert einen dreidimensionalen Zug- oder Druckstab. Es kann keine Biegelast aufnehmen.
Die Massenmatrix des Link8-Elements lässt sich geschlossen darstellen:

$$\mathbf{m} = \frac{1}{6} Al\rho \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{3.2}$$

3.3 Beam3

Das Beam3-Element modelliert einen zweidimensionalen Biegebalken. Es kann nebst Zug und Druck auch eine Biegelast aufnehmen.

Die Massenmatrix des Beam3-Elements lässt sich geschlossen darstellen [9]:

$$\mathbf{m} = Al\rho \begin{pmatrix} \frac{1}{3} & 0 & 0 & \frac{1}{6} & 0 & 0 \\ 0 & A & C & 0 & B & -D \\ 0 & C & E & 0 & D & F \\ \frac{1}{6} & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & B & D & 0 & A & -C \\ 0 & -D & F & 0 & -C & E \end{pmatrix} \tag{3.3}$$

Die Bedeutung der Einträge steht in den Gleichungen 3.5.

3.4 Beam4

Das Beam4-Element modelliert einen dreidimensionalen Biegebalken. Es kann nebst Zug und Druck auch eine Biegelast in zwei Achsenrichtungen sowie Torsion aufnehmen.
Die Massenmatrix des Beam4-Elements lässt sich geschlossen darstellen [7]:

$$\mathbf{m} = Al\rho \begin{pmatrix} \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & 0 & 0 & 0 & 0 \\ 0 & A_z & 0 & 0 & 0 & C_z & 0 & B_z & 0 & 0 & 0 & -D_z \\ 0 & 0 & A_y & 0 & -C_y & 0 & 0 & 0 & B_y & 0 & D_y & 0 \\ 0 & 0 & 0 & \frac{I_y+I_z}{3A} & 0 & 0 & 0 & 0 & 0 & \frac{I_y+I_z}{6A} & 0 & 0 \\ 0 & 0 & -C_y & 0 & E_y & 0 & 0 & 0 & -D_y & 0 & F_y & 0 \\ 0 & C_z & 0 & 0 & 0 & E_z & 0 & D_z & 0 & 0 & 0 & F_z \\ \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & B_z & 0 & 0 & 0 & D_z & 0 & A_z & 0 & 0 & 0 & C_z \\ 0 & 0 & B_y & 0 & -D_y & 0 & 0 & 0 & A_y & 0 & -C_y & 0 \\ 0 & 0 & 0 & \frac{I_y+I_z}{6A} & 0 & 0 & 0 & 0 & 0 & \frac{I_y+I_z}{3A} & 0 & 0 \\ 0 & 0 & D_y & 0 & F_y & 0 & 0 & 0 & -C_y & 0 & E_y & 0 \\ 0 & -D_y & 0 & 0 & 0 & F_z & 0 & C_z & 0 & 0 & 0 & E_z \end{pmatrix} \tag{3.4}$$

Wobei gilt

$$\begin{aligned} A_i &= \frac{6}{5}\frac{I_i}{Al^2} + \frac{13}{35} \\ B_i &= -\frac{6}{5}\frac{I_i}{Al^2} + \frac{9}{70} \\ C_i &= \frac{1}{10}\frac{I_i}{Al} + \frac{11l}{210} \\ D_i &= \frac{1}{10}\frac{I_i}{Al} - \frac{13l}{420} \\ E_i &= \frac{2}{15}\frac{I_i}{A} + \frac{l^2}{105} \\ F_i &= -\frac{1}{30}\frac{I_i}{A} - \frac{l^2}{140} \end{aligned} \tag{3.5}$$

und $i = y, z$.

3.5 Plane2

Das Plane2-Element ist ein zweidimensionales Dreieckelement mit sechs Knoten (Abb. 3.1).

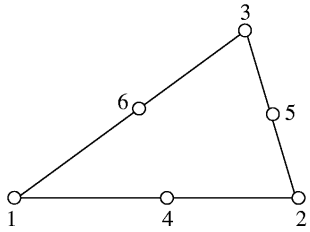

Abbildung 3.1: Plane2 Element

Die Formfunktionen sind [8]:

$$\begin{aligned} N_1 &= (2L_1 - 1)L_1 \\ N_2 &= (2L_2 - 1)L_2 \\ N_3 &= (2L_3 - 1)L_3 \\ N_4 &= 4L_1L_2 \\ N_5 &= 4L_2L_3 \\ N_6 &= 4L_1L_3 \end{aligned} \tag{3.6}$$

wobei L_1, L_2, und L_3 die Dreieckskoordinaten sind [8].

3.6 Plane182

Das Plane182-Element ist ein zweidimensionales Viereckelement mit vier in den Ecken angeordneten Knoten (Abb. 3.2).

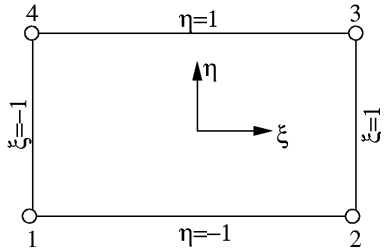

Abbildung 3.2: Plane182 Element

Die Formfunktionen sind [8]:

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\ N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\ N_4 &= \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{3.7}$$

3.7 Plane183

Das Plane183-Element ist ein zweidimensionales Viereckelement mit acht Knoten. Diese sind in den Ecken und an den Seitenmitten angeordnet (Abb. 3.3).

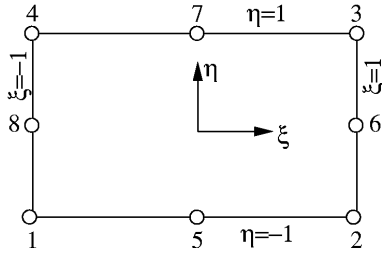

Abbildung 3.3: Plane183 Element

Die Formfunktionen sind [8]:

$$
\begin{aligned}
N_1 &= \frac{1}{4}(1-\xi)(1-\eta)(-\xi-\eta-1) \\
N_2 &= \frac{1}{4}(1+\xi)(1-\eta)(\xi-\eta-1) \\
N_3 &= \frac{1}{4}(1+\xi)(1+\eta)(\xi+\eta-1) \\
N_4 &= \frac{1}{4}(1-\xi)(1+\eta)(-\xi+\eta-1) \\
N_5 &= \frac{1}{2}(1-\xi^2)(1-\eta) \\
N_6 &= \frac{1}{2}(1+\xi)(1-\eta^2) \\
N_7 &= \frac{1}{2}(1-\xi^2)(1+\eta) \\
N_8 &= \frac{1}{2}(1-\xi)(1-\eta^2)
\end{aligned}
\tag{3.8}
$$

3.8 Solid185

Das Solid185-Element ist ein dreidimensionales Quaderelement mit acht Knoten. Diese sind in den Ecken angeordnet (Abb. 3.4).

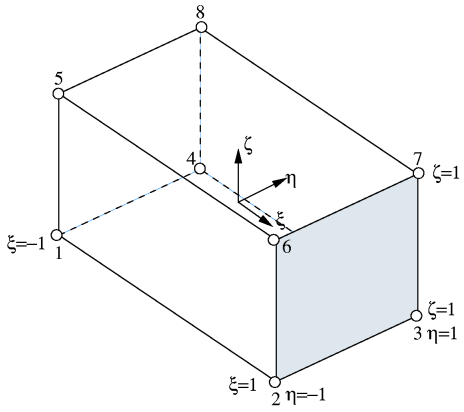

Abbildung 3.4: Solid185 Element

Die Formfunktionen sind [8]:

$$
\begin{aligned}
N_1 &= \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta) \\
N_2 &= \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta) \\
N_3 &= \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta) \\
N_4 &= \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta) \\
N_5 &= \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta) \\
N_6 &= \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta) \\
N_7 &= \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta) \\
N_8 &= \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)
\end{aligned}
\tag{3.9}
$$

3.9 Solid186

Das Solid186-Element ist ein dreidimensionales Quaderelement mit 20 Knoten. Diese sind in den Ecken und in den Kantenmitten angeordnet (Abb. 3.5).

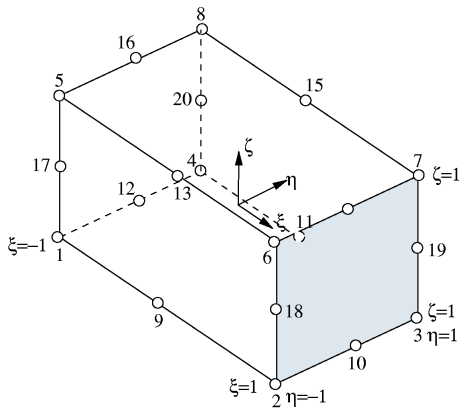

Abbildung 3.5: Solid186 Element

Die Formfunktionen sind [8]:

$$
\begin{aligned}
N_1 &= \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)(-\xi-\eta-\zeta-2) \qquad (3.10)\\
N_2 &= \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)(\xi-\eta-\zeta-2)\\
N_3 &= \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)(\xi+\eta-\zeta-2)\\
N_4 &= \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)(-\xi+\eta-\zeta-2)\\
N_5 &= \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)(-\xi-\eta+\zeta-2)\\
N_6 &= \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)(\xi-\eta+\zeta-2)\\
N_7 &= \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)(\xi+\eta+\zeta-2)\\
N_8 &= \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)(-\xi+\eta+\zeta-2)\\
N_9 &= \frac{1}{4}(1-\xi^2)(1-\eta)(1-\zeta)\\
N_{10} &= \frac{1}{4}(1-\eta^2)(1+\xi)(1-\zeta)\\
N_{11} &= \frac{1}{4}(1-\xi^2)(1+\eta)(1-\zeta)\\
N_{12} &= \frac{1}{4}(1-\eta^2)(1-\xi)(1-\zeta)\\
N_{13} &= \frac{1}{4}(1-\xi^2)(1-\eta)(1+\zeta)\\
N_{14} &= \frac{1}{4}(1-\eta^2)(1+\xi)(1+\zeta)\\
N_{15} &= \frac{1}{4}(1-\xi^2)(1+\eta)(1+\zeta)\\
N_{16} &= \frac{1}{4}(1-\eta^2)(1-\xi)(1+\zeta)\\
N_{17} &= \frac{1}{4}(1-\zeta^2)(1-\xi)(1-\eta)\\
N_{18} &= \frac{1}{4}(1-\zeta^2)(1+\xi)(1-\eta)\\
N_{19} &= \frac{1}{4}(1-\zeta^2)(1+\xi)(1+\eta)\\
N_{20} &= \frac{1}{4}(1-\zeta^2)(1-\xi)(1+\eta)
\end{aligned}
$$

3.10 Solid187

Das Solid187-Element ist ein dreidimensionales Tetraederelement mit zehn Knoten. Diese sind in den Ecken und in den Kantenmitten angeordnet (Abb. 3.6).

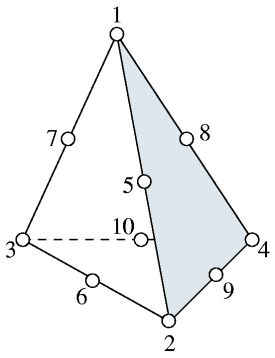

Abbildung 3.6: Solid187 Element

Die Formfunktionen sind [8]:

$$\begin{aligned} N_1 &= (2L_1 - 1)L_1 \\ N_2 &= (2L_2 - 1)L_2 \\ N_3 &= (2L_3 - 1)L_3 \\ N_4 &= (2L_4 - 1)L_4 \\ N_5 &= 4L_1L_2 \\ N_6 &= 4L_2L_3 \\ N_7 &= 4L_1L_3 \\ N_8 &= 4L_1L_4 \\ N_9 &= 4L_2L_4 \\ N_{10} &= 4L_3L_4 \end{aligned} \tag{3.11}$$

wobei L_1, L_2, L_3 und L_4 Tetraeder-Koordinaten sind [8].

3.11 Numerische Integration

Für die Elementtypen in den Abschnitten 3.5 - 3.10 muss die Elementmassenmatrix durch numerisches Aufintegrieren an bestimmten Integrationspunkten ermittelt werden. Für zweidimensionale Elementtypen gilt an jedem Integrationspunkt i:

$$\mathbf{N}_i = \begin{bmatrix} N_{1,i} & 0 & N_{2,i} & 0 & N_{3,i} & 0 & \cdots & N_{n,i} & 0 \\ 0 & N_{1,i} & 0 & N_{2,i} & 0 & N_{3,i} & \cdots & 0 & N_{n,i} \end{bmatrix} \tag{3.12}$$

wobei n die Anzahl Knoten des Elements ist. Für dreidimensionale Elemente gilt:

$$\mathbf{N}_i = \begin{bmatrix} N_{1,i} & 0 & 0 & N_{2,i} & 0 & 0 & \cdots & N_{n,i} & 0 & 0 \\ 0 & N_{1,i} & 0 & 0 & N_{2,i} & 0 & \cdots & 0 & N_{n,i} & 0 \\ 0 & 0 & N_{1,i} & 0 & 0 & N_{2,i} & \cdots & 0 & 0 & N_{n,i} \end{bmatrix} \tag{3.13}$$

Die Massenmatrix $\mathbf{m}$ eines Elements wird durch Aufsummierung der skalierten Matrizen $\mathbf{N}^T\mathbf{N}$ erhalten.

$$\mathbf{m} = \sum_{i=1}^{k} w_i \rho \det(\mathbf{J}_i) \mathbf{N}^T \mathbf{N} \tag{3.14}$$

wobei w_i die Gewichtung des Integrationspunkts, ρ die Dichte des Materials und k die Anzahl Integrationspunkte des Elements sind. $\mathbf{J}_i$ ist die Jacobi-Matrix am Integrationspunkt i. Sie wird auch für das Aufstellen der Steifigkeitsmatrix gebraucht und von dort übernommen.

3.12 Globale Massenmatrix

Das Aufstellen der globalen Massenmatrix GMM geschieht analog zum Aufstellen der globalen Steifigkeitsmatrix GSM. Der Code, der die GSM generiert, ist übernommen und entsprechend modifiziert worden. Anpassungen waren nötig in `PreProcessing.inl`, `Element.inl`, der Basisklassen der Elementtypen und der Klassen der Elementtypen selber.

Kapitel 4

Spezielles Eigenwertproblem

4.1 Idee

Um die Eigenfrequenzen einer Struktur zu berechnen, ist ein allgemeines Eigenwertproblem (Gleichung 4.1) zu lösen.

$$\mathbf{K}\mathbf{u}_i = \lambda_i \mathbf{M}\mathbf{u}_i \tag{4.1}$$

wobei λ_i die Eigenwerte und $\mathbf{u}_i$ die Eigenvektoren des Problems sind.

Einfacher zu lösen und in der Literatur häufiger beschrieben ist allerdings ein spezielles Eigenwertproblem (Gleichung 4.2).

$$\mathbf{C}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{4.2}$$

Wäre die Matrix $\mathbf{C}$ bekannt, könnte das Eigenwertproblem auf einfachere Weise gelöst werden. Als Solver könnte zum Beispiel die `Iterative Eigenproblem Template library` [11] verwendet werden.

Die Matrix $\mathbf{C}$ kann man durch Invertieren der Matrix $\mathbf{M}$ und Multiplizieren mit der Matrix $\mathbf{K}$ (Gleichung 4.3) erhalten.

$$\mathbf{C} = \mathbf{M}^{-1}\mathbf{K} \tag{4.3}$$

Dieser Weg hat zwei Schwachpunkte. Die Inversion der Matrix $\mathbf{M}$ ist mit sehr viel Rechenaufwand verbunden. Daneben ist die Matrix $\mathbf{C}$ im Allgemeinen nicht mehr symmetrisch.

4.2 Durchführung

Es gibt noch einen anderen Weg, das allgemeine Eigenwertproblem auf ein spezielles zurückzuführen [4]. Hier ist keine Inversion nötig und die erhaltene Matrix $\mathbf{C}$ ist symmetrisch, was die Eigenwertsuche vereinfacht.
Zuerst führt man eine Cholesky-Zerlegung der Massenmatrix durch.

$$\mathbf{M} = \mathbf{L}\mathbf{L}^T \tag{4.4}$$

Da $\mathbf{L}$ keine grössere Enveloppe als $\mathbf{M}$ hat, kann die Zerlegung an der Stelle von $\mathbf{M}$ passieren, um Speicherplatz zu sparen. Dann wird eine Hilfsmatrix $\mathbf{H}$ gebildet.

$$\mathbf{H} = \mathbf{K}\mathbf{L}^{-T} \tag{4.5}$$

Die Matrix **H** ist voll besetzt und nicht mehr symmetrisch. Es genügt jedoch, ihre untere Hälfte zu bestimmen. Dies geschieht kollonnenweise mittels Gleichung 4.6.

$$\mathbf{H}\mathbf{L}^T = \mathbf{K} \tag{4.6}$$

Jetzt kann aus Gleichung 4.7 die symmetrische Matrix **C** durch Vorwärtseinsetzen berechnet werden.

$$\mathbf{L}\mathbf{C} = \mathbf{H} \tag{4.7}$$

Die Matrix **C** ist zwar symmetrisch, aber vollbesetzt. Da die Speicherung einer vollbesetzten Matrix bei vielen Freiheitsgraden enorm viel Speicherplatz beansprucht, wurde dieser Lösungsweg als unrealistisch bewertet und nicht weiter verfolgt.

4.3 Beispiel

Das folgende Beispiel soll die Reduktion eines allgemeinen auf ein spezielles Eigenwertproblem illustrieren. Die Steifigkeits- und Massenmatrix stammen aus dem Balkenproblem in Kapitel 2.

$$\mathbf{K} = 10^6 \cdot \begin{bmatrix}
260 & 0 & 0 & -130 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.41 & 0 & 0 & -0.2 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.14 & 0 & -0.1 & 0.034 & 0 & 0 & 0 & 0 & 0 & 0 \\
-130 & 0 & 0 & 260 & 0 & 0 & -130 & 0 & 0 & 0 & 0 & 0 \\
0 & -0.2 & -0.1 & 0 & 0.41 & 0 & 0 & -0.2 & 0.1 & 0 & 0 & 0 \\
0 & 0.1 & 0.034 & 0 & 0 & 0.14 & 0 & -0.1 & 0.034 & 0 & 0 & 0 \\
0 & 0 & 0 & -130 & 0 & 0 & 260 & 0 & 0 & -130 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.2 & -0.1 & 0 & 0.41 & 0 & 0 & -0.2 & 0.1 \\
0 & 0 & 0 & 0 & 0.1 & 0.034 & 0 & 0 & 0.14 & 0 & -0.1 & 0.034 \\
0 & 0 & 0 & 0 & 0 & 0 & -130 & 0 & 0 & 130 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.2 & -0.1 & 0 & 0.2 & -0.1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.034 & 0 & -0.1 & 0.068
\end{bmatrix} \tag{4.8}$$

$$\mathbf{M} = \begin{bmatrix}
2.9 & 0 & 0 & 0.72 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3.2 & 0 & 0 & 0.56 & -0.13 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.082 & 0 & 0.13 & -0.031 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.72 & 0 & 0 & 2.9 & 0 & 0 & 0.72 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.56 & 0.13 & 0 & 3.2 & 0 & 0 & 0.56 & -0.13 & 0 & 0 & 0 \\
0 & -0.13 & -0.031 & 0 & 0 & 0.082 & 0 & 0.13 & -0.031 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.72 & 0 & 0 & 2.9 & 0 & 0 & 0.72 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.56 & 0.13 & 0 & 3.2 & 0 & 0 & 0.56 & -0.13 \\
0 & 0 & 0 & 0 & -0.13 & -0.031 & 0 & 0 & 0.082 & 0 & 0.13 & -0.031 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.72 & 0 & 0 & 1.4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.56 & 0.13 & 0 & 1.6 & -0.23 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.13 & -0.031 & 0 & -0.23 & 0.041
\end{bmatrix} \tag{4.9}$$

$$\mathbf{L} = \begin{bmatrix}
1.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.42 & 0 & 0 & 1.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.31 & 0.47 & 0 & 1.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -0.075 & -0.11 & 0 & 0.043 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.44 & 0 & 0 & 1.6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0.48 & 0 & 1.7 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.079 & -0.11 & 0 & 0.046 & 0.25 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.44 & 0 & 0 & 1.1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0.48 & 0 & 1.1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.079 & -0.11 & 0 & -0.13 & 0.075
\end{bmatrix} \tag{4.10}$$

$$\mathbf{H} = 10^6 \cdot \begin{bmatrix}
150 & 0 & 0 & -120 & 0 & 0 & 31 & 0 & 0 & -12 & 0 & 0 \\
0 & 0.23 & 0 & 0 & -0.16 & 0.5 & 0 & -0.11 & 0.19 & 0 & -0.048 & 0.076 \\
0 & 0 & 0.48 & 0 & -0.19 & 0.37 & 0 & -0.068 & 0.12 & 0 & -0.029 & 0.046 \\
-75 & 0 & 0 & 180 & 0 & 0 & -120 & 0 & 0 & 49 & 0 & 0 \\
0 & -0.11 & -0.36 & 0 & 0.36 & -0.25 & 0 & -0.12 & 0.44 & 0 & -0.15 & 0.25 \\
0 & 0.057 & 0.12 & 0 & -0.043 & 0.62 & 0 & -0.23 & 0.44 & 0 & -0.12 & 0.19 \\
0 & 0 & 0 & -78 & 0 & 0 & 180 & 0 & 0 & -180 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.12 & -0.39 & 0 & 0.37 & -0.28 & 0 & -0.17 & 1 \\
0 & 0 & 0 & 0 & 0.06 & 0.13 & 0 & -0.047 & 0.63 & 0 & -0.34 & 0.72 \\
0 & 0 & 0 & 0 & 0 & 0 & -78 & 0 & 0 & 150 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.12 & -0.39 & 0 & 0.38 & -1.4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.06 & 0.13 & 0 & -0.16 & 0.87
\end{bmatrix} \tag{4.11}$$

$$\mathbf{C} = 10^6 \cdot \begin{bmatrix} 89 & 0 & 0 & -69 & 0 & 0 & 18 & 0 & 0 & -7.2 & 0 & 0 \\ 0 & 0.13 & 0 & 0 & -0.09 & 0.28 & 0 & -0.061 & 0.11 & 0 & -0.027 & 0.042 \\ 0 & 0 & 1.7 & 0 & -0.66 & 1.3 & 0 & -0.24 & 0.4 & 0 & -0.1 & 0.16 \\ -69 & 0 & 0 & 120 & 0 & 0 & -81 & 0 & 0 & 32 & 0 & 0 \\ 0 & -0.09 & -0.66 & 0 & 0.41 & -0.55 & 0 & 0.005 & 0.13 & 0 & -0.055 & 0.094 \\ 0 & 0.28 & 1.3 & 0 & -0.55 & 3.2 & 0 & -1 & 1.9 & 0 & -0.51 & 0.82 \\ 18 & 0 & 0 & -81 & 0 & 0 & 130 & 0 & 0 & -120 & 0 & 0 \\ 0 & -0.061 & -0.24 & 0 & 0.005 & -1 & 0 & 0.5 & -0.72 & 0 & 0.052 & 0.37 \\ 0 & 0.11 & 0.4 & 0 & 0.13 & 1.9 & 0 & -0.72 & 3.5 & 0 & -1.6 & 3.2 \\ -7.2 & 0 & 0 & 32 & 0 & 0 & -120 & 0 & 0 & 180 & 0 & 0 \\ 0 & -0.027 & -0.1 & 0 & -0.055 & -0.51 & 0 & 0.052 & -1.6 & 0 & 1 & -2.7 \\ 0 & 0.042 & 0.16 & 0 & 0.094 & 0.82 & 0 & 0.37 & 3.2 & 0 & -2.7 & 12 \end{bmatrix} \tag{4.12}$$

Hier und in Rechnungen mit Matrizen höherer Ordnung fällt auf, dass die Matrix $\mathbf{C}$ zwar vollbesetzt ist, dass aber Einträge stark kleiner werden, je weiter weg sie von der Diagonalen sind. Es ist wohl gut möglich, in genügendem Abstand von der Diagonalen ein Band zu legen und die Werte ausserhalb des Bandes Null zu setzen, ohne dass die Eigenwerte von $\mathbf{C}$ signifikant verfälscht werden.

Da dieser Weg in der Literatur aber kaum beschrieben ist, wird dies nicht die optimale Lösung für das gegebene Problem sein.

Kapitel 5

Vektoriteration

5.1 Idee

Die Vektoriteration eignet sich, um die kleinsten Eigenwerte und die dazugehörigen Eigenvektoren einer allgemeinen Eigenwertaufgabe zu berechnen. Bei dieser Methode wird, im Gegensatz zur Reduktion auf ein spezielles Eigenwertproblem, die Hüllenform der Steifigkeits- und Massenmatrix optimal ausgenützt.

Die Vektoriteration startet mit einem beliebigen Startvektor, dessen Dimension gleich der Anzahl Freiheitsgrade des Problems ist. Im Verlauf der Iteration nähert er sich immer mehr einem Eigenvektor an. Bei genügender Genauigkeit wird der Algorithmus abgebrochen und mit Hilfe des Eigenvektors der dazugehörige Eigenwert berechnet.

Für die mathematische Herleitung des Algorithmus sei auf Schwarz [4] verwiesen.

5.2 Umsetzung

Für die Umsetzung in C++ wird die *simultane Vektoriteration* verwendet. Dabei werden statt nur einem Vektor $\mathbf{y}$ mehrere Vektoren gleichzeitig iteriert. Die Anzahl der Vektoren entspricht der Anzahl der gesuchten Eigenwerte. Die Vektoren werden in die Matrix $\mathbf{Y}$ geschrieben.

Abbildung 5.1 zeigt das Flussdiagramm der simultanen Vektoriteration. Die Matrizen $\mathbf{Y}$, $\mathbf{H}_1$, $\mathbf{H}_2$, $\mathbf{H}_3$, und $\mathbf{Z}$ haben die Dimensionen $m \times n$, wobei m die Ordnung der Steifigkeits- bzw. Massenmatrix und n die Anzahl der gesuchten Eigenwerte ist. Die Matrizen $\mathbf{R}$, $\mathbf{C}$, und $\mathbf{G}$ haben die Dimensionen $n \times n$.

In einem ersten Schritt wird die Cholesky-Zerlegung (Gl. 5.1) auf die Steifigkeits- und Massenmatrix angewendet. Daraus ergeben sich die Matrizen $\mathbf{L}_K$ und $\mathbf{L}_M$, welche in der gleichen Hülle wie $\mathbf{K}$ und $\mathbf{M}$ gespeichert werden können. Die Startmatrix $\mathbf{Y}_0$ kann beliebig gewählt werden.

$$\mathbf{K} = \mathbf{L}_K \mathbf{L}_K^T \qquad \mathbf{M} = \mathbf{L}_M \mathbf{L}_M^T \tag{5.1}$$

Im Iterationsprozess wird zuerst die Matrix $\mathbf{H}_1$ berechnet durch Multiplikation der Matrizen $\mathbf{L}_M$ und $\mathbf{Y}$. Damit wird nun die Matrix $\mathbf{H}_2$ mittels Vorwärtseinsetzen im Glei-

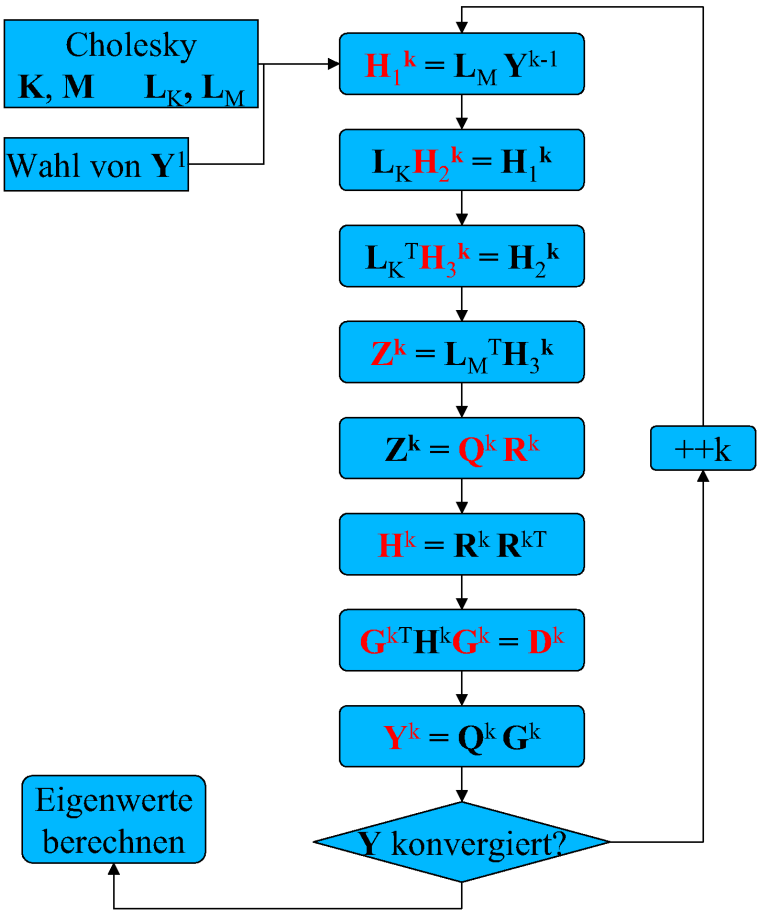

Abbildung 5.1: Simultane Vektoriteration

chungssystem 5.2 ermittelt.

$$\mathbf{L}_K \mathbf{H}_2 = \mathbf{H}_1 \tag{5.2}$$

Im Gleichungssystem 5.3 wird dann die Matrix $\mathbf{H}_3$ mittels Rückwärtseinsetzen berechnet.

$$\mathbf{L}_K^T \mathbf{H}_3 = \mathbf{H}_2 \tag{5.3}$$

Nach einer weiteren Multiplikation von $\mathbf{L}_M^T$ mit $\mathbf{H}_3$ gelangt man zur Matrix $\mathbf{Z}$. Diese wird einer QR-Zerlegung unterzogen, wobei nur die Matrix $\mathbf{R}$ berechnet und für den weiteren Verlauf benötigt wird. Aus der Multiplikation der Matrix $\mathbf{R}$ mit ihrer Transponierten ergibt sich die Matrix $\mathbf{H}$.

In einem weiteren Schritt wird die Matrix $\mathbf{H}$ diagonalisiert, daraus ergibt sich die orthogonale Matrix $\mathbf{G}$. Die neue Matrix $\mathbf{Y}$ erhält man schliesslich durch Multiplikation von $\mathbf{Q}$ mit $\mathbf{G}$.

Falls $\mathbf{Y}$ schon genügend gegen die Eigenvektoren konvergiert ist, wird die Iteration abgebrochen und die Eigenwerte berechnet. Ansonsten wird der ganze Rechenablauf nochmals mit dem neuen Vektor $\mathbf{Y}$ wiederholt.

Ein *Matlab*-Programm für die simultane Vektoriteration findet sich in Anhang A.

5.3 Orthogonalisierung

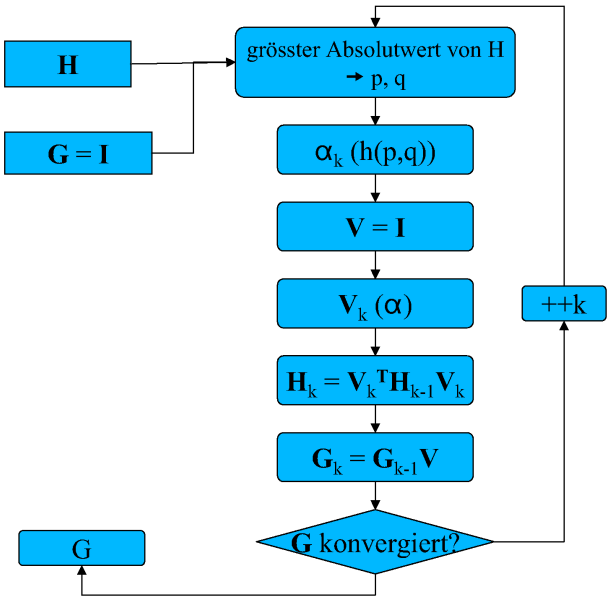

Abbildung 5.2: Orthogonalisierung

Die Orthogonalisierung der Matrix **H** geschieht mit dem Jacobi-Verfahren [12]. Dafür ist eine eigene Iteration nötig. **G** und **V** haben die Dimensionen von **H**. Die Matrix **G** wird gleich der Einheitsmatrix gesetzt. Im Iterationsschritt wird zuerst der absolut grösste Eintrag $h_{p,q}$ der Matrix **H** gesucht. Dann wird der Wert α gemäss Gleichung 5.4 berechnet

$$\alpha = \frac{1}{2} \arctan\left(\frac{2h_{p,q}}{h_{p,p} - h_{q,q}}\right) \tag{5.4}$$

wobei $h_{p,q}$ der Eintrag in der $p$ten Zeile und $q$ten Spalte von **H** ist. Dann wird **V** gleich der Einheitsmatrix gesetzt. Die folgenden Einträge werden in **V** vorgenommen.

$$\begin{aligned} v_{p,p} = v_{q,q} &= \cos(\alpha) \\ v_{p,q} &= -\sin(\alpha) \\ v_{q,p} &= \sin(\alpha) \end{aligned} \tag{5.5}$$

Aus $\mathbf{V}^T\mathbf{HV}$ wird ein neues **H** berechnet, welches sich immer mehr einer Diagonalmatrix annähert. Schliesslich wird mit **GV** ein neues **G** berechnet. Wenn dieses sich nicht mehr stark ändert oder eine bestimmte Anzahl Iterationsschritte erreicht ist, approximiert **G** die orthogonalisierte Matrix **H**.

Ein *Matlab*-Programm zur Orthogonalisierung findet sich ebenfalls in Anhang A.

5.4 Weitere Detaillierungen

Für die genaue Beschreibung der Cholesky- bzw. QR-Zerlegung sei auf Schwarz [4] verwiesen.

Die in der simultanen Vektoriteration nach erfolgter Konvergenz erhaltenen Vektoren $\mathbf{y}_i$ müssen noch durch Rückwärtseinsetzen mit $\mathbf{L}_M$ auf die eigentlich gesuchten Eigenvektoren $\mathbf{u}_i$ zurücktransformiert werden. Da Rückwärtseinsetzen schon im dritten Schritt der Iteration vorkommt, kann dieselbe Funktion verwendet werden.

Die Eigenwerte ω_i^2 ergeben sich als Rayleighsche Quotienten der Eigenvektoren.

$$\omega_i^2 = \frac{\mathbf{u}_i^T \mathbf{K} \mathbf{u}_i}{\mathbf{u}_i^T \mathbf{M} \mathbf{u}_i} = \frac{(\mathbf{L}_K^T \mathbf{u}_i)^T (\mathbf{L}_K^T \mathbf{u}_i)}{\mathbf{y}_i^T \mathbf{y}_i} \tag{5.6}$$

Für den praktischen Einsatz eignet sich die Vektoriteration in der C++-Implementierung nicht. Sie benötigt schon für kleine Matrizen über 100 Mal mehr Zeit um die Lösung zu berechnen, als das Verfahren von Lanczos. Die Implementierung ist nicht optimal, ihre Effizienz könnte sicher noch beträchtlich gesteigert werden.

Kapitel 6

Verfahren von Lanczos

6.1 Idee

Das eigentliche Verfahren von Lanczos eignet sich um die grössten Eigenwerte eines speziellen Eigenwertproblems zu berechnen. Um ein bestimmtes Spektrum von Eigenwerten zu erhalten, benützt man ein invertiertes, spektralverschobenes Eigenwertproblem. Auf das allgemeine Eigenwertproblem wird zuerst eine Spektralverschiebung um μ angewandt.

$$(\mathbf{K} - \mu\mathbf{M})\mathbf{u}_i = (\omega_i^2 - \mu)\mathbf{M}\mathbf{u}_i \tag{6.1}$$

Die Reduktion auf ein spezielles Eigenwertproblem geschieht nicht explizit wie in Kapitel 4, sondern implizit im Verfahren von Lanczos. Formal sieht das spezielle spektralverschobene Eigenwertproblem dann folgendermassen aus

$$\mathbf{C}\mathbf{y}_i = \mathbf{L}_M^{-1}(\mathbf{K} - \mu\mathbf{M})\mathbf{L}_M^{-T}\mathbf{y}_i = (\omega_i^2 - \mu)\mathbf{y}_i \tag{6.2}$$

wobei $\mathbf{L}_M$ die Cholesky-Zerlegung von $\mathbf{M}$ und $\mathbf{y}_i = \mathbf{L}_M^T\mathbf{u}_i$ sei.

Ursprünglich sind ja diejenigen Eigenwerte gesucht, die am nächsten zum Wert μ liegen. Diese sind die betragsmässig kleinsten Eigenwerte von $\mathbf{C}$. Da aber die kleinsten Eigenwerte im vergleich zum gesamten Spektrum von Eigenwerten schlecht voneinander getrennt sind, würde der Lanczos-Algorithmus nur langsam gegen die gesuchten Werte konvergieren. Deshalb wird mit der Inversen von $\mathbf{C}$ gearbeitet, deren Eigenwerte $\varrho_i = \frac{1}{\omega_i^2 - \mu}$ sind. Die betragskleinen Eigenwerte werden so in betragsgrosse überführt, welche in der Regel gut getrennt sind (Abb. 6.1). Dies hat eine schnelle Konvergenz des Algorithmus zur Folge.

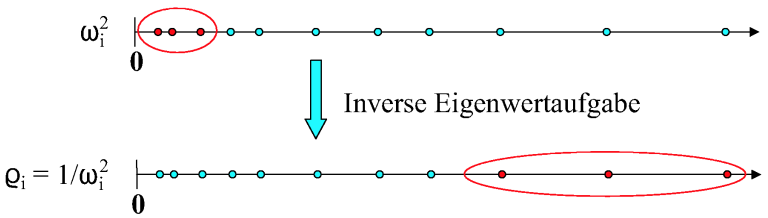

Abbildung 6.1: Eigenwerte beim inversen Eigenwertproblem ($\mu = 0$)

Die Aufgabe dieser Arbeit ist es, die kleinsten Eigenfrequenzen einer Struktur zu bestimmen. Aus diesem Grund beträgt bei der Implementierung der Wert μ immer Null.

Es wird also mit einem invertierten, aber nicht spektralverschobenen System gerechnet. Dies hat insbesondere beim Lösen des Gleichungssystems $\mathbf{Fu} = \mathbf{h}$ im Lanczos-Algorithmus den Vorteil, dass die Matrix $\mathbf{F}$ immer positiv definit, und so das Lösen nach $\mathbf{u}$ deutlich einfacher ist. Um die Allgemeinheit jedoch nicht einzuschränken, wird im Bericht aber die Methode mit einer Spektralverschiebung weiterverfolgt.

Für die mathematische Herleitung des Algorithmus, insbesondere der impliziten Reduktion auf ein spezielles Eigenwertproblem, sei auf Schwarz [4] verwiesen.

6.2 Umsetzung

Abbildung 6.2 zeigt das Flussdiagramm des Verfahren von Lanczos. Die Vektoren $\mathbf{h}$, $\mathbf{q}$, $\mathbf{r}$ und $\mathbf{u}$ haben dieselbe Ordnung wie die Steifigkeits- bzw. Massenmatrix. $\mathbf{F}$ kann in dieselbe Hülle wie $\mathbf{K}$ oder $\mathbf{M}$ geschrieben werden. α und β sind Skalare.

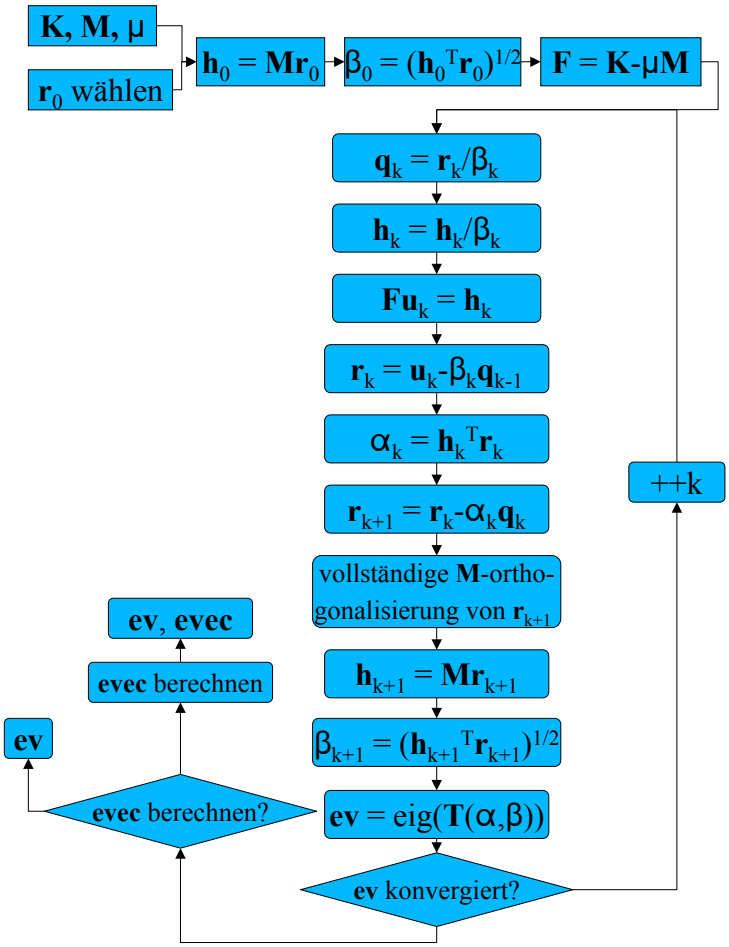

Abbildung 6.2: Das Verfahren von Lanczos

Zuerst wird der Startvektor $\mathbf{r}_0$ beliebig gewählt. Mittels Multiplikation mit der Massenmatrix wird damit $\mathbf{h}_0$ berechnet. Das Skalarprodukt von $\mathbf{r}_0$ und $\mathbf{h}_0$ ergibt β_0^2. Die Matrix $\mathbf{F}$ kann an der Stelle von $\mathbf{K}$ aufgebaut werden, da die Steifigkeitsmatrix im und nach dem Lanczos-Algorithmus nicht mehr benötigt wird. Ohne Spektraltransformation sind $\mathbf{F}$ und $\mathbf{K}$ ohnehin identisch.

Um das Gleichungssystem $\mathbf{Fu} = \mathbf{h}$ im dritten Schritt der Iteration möglichst effizient Lösen zu können, ist eine Zerlegung der Matrix $\mathbf{F}$ notwendig. Diese geschieht am besten gleich vor der ersten Iteration. Wie in Abschnitt 6.1 bereits erwähnt, ist $\mathbf{F}$ ohne Spektralverschiebung immer positiv definit. Die Matrix $\mathbf{F}$ kann direkt zerlegt und in dieselbe Hülle geschrieben werden, weil sie nur zum Lösen des Gleichungssystems benötigt wird.

Mit `skyline_solve()` existiert bereits eine effiziente Funktion in *FELyX*, um die Zerlegung und die anschliessende Rücksubstition zur Lösung eines Gleichungssystems vorzunehmen. Da die rechenaufwändige Zerlegung nur einmal, die Rücksubstition aber in jedem Iterationsschritt gebraucht wird, wird der Algorithmus zur Rücksubstition in die neue Funktion `backsubstitution()` geschrieben.

Bei einer Spektralverschiebung ist $\mathbf{F}$ in der Regel nicht positiv definit. Bei der Zerlegung müsste anders vorgegangen werden. $\mathbf{F}$ müsste in einer erweiterten Hülle von $\mathbf{K}$ aufgebaut werden, die pro Zeile zwei Einträge mehr aufnehmen könnte [4].

Beim eigentlichen Lanczos-Schritt werden zuerst die Vektoren $\mathbf{r}$ bzw. $\mathbf{h}$ mit $\frac{1}{\beta}$ skaliert und in den Vektor $\mathbf{q}$ bzw. wieder in $\mathbf{h}$ geschrieben. Dann wird das Gleichungssystem $\mathbf{Fu} = \mathbf{h}$, wie bereits erwähnt, mit der Funktion `backsubstitution()` gelöst. Im ersten Lanczos-Schritt wird $\mathbf{r}$ gleich $\mathbf{u}$ gesetzt und in jedem folgenden dann gleich $\mathbf{u}$ minus dem mit β skalierten Vektor $\mathbf{q}$ aus dem vorhergehenden Lanczos-Schritt. Aus dem Skalarprodukt der Vektoren $\mathbf{h}$ und $\mathbf{r}$ ergibt sich α und damit ein neuer Vektor $\mathbf{r}$, indem der mit α skalierte Vektor $\mathbf{q}$ von $\mathbf{r}$ subtrahiert wird.

$\mathbf{q}_k$ sind die Basisvektoren des Krylov-Unterraums. Mit zunehmender Schrittzahl k geht ihre Orthogonalität verloren, so dass sie schliesslich linear abhängig werden und man nicht mehr von einer Basis sprechen kann. Dies führt dazu, dass mehrfache Kopien von Eigenpaaren berechnet werden. Der Lanczos-Algorithmus galt deshalb lange Zeit als instabil und nicht brauchbar. Die Orthogonalität der Basisvektoren $\mathbf{q}_k$ kann aber durch eine vollständige Orthogonalisierung bezüglich aller vorangehenden Vektoren $\mathbf{q}$ numerisch erzwungen werden. Die Vektoren $\mathbf{q}_k$ müssen also in jedem Lanczos-Schritt gespeichert werden. Sie werden in die $k$te Spalte der $(m \times n)$-Matrix $\mathbf{Q}$ geschrieben, wobei m der Ordnung von $\mathbf{q}$ entspricht und n die zu Beginn festzulegende maximale Anzahl der Lanczos-Schritte ist. Die vollständige Orthogonalisierung bezüglich $\mathbf{M}$ ist in Gleichung 6.3 dargestellt.

$$\mathbf{r} = \mathbf{r} - \mathbf{Q}(\mathbf{Q}^T(\mathbf{Mr})) \tag{6.3}$$

Nach erfolgter Orthogonalisierung wird $\mathbf{M}$ mit $\mathbf{r}$ multipliziert und so ein neuer Vektor $\mathbf{h}$ erzeugt. β^2 ergibt sich dann als Skalarprodukt von $\mathbf{h}$ und $\mathbf{r}$.

6.3 Bisektionsmethode

Die skalaren Werte α_k und β_k werden in eine mit jedem Lanczos-Schritt wachsende Matrix $\mathbf{T}$ geschrieben.

$$\mathbf{T} = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & \\ & & \beta_3 & \alpha_4 & \beta_4 & \\ & & & \ddots & \ddots & \ddots \\ & & & & & \end{bmatrix} \tag{6.4}$$

Die gesuchten grössten Eigenwerte der Matrix $\mathbf{C}$ (Gl. 6.2) entsprechen den Eigenwerten der Matrix $\mathbf{T}$. Diese ist mit der Ordnung k relativ klein und ausserdem tridiagonal. Dies erlaubt ein effizientes Berechnen ihrer Eigenwerte mit der Bisektionsmethode.

Zuerst wird eine untere und obere Schranke a und b für das Spektrum der Eigenwerte festgelegt (Abb. 6.3). Für die obere Schranke eignet sich eine beliebige Matrixnorm, in der Implementierung die Zeilenmaximumnorm. Die untere Schranke ist im allgemeinen Fall $a = -b$. Ohne Spektralverschiebung kann Null als untere Schranke angenommen werden, da dann sicher keine Eigenwerte von $\mathbf{C}$ negativ sind.

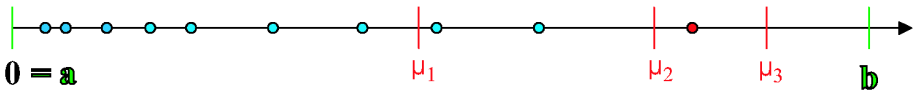

Abbildung 6.3: Bisektionsmethode

In jedem Bisektionsschritt wird die Folge $[q_1, q_2, q_3, ..., q_k]$ rekursiv definiert nach der Rekursionsvorschrift 6.5.

$$\begin{aligned} q_1 &= \mu - \alpha_1 \\ q_j &= (\mu - \alpha_j) - \frac{\beta_{j-1}^2}{q_{j-1}} \end{aligned} \tag{6.5}$$

mit $j = 2, 3, ..., k$. Falls ein q Null werden sollte, wird es gleich minus der kleinsten Maschinenzahl (Maschinen-Epsilon) gesetzt, damit keine Division durch Null auftritt [4]. Die Anzahl Elemente der Folge mit negativem Vorzeichen ist gleich der Anzahl Eigenwerte, die grösser als der Wert μ sind.

Damit lässt sich nun jeder Eigenwert beliebig genau lokalisieren. Zur Illustration des Vorgehens dient das Beispiel in Abbildung 6.3. Es sei der grösste Eigenwert gesucht. Zu Beginn wird $\mu_1 = \frac{a+b}{2}$ gesetzt. Die Folge 6.5 wird berechnet, woraus sich ergibt, dass drei Eigenwerte grösser als μ_1 sind. μ_1 wird zur neuen unteren Schranke a und $\mu_2 = \frac{a+b}{2}$ wird berechnet. Die Folge sagt aus, dass noch ein Eigenwert grösser als μ_2 ist. μ_2 wird neue untere Schranke a. Oberhalb von μ_3 liegt kein Eigenwert mehr, so dass die obere Schranke b auf μ_3 gesetzt wird. Und so weiter.

Ungleichung 6.6 beschreibt die Anzahl nötiger Bisektionsschritte t, um eine absolute Genauigkeit ϵ zu erreichen.

$$t \geqslant \log_2 \left(\frac{b-a}{\epsilon} \right) - 1 \tag{6.6}$$

Ein *Matlab*-Programm zur Bisektionsmethode findet sich in Anhang B.

6.4 Konvergenz und Eigenpaare

Mit jedem Lanczos-Schritt, das heisst mit grösser werdender Matrix **T** approximieren die ermittelten Eigenwerte von **T** die grössten Eigenwerte von **C** besser (Abb. 6.4). In der Implementation gilt als Konvergenzkriterium die relative Änderung eines Eigenvektors von einem zum nächsten Lanczos-Schritt. Fällt diese relative Änderung bei allen gesuchten Eigenvektoren unter einen bestimmten Wert *tol*, wird das Verfahren von Lanczos abgebrochen. Es macht natürlich erst Sinn die Eigenwerte von **T** zu berechnen, wenn die Ordnung der Matrix grösser oder gleich der Anzahl gesuchter Eigenwerte ist.

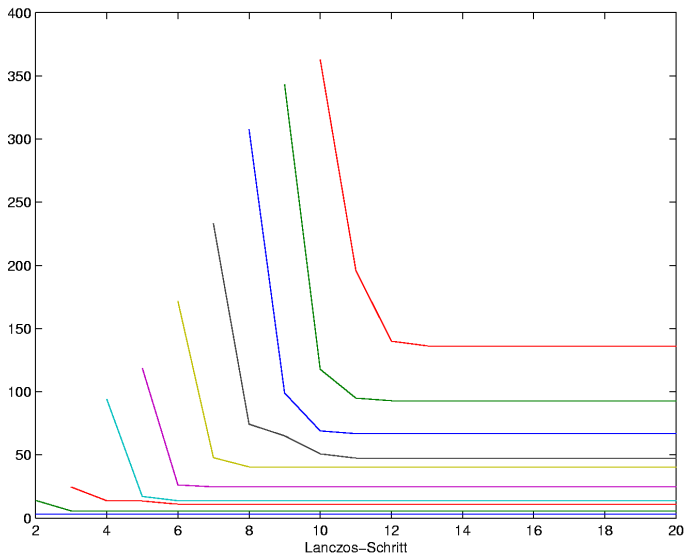

Abbildung 6.4: Konvergenz der Eigenwerte

Da ja ein inverses und eventuell spektralverschobenes Eigenwertproblem behandelt worden ist (Abschn. 6.1), müssen die konvergierten Eigenwerte ϱ_i von **C** auf die ursprünglich gesuchten Eigenwerte ω_i^2 mittels Gleichung 6.7 zurücktransformiert werden.

$$\omega_i^2 = \frac{1}{\varrho_i} + \mu \tag{6.7}$$

Die erhaltenen Eigenwerte ω_i sind schliesslich noch durch 2π zu dividieren, um die gesuchten Eigenfrequenzen der Struktur in Hertz zu erhalten.

$$f_i = \frac{\omega_i}{2\pi} \tag{6.8}$$

Für die Berechnung der Eigenvektoren, die erst am Schluss erfolgt und kaum Zeit beansprucht, werden nur die Matrizen **Q** und **T** benötigt. Der Algorithmus dazu stammt aus einem Fortran-Programm [5] und ist in C++ übersetzt worden.

6.5 Probleme, Block-Lanczos

Das Problem von mehrfachen Kopien einzelner Eigenpaar ist durch die vollständige Orthogonalisierung eliminiert worden. Dabei wird die Matrix **Q** mit sämtlichen je berech-

neten Vektoren **q** benützt. Um die Effizienz noch etwas zu steigern, könnte die selektive Orthogonalisierung angewendet werden, welche aber schwieriger zu implementieren ist.

Es kann im Verfahren von Lanczos gelegentlich vorkommen, dass Eigenwerte ausgelassen werden. Dies ist bei Testberechnungen nur einmal geschehen. Bei `felyx-solid186-200.ansys` wird der 18. Eigenwert ausgelassen (Tab. 6.1). Die Werte LANB sind mit der Block-Lanczos-Methode von *Ansys6.0* berechnet worden. Die Vollständigkeit des Eigenwertspektrums müsste nachträglich mittels erneuter Rechnung und geeignet gewälter Spektralverschiebung um μ überprüft werden [4].

Ein auffälligeres Problem ist, dass der Lanczos-Algorithmus grundsätzlich nicht fähig ist, mehrfache Eigenwerte zu bestimmen, weil die tridiagonale Matrix **T** nur paarweise verschiedene Eigenwerte haben kann. Bei den Testberechnungen ist dies jedoch nicht oft aufgetreten. Bei der oben erwähnten Struktur werden zwei doppelte Eigenwerte nur einfach berechnet.

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	20.835	20.835	11	1120.8	1120.8
2	20.835	20.835	12	1120.8	1120.8
3	129.61	129.61	13	1382.3	1382.3
4	129.61	129.61	14	1635.6	1635.6
5	358.77	358.77	15	1635.6	
6	358.77	358.77	16	2225.9	2225.9
7	460.77	460.77	17	2225.9	
8	691.75	691.75	18	2303.8	
9	691.75	691.75	19	2303.8	
10	806.22	806.22			

Tabelle 6.1: Tiefste 19 Eigenwerte von `felyx-solid186-200.ansys` mit ρ = 2.7e-6

Der Grund, warum die meisten mehrfachen Eigenwerte auch als mehrfache berechnet werden, sind wohl numerische Ungenauheiten. Bei der Berechnung eines doppelten Eigenwerts werden wohl kaum zwei genau gleiche Eigenwerte resultieren, sondern sie werden sich ein wenig unterscheiden. Sei es aus Gründen von kleinen Fehlern, die schon beim Meshen entstehen, oder aus Rundungsfehlern irgendwo im ganzen Rechengang. Hier hat das Rechnen mit approximierten Werten für einmal einen Vorteil.

Um mehrfache Eigenwerte sicher berechnen zu können, ist das Block-Lanczos-Verfahren entwickelt worden [3]. Die Matrix **T** ist dabei nur blockweise tridiagonal. Die Eigenwerte können nicht mehr mit der Bisektionsmethode bestimmt werden, es muss zum Beispiel die Methode der Vektoriteration (Kap. 5) angewendet werden.

Kapitel 7

Testresultate

Das Aufstellen der Massenmatrizen der verschiedenen Elementtypen sowie das programmierte Verfahren von Lanczos bedürfen einer Verifikation. Dafür werden die tiefsten zehn Eigenfrequenzen der Strukturen in den Testfiles von *FELyX* berechnet. Es muss jeweils noch die Dichte ρ des Materials bestimmt werden, die für die statische Berechnung nicht nötig ist.

In den folgenden Abschnitten ist jeweils angegeben, wie lange die Berechnung mit den Eigensolvern *Subspace* (SUBSP) und *Block-Lanczos* (LANB) von *Ansys6.0* sowie mit *FELyX* auf einem *SunBlade100, 450 MHz* dauert. Die Zeit bezieht sich nur auf das Lösen des Eigenwertproblems nach Eigenwerten und Eigenvektoren.

Die Zeiten t für SUBSP und LANB werden folgendermassen berechnet.

$$t = (c_2 - c_1)\frac{\Delta w}{\Delta c} \tag{7.1}$$

wobei c_1 bzw. c_2 die CPU-Zeiten vor bzw. nach dem Lösen des Eigenwertproblems bedeuten. Δc ist die CPU-Zeit und Δw die tatsächlich verstrichene Zeit für die gesamte Berechnung.

7.1 Link8

Spezifikationen für `felyx-link8-21.ansys`:

Freiheitsgrade	18
ρ	2.7e-6
SUBSP [s]	-
LANB [s]	-
FELyX [s]	0.03

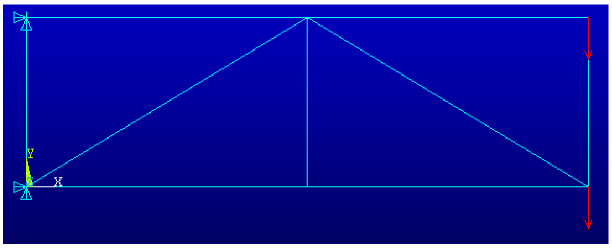

Abbildung 7.1: `felyx-link8-21.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	41.914	41.9143	6	194.14	194.144
2	56.474	56.4743	7	270.32	270.317
3	78.768	78.7680	8	431.48	431.475
4	134.39	134.387	9	437.00	436.995
5	166.86	166.862	10	492.41	492.408

Tabelle 7.1: Tiefste zehn Eigenfrequenzen für `felyx-link8-21.ansys`

7.2 Beam3

Spezifikationen für `felyx-beam3-800.ansys`:

Freiheitsgrade	2387
ρ	2.7e-6
SUBSP [s]	3.1
LANB [s]	2.2
FELyX [s]	1.3

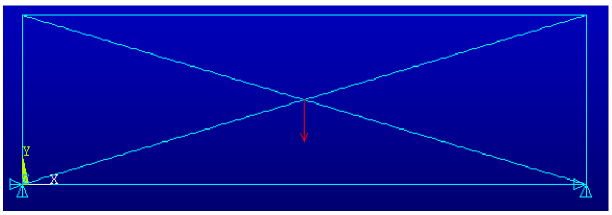

Abbildung 7.2: `felyx-beam3-800.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	46.5030	46.5030	6	151.928	151.928
2	50.4121	50.4121	7	174.278	174.278
3	120.891	120.891	8	188.651	188.651
4	124.722	124.722	9	245.948	245.948
5	149.600	149.600	10	272.512	272.512

Tabelle 7.2: Tiefste zehn Eigenfrequenzen für `felyx-beam3-800.ansys`

7.3 Beam4

Spezifikationen für `felyx-beam4-50.ansys`:

Freiheitsgrade	3558
ρ	0.0027
SUBSP [s]	10.1
LANB [s]	5.1
FELyX [s]	2.4

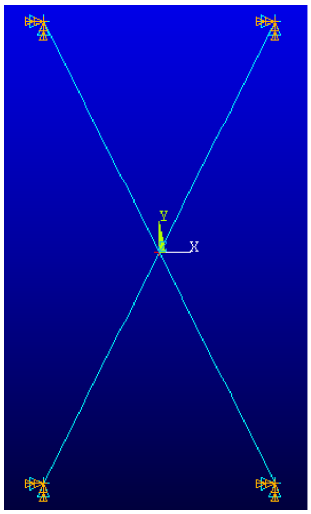

Abbildung 7.3: `felyx-beam4-50.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	0.207801	0.207801	6	0.370772	0.370772
2	0.225336	0.225336	7	0.383465	0.383465
3	0.243170	0.243170	8	0.448966	0.448966
4	0.247151	0.247151	9	0.460388	0.460388
5	0.350692	0.350692	10	0.475614	0.475614

Tabelle 7.3: Tiefste zehn Eigenfrequenzen für `felyx-beam4-50.ansys`

7.4 Plane2

Spezifikationen für `felyx-plane2-6654.ansys`:

Freiheitsgrade	27196
ρ	0.0027
SUBSP [s]	82.9
LANB [s]	79.6
FELyX [s]	56.2

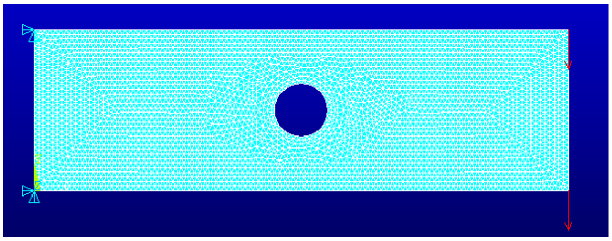

Abbildung 7.4: `felyx-plane2-6654.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	1.6225	1.6225	6	26.422	26.422
2	7.3227	7.3227	7	43.945	43.945
3	9.5692	9.5692	8	52.061	52.061
4	18.453	18.453	9	52.078	52.078
5	25.681	25.681	10	53.736	53.736

Tabelle 7.4: Tiefste zehn Eigenfrequenzen für `felyx-plane2-6654.ansys`

7.5 Plane182

Spezifikationen für `felyx-plane182-bwopt-4731.ansys`:

Freiheitsgrade	9996
ρ	0.0027
SUBSP [s]	13.1
LANB [s]	19.1
FELyX [s]	7.6

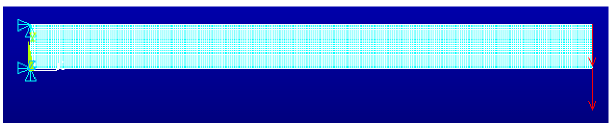

Abbildung 7.5: `felyx-plane182-bwopt-4731.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	0.228951	0.228951	6	11.6738	11.6738
2	1.43920	1.43920	7	13.5148	13.5148
3	3.94614	3.94614	8	16.4226	16.4226
4	4.45516	4.45516	9	21.4727	21.4727
5	7.43561	7.43561	10	22.8690	22.8690

Tabelle 7.5: Tiefste zehn Eigenfrequenzen für `felyx-plane182-bwopt-4731.ansys`

7.6 Plane183

Spezifikationen für `felyx-plane183-2500.ansys`:

Freiheitsgrade	15398
ρ	0.0027
SUBSP [s]	80.4
LANB [s]	45.3
FELyX [s]	45.7

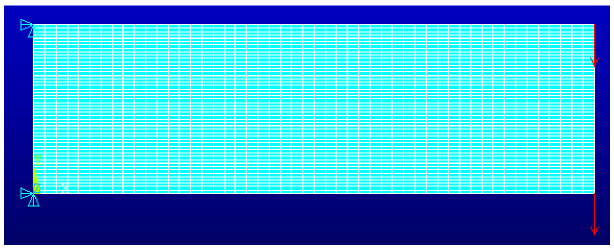

Abbildung 7.6: `felyx-plane183-2500.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	1.63086	1.63086	6	29.0269	29.0273
2	7.41467	7.41468	7	43.2297	43.2301
3	9.38726	9.38726	8	50.8702	50.8703
4	18.3010	18.3011	9	53.3324	53.3327
5	27.8620	27.8621	10	63.0181	63.0187

Tabelle 7.6: Tiefste zehn Eigenfrequenzen für `felyx-plane183-2500.ansys`

7.7 Solid185

Spezifikationen für `felyx-solid185-200.ansys`:

Freiheitsgrade	1350
ρ	0.0027
SUBSP [s]	2.20
LANB [s]	1.52
FELyX [s]	1.14

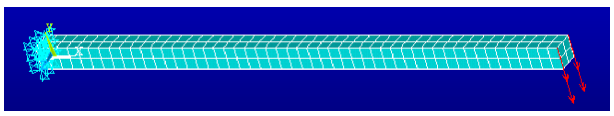

Abbildung 7.7: `felyx-solid185-200.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	0.663850	0.663850	6	11.4807	11.4807
2	0.663850	0.663850	7	15.7895	15.7895
3	4.13633	4.13633	8	22.2247	22.2247
4	4.13633	4.13633	9	22.2247	22.2247
5	11.4807	11.4807	10	25.4991	25.4991

Tabelle 7.7: Tiefste zehn Eigenfrequenzen für `felyx-solid185-200.ansys`

7.8 Solid186

Spezifikationen für `felyx-solid186-200.ansys`:

Freiheitsgrade	4500
ρ	2.7e-6
SUBSP [s]	10.2
LANB [s]	13.9
FELyX [s]	6.8

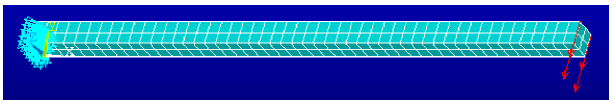

Abbildung 7.8: `felyx-solid186-200.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	20.8345	20.8345	6	358.773	358.773
2	20.8345	20.8345	7	460.774	460.774
3	129.615	129.615	8	691.746	691.747
4	129.615	129.615	9	691.746	691.747
5	358.773	358.773	10	806.219	806.219

Tabelle 7.8: Tiefste zehn Eigenfrequenzen für `felyx-solid186-200.ansys`

7.9 Solid187

Spezifikationen für `felyx-solid187-1453.ansys`:

Freiheitsgrade	8037
ρ	0.0027
SUBSP [s]	36.4
LANB [s]	33.8
FELyX [s]	33.0

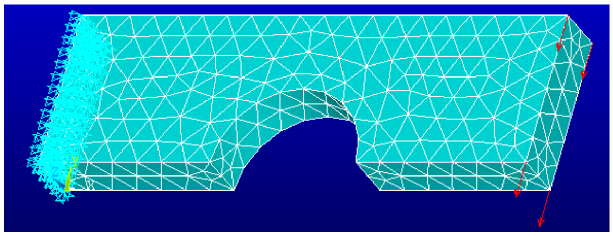

Abbildung 7.9: `felyx-solid187-1453.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	0.779944	0.779944	6	11.6794	11.6794
2	1.71854	1.71854	7	12.8275	12.8275
3	4.03810	4.03810	8	14.9571	14.9571
4	4.87118	4.87118	9	20.4464	20.4464
5	7.35133	7.35133	10	22.9424	22.9424

Tabelle 7.9: Tiefste zehn Eigenfrequenzen für `felyx-solid186-200.ansys`

7.10 2D-Mix

Spezifikationen für `felyx-2d-mixed-316.ansys`:

Freiheitsgrade	1391
ρ_1	2.7e-6
ρ_2	8e-6
SUBSP [s]	1.7
LANB [s]	2.0
FELyX [s]	1.0

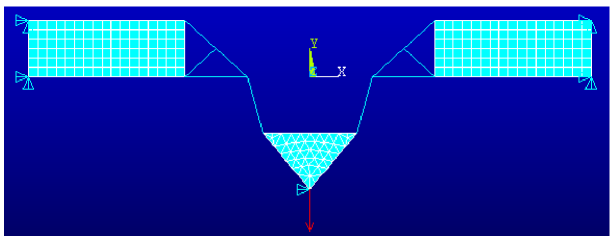

Abbildung 7.10: `felyx-2d-mixed-316.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	29.2206	29.2206	6	217.154	217.155
2	34.4458	34.4458	7	223.287	223.287
3	124.241	124.241	8	268.475	268.476
4	161.142	161.143	9	361.223	361.237
5	187.543	187.544	10	381.241	381.241

Tabelle 7.10: Tiefste zehn Eigenfrequenzen für `felyx-2d-mixed-316.ansys`

7.11 3D-Mix

Spezifikationen für `felyx-3d-mixed-718.ansys`:

Freiheitsgrade	6796
ρ_1	2.7e-6
ρ_2	8e-6
SUBSP [s]	27.7
LANB [s]	25.5
FELyX [s]	16.5

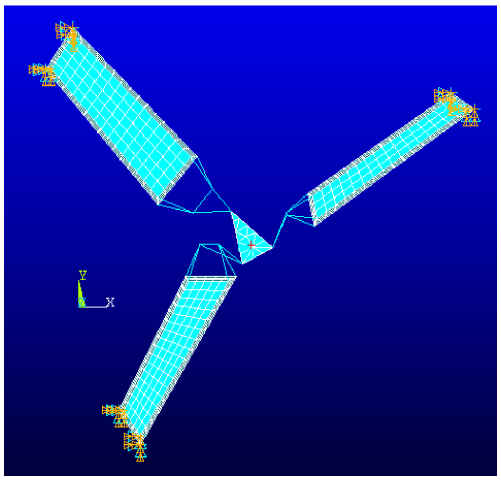

Abbildung 7.11: `felyx-3d-mixed-718.ansys`

Nr.	LANB	FELyX	Nr.	LANB	FELyX
1	2.18296	2.18295	6	16.9298	16.9299
2	6.01187	6.01289	7	17.5030	17.5012
3	6.22682	6.22696	8	31.4392	31.4397
4	10.9747	10.9785	9	33.7892	33.7893
5	14.4605	14.4558	10	39.6353	39.6356

Tabelle 7.11: Tiefste zehn Eigenfrequenzen für `felyx-3d-mixed-718.ansys`

7.12 Eigenvektoren

Die berechneten Eigenvektoren zu verifizieren gestaltet sich etwas umständlich. Das Beispiel aus Kapitel 2 wird mit 16 Elementen berechnet. Dies führt auf 48 Freiheitsgrade, also auch auf Eigenvektoren der Dimension 48.

In Anhang C sind die berechneten Eigenvektoren dargestellt. Die extremalen Verschiebungen in horizontale Richtung sind sehr klein, jene in vertikale Richtung sind in Abbildung 7.12 dargestellt. Vergleicht man die entsprechenden Einträge in den Vektoren mit den berechneten Eigenvektoren in Gleichung 2.14 erkennt man nach entsprechender Skalierung Übereinstimmung.

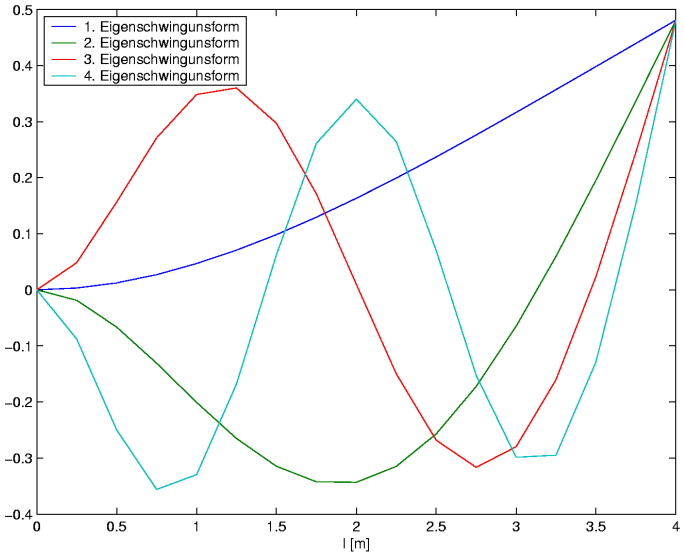

Abbildung 7.12: Erste vier Eigenformen des einseitig eingespannten Balken aus Kapitel 2

Kapitel 8

Benchmarking

8.1 Allgemeines

Zum Bestimmen der Effizienz des erstellten Eigensolvers wird ein Modell sowohl mit *FELyX* wie auch mit *Ansys6.0* gerechnet. Als Testmodell dient die in der y-z-Ebene eingespannte Struktur in Abbildung 8.1 mit Solid187 Elementen.
Alle mit *FELyX* errechneten Eigenfrequenzen stimmen exakt mit den Werten von *Ansys6.0* überein. In der Rechenzeit auf einem *SunBlade100, 450 MHz* gibt es aber Unterschiede.

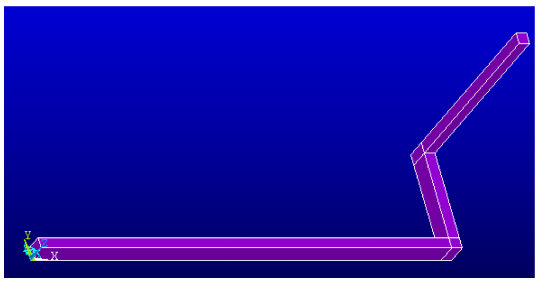

Abbildung 8.1: Testmodell

8.2 Anzahl Freiheitsgrade

In Tabelle 8.1 sind die Rechenzeiten zusammengefasst, um die Steifigkeits- und Massenamtrizen zu generieren und die tiefsten sechs Eigenfrequenzen und ihre Eigenvektoren der Struktur in Abbildung 8.1 zu berechnen.

Elemente	Freiheitsgrade	SUBSP	LANB	FELyX
311	2487	4	5	3.0
833	6165	11	15	8.2
1900	12804	25	34	23.4
5408	31584	80	111	92.9
9373	50043	192	313	206.9
12577	65547	291	509	295.6

Tabelle 8.1: Vergleich der gesamten Rechenzeiten

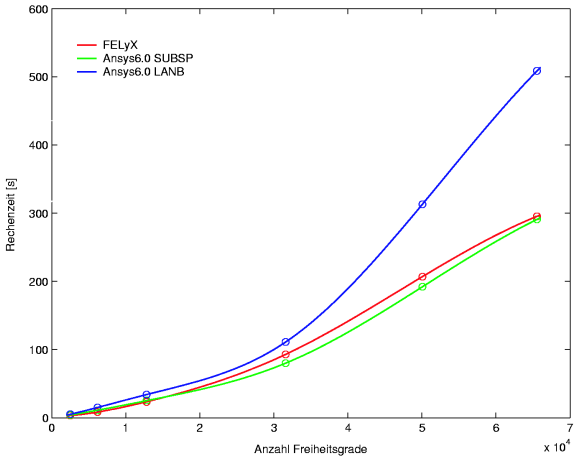

Abbildung 8.2: Vergleich der gesamten Rechenzeiten nach Anzahl Freiheitsgraden

8.3 Anzahl Eigenfrequenzen

Als Testmodell dient wieder die Struktur in Abbildung 8.1 mit 5408 Solid187 Elementen, das heisst 31584 Freiheitsgraden. In Tabelle 8.2 sind die Rechenzeiten zusammengefasst, um die Steifigkeits- und Massenamtrizen zu generieren und die tiefsten n Eigenfrequenzen und ihre Eigenvektoren zu berechnen.

n	SUBSP	LANB	FELyX
1	86	95	72
3	84	126	82
5	83	107	88
7	92	117	97
9	96	134	104
11	126	137	119
15	160	152	144
19	142	153	142

Tabelle 8.2: Vergleich der gesamten Rechenzeiten

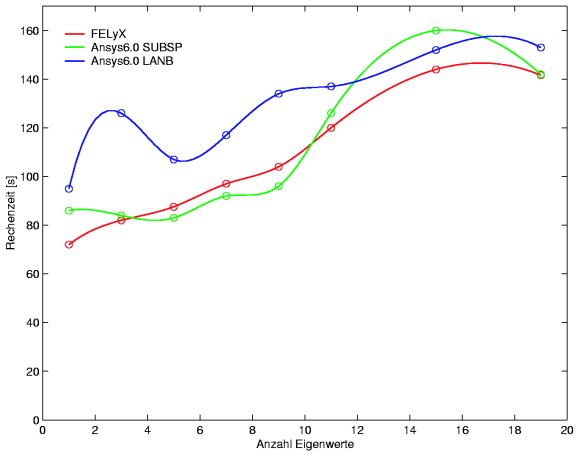

Abbildung 8.3: Vergleich der gesamten Rechenzeiten nach Anzahl Eigenwerten

Kapitel 9

Schlussbemerkungen

9.1 Dämpfungsmatrix C

Bei der Berechnung von Eigenfrequenzen und Eigenschwingungsformen wird von idealen Strukturen ohne Dämpfung ausgegangen. In der Realität tritt jedoch immer Dämpfung auf, so dass eine Schwingung ohne äussere Anregung mit der Zeit abklingt (Abschn. 1.1).

Dämpfung bewirkt, dass die Eigenfrequenzen in Wirklichkeit tiefer liegen als die im idealisierten Modell berechneten. Der Einfluss der Dämpfung auf die Eigenfrequenzen ist jedoch gering, dass diese in der Rechnung vernachlässigt werden darf [2]. Von der Dämpfung verursachte Kräfte sind in der Regel nur etwa 10 % so stark wie die anderen dynamischen Kräfte. Zur Bestimmung von Antworten eines Systems auf dynamische äussere Einflüsse muss die Dämpfung jedoch berücksichtigt werden.

Die Dämpfungsmatrix wird gleich wie die Massenmatrix aus den Elementmatrizen generiert. Die Elementdämpfungsmatrix wird analog zur Elementmassenmatrix (Gl. 3.14) aufintegriert.

$$\mathbf{c} = \sum_{i=1}^{k} w_i \, c \det(\mathbf{J}_i) \, \mathbf{N}^T \mathbf{N} \tag{9.1}$$

wobei c der Dämpfungskoeffizient des Elements ist.

9.2 Lumped Mass Matrix

Um die Eigenwertberechnung etwas zu vereinfachen, könnte anstatt der Elementmassenmatrizen in Kapitel 3 die *Lumped Mass Matrices* angewendet werden. Das Ziel besteht darin, in den Elementmassematrizen nur Einträge auf der Diagonalen zu haben. Dies geschieht so, dass nur die Einträge auf der Diagonalen gemäss Kapitel 3 berechnet und dann so skaliert werden, dass die Elementmasse erhalten wird. Für mehr Details sei auf [2] verwiesen. Als weitere Variante existiert *Optimal Lumping*. Bei der Verwendung dieser Methoden wird die Rechenzeit etwas kürzer auf Kosten der Genauigkeit der Resultate.

9.3 Simultane Generierung von K und M

Bei Vergleichen der Rechengeschwindigkeit von *FELyX* mit *Ansys6.0* fällt auf, dass die Generierung der globalen Steifigkeits- und Massenmatrix bei *FELyX* mehr als doppelt

so lange dauert als bei *Ansys6.0*. Bei *FELyX* werden die beiden Matrizen *nacheinander* aufgebaut, was zu Doppelspurigkeiten führt. Die Jacobi-Matrix **J** muss für jedes Element zweimal berechnet werden. Hier liesse sich Zeit sparen, wenn **K** und **M** *simultan* generiert würden.

Um statische Probleme zu lösen ist die Massenmatrix nicht nötig. Ihre Generierung muss sicher vermieden werden. Das ist bei der aufeinanderfolgenden Generierung einfach. Bei der simultanen Generierung muss jedoch "tiefer" im Code etwas verändert werden.

9.4 Ziel

Das Ziel der Arbeit ist, einen effizienten Algorithmus zur Eigenfrequenz- und Eigenschwingungsformberechnung in *FELyX* zu implementieren. Vom Standpunkt der Effizienz ist das Ziel erreicht. Der implementierte Lanczos-Algorithmus kann mit kommerziellen Anwendungen mithalten. Es können nur die tiefsten Eigenpaare ermittelt werden. Eine Berechnung der Eigenpaare irgendwo im Spektrum ist aber ausdrücklich nicht verlangt.

Vom Standpunkt der Zuverlässigkeit gibt es noch kleine Makel. Es kann gelegentlich vorkommen, dass Eigenpaare ausgelassen werden (Abschn. 6.5). Bei Tests geschieht das aber selten, und wenn, dann nicht bei den tiefsten zehn Eigenwerten. Gefordert ist, dass die tiefsten etwa zehn Eigenpaare berechnet werden können. Der Algorithmus ist also sicher gut für den alltäglichen Einsatz brauchbar.

9.5 Matlab

Die Implementierung eines Rechenalgorithmus direkt in *C++* ist schwierig, weil dabei ein kleiner Fehler zu falschen Ergebnissen oder zum Abbruch der Berechnung führt. Als hilfreiche Anwendung hat sich dabei *Matlab* erwiesen, als Zwischenschritt zwischen dem Flowchart auf Papier und der Implementierung in *C++*. Weil *Matlab* eine "hohe" Programmiersprache ist, können Algorithmen unkompliziert auf wenigen Zeilen programmiert werden (Anh. A, B). Sobald der Rechenablauf einwandfrei funktioniert, kann mit der Implementation in *C++* begonnen werden. Falls dabei schliesslich falsche Ergebnisse resultieren, können mit *Matlab* Schritt für Schritt die Zwischenergebnisse kontrolliert und so der Fehler lokalisiert werden. Dieses Vorgehen hat sich bewährt und als effizient erwiesen.

9.6 To do

Um die Arbeit zu kompletisieren und optimieren, könnten folgende Aufgaben erledigt werden:

- **Massenmatrix der Shell-Elemente.** Während dem Verlauf der Arbeit sind die Shell-Klassen noch in Bearbeitung. Die Generierung der Massenmatrix wird nicht implementiert.
- **Block-Lanczos-Verfahren.** Um auch mehrfache Eigenwerte sicher darzustellen, kann die Block-Lanczos-Methode [3] implementiert werden (Abschn. 6.5).
- **Spektralverschiebung.** Um auch Eigenpaare an einer bestimmten Stelle im ganzen Spektrum ermitteln zu können, kann die Spektralverschiebung des Eigenwert-

problems um μ implementiert werden. Dabei ist ein anderer Zerlegungsalgorithmus für die Matrix $\mathbf{F}$ zu wählen, da diese nicht mehr positiv definit ist (Abschn. 6.1).

Literaturverzeichnis

[1] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst: *Templates for the Solution of Algebraic Eigenvalue Problems*, Siam, 2000

[2] R. Cook, D. Marlkus, M. Plesha, R. Witt: *Concepts and Applications of Finite Element Analysis*, Wiley, 2002

[3] R. Grimes, J. Lewis, H. Simon: *A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems*, Journal on Matrix Analysis and Applications, Vol. 15, 1994

[4] H.R. Schwarz: *Methode der finiten Elemente*, Teubner, 1991

[5] H.R. Schwarz: *FORTRAN-Programme zur Methode der finiten Elemente*, Teubner, 1991

[6] G.W. Stewart: *Matrix Algorithms, Volume II*, Siam, 2001

[7] T. Yokoyama: *Vibrations of a Hanging Timoshenko Beam Under Gravity*, Journal of Sound and Vibration, Vol. 141, 1990

[8] O.C. Zienkiewicz, R.L. Taylor: *The Finite Element Method*, Butterworth, Heinemann, 2000

[9] *Ansys Theory Reference*, `www1.ansys.com/customer/content/documentation/60/theory_toc.html`

[10] T. Irvine: *The Tacoma Narrows Bridge Failure*, `www.vibrationdata.com/Tacoma.htm`

[11] P. Dayal, M. Troyer: *The Iterative Eigenproblem Template library*, `www.comp-phys.org/software/ietl/ietl.html`, 2002

[12] M. Ramek: *Mathematik Tutorial: Das Jacobi-Verfahren*, `fptchal1.tu-graz.ac.at:8000/quanten/mjacobi.html`, 2002

Anhang A

Matlab: Simultane Vektoriteration

Programmcode für die simultane Vektoriteration (Kap. 5).

```
function [ev] = iteration(A,B,p,loops)
% A, B:  Matrizen der allgemeinen Eigenwertaufgabe
% p:     Anzahl Eigenwerte
% loops: Anzahl Iterationen

[m,n]=size(A);

LAT = chol(A);
LA = LAT';
LBT = chol(B);
LB = LBT';

Y=zeros(n,p);
H1=zeros(n,p);
H2=zeros(n,p);
H3=zeros(n,p);

for i=1:p Y(i,i)=1; end

for k=1:loops

    H1=LB*Y;
    H2=LA^(-1)*H1;
    H3=LAT^(-1)*H2;
    Z=LBT*H3;

    [Q,R]=qr(Z);

    C1=R*R';

    G=orth(C1);
    D=G'*C1*G;

    Y=Q*G;
```

```
end

for j=1:p
    x=LBT^(-1)*Y(:,j);
    ev(j)=sqrt((x'*A*x)/(x'*B*x));
end
```

Programmcode für die Orthogonalisierung (Abschn. 5.3).

```
function [U] = or2(C,loops)
% C:     zu orthogonalisierende Matrix
% loops: Anzahl Iterationen
[m,n]=size(C);

U=eye(m);

for k=1:loops
    temp=0;
    for j=1:m
        for i=1:j-1
            if (abs(C(j,i))>temp)
                temp=abs(C(j,i)); jj=j; ii=i;
            end
        end
    end
    temp=2*C(jj,ii) / (C(jj,jj) - C(ii,ii))
    alpha=.5*atan (temp)
    V=eye(m);
    V(ii,ii)=cos(alpha);
    V(jj,jj)=cos(alpha);
    V(jj,ii)=-sin(alpha);
    V(ii,jj)=sin(alpha);

    C=(V'*C)*V;
    U=U*V;
end
```

Anhang B

Matlab: Verfahren von Lanczos

Programmcode für das Verfahren von Lanczos (Kap. 6).

```
function [d,x] = lanczos(A,B,nev,mu,jmax)
% A, B: Matrizen des allgemeinen Eigenwertproblems
% nev:  Anzahl der zu bestimmenden Eigenpaare
% mu:   Spektralverschiebung
% jmax: maximale Anzahl Lanczos-Schritte
% d, x: Eigenwerte bzw. -vektoren

[m,n]=size(A);
r=rand(m,1);
h=B*r;
beta(1)=sqrt(h'*r);
q=zeros(m,1);
tolconv=0.00000000000000000001;

for j=1:jmax
    q(:,j)=r/beta(j);
    h=h/beta(j);
    u=(A-mu*B)^(-1)*h;      % Loesen des Gleichungssystems
    if j>1, r=u-beta(j)*q(:,j-1);
    else r=u; end;
    alpha(j)=r'*h;
    r=r-alpha(j)*q(:,j);
    r=r-q*(q'*(B*r));       % Orthogonalisierung
    h=B*r;
    beta(j+1)=sqrt(r'*h);
    T= diag(alpha)+diag(beta(2:j),1)+diag(beta(2:j),-1);
    bndv=abs(beta(j+1)*s(j,:)); % Konvergenztest
    anorm=max(abs(diag(d)));
    convd=bndv<tolconv*anorm;
    nconv=sum(convd);
    if nconv>=nev , break; end;
end

d=ewerte(T,nev);            % d: Eigenwerte
```

```
[ll,n]=size(d);

% Berechnung der Eigenvektoren
for j=1:n
    a(j,1)=mu+1/d(j);
    for i=1:jmax-1
        al(i)=alpha(i)-d(j);
        be(i)=beta(i+1);
        ga(i)=beta(i+1);
    end;
    al(jmax)=alpha(jmax)-d(j);
    be(jmax)=0;
    for i=1:jmax-1
        s1=abs(al(i))+abs(be(i));
        s2=abs(ga(i))+abs(al(i+1))+abs(be(i+1));
        if(abs(al(i))/s1 >= abs(ga(i))/s2)
            de(i)=0;
            uu=ga(i);
            u1=al(i+1);
            u2=be(i+1);
            vert(i)=0;
        else
            uu=al(i); u1=be(i); u2=0; al(i)=ga(i);
            be(i)=al(i+1); de(i)=be(i+1); vert(i)=1;
        end;
        ga(i)=uu/al(i); al(i+1)=u1-ga(i)*be(i);
        be(i+1)=u2-ga(i)*de(i);
    end;
    for i=1:jmax
        v1(i)=1; v(i)=0;
    end;
    iter=0;
    dif=1;
    while (dif >= 1e-8)
        v1(jmax)=-v1(jmax)/al(jmax);
        s=v1(jmax);
        v1(jmax-1)=-(v1(jmax-1)+be(jmax-1)*v1(jmax))/al(jmax-1);
        if (abs(v1(jmax-1))>abs(s)) s=v1(jmax-1); end;
        for i=jmax-2:-1:1
            v1(i)=-(v1(i)+be(i)*v1(i+1)+de(i)*v1(i+2))/al(i);
            if(abs(v1(i))>abs(s)) s=v1(i); end;
        end;
        zz(:,iter+1)=v1';
        dif=0; vn=0;
        for i=1:jmax
            aux=v(i);
            v(i)=v1(i)/s;
            v1(i)=v(i);
            vn=vn+v(i)^2;
```

```
            dif=max(dif,abs(aux-v(i)));
        end;
        if (dif >= 1e-8)
            if (iter>4) fprintf('keine konvergenz!   ');  break;
            end;
            for i=1:jmax-1
                if (vert(i)==0) uu=v1(i+1);
                else uu=v1(i); v1(i)=v1(i+1); end;
                v1(i+1)=uu-ga(i)*v1(i);
            end;
            iter=iter+1;
        end;
    end;
    vn=sqrt(vn);
    v=v/vn;
    for i=1:m
        s=0;
        for l=1:jmax
            s=s+v(l)*q(i,l);
        end;
        x(i,j)=s;
    end;
    rhoj=beta(jmax+1)*abs(v(jmax));
end;
```

Programmcode für die Bisektionsmethode (Abschn. 6.3).

```
function [ev] = ewerte(T,nev)
% T:   T-Matrix (tridiagonal)
% nev: Anzahl der gewuenschten Eigenwerte

[m,n]=size(T);

for i=1:n
    zeile(i)=sum(T(i,:));
end;
b=max(zeile);
a=-b;

for nrev=1:nev
    an=a; bn=b;
    for k=1:50
        mu=(an+bn)/2;
        q(1)=mu-T(1,1);
        if q(1)==0, q(k)=-eps; end;
        for k=2:n
            q(k)=(mu-T(k,k))-T(k-1,k)^2/q(k-1);
```

```
            if q(k)==0, q(k)=-eps; end;
        end;
        vf=0;
        for k=1:n
            if q(k)<0, vf=vf+1; end;
        end;

        if vf>=nrev, an=mu;
        else bn=mu; end;
    end;
    ev(nrev)=(an+bn)/2;
    b=bn;
end;
```

Anhang C

Eigenvektorberechnung

Mit *FELyX* berechnete Eigenvektoren zu den tiefsten vier Eigenwerten der Aufgabe in Kapitel 2 mit 16 Beam3-Elementen.

$$
\begin{bmatrix}
2.0024e{-}17 & -5.9001e{-}16 & -4.0367e{-}13 & -1.1547e{-}08 \\
0.0032092 & -0.018641 & 0.048488 & -0.087607 \\
0.025294 & -0.14088 & 0.35014 & -0.59783 \\
1.2918e{-}18 & -5.6318e{-}16 & -7.4874e{-}14 & -2.1468e{-}09 \\
0.012458 & -0.066338 & 0.15653 & -0.24977 \\
0.048316 & -0.23256 & 0.47848 & -0.61015 \\
6.7996e{-}18 & -5.9877e{-}16 & 1.2164e{-}13 & 3.4767e{-}09 \\
0.027178 & -0.13092 & 0.27115 & -0.35606 \\
0.069071 & -0.27629 & 0.40879 & -0.18742 \\
3.9399e{-}18 & -6.016e{-}16 & 2.2223e{-}13 & 6.3666e{-}09 \\
0.046805 & -0.20072 & 0.3485 & -0.32964 \\
0.087574 & -0.27498 & 0.19076 & 0.4 \\
1.0081e{-}17 & -5.0841e{-}16 & 2.5562e{-}13 & 7.3456e{-}09 \\
0.07078 & -0.26506 & 0.36013 & -0.16809 \\
0.10385 & -0.23357 & -0.10364 & 0.84602 \\
1.143e{-}17 & -6.322e{-}16 & 2.4557e{-}13 & 7.0522e{-}09 \\
0.09855 & -0.31473 & 0.29719 & 0.062555 \\
0.11795 & -0.15893 & -0.39209 & 0.92846 \\
9.0832e{-}18 & -5.1654e{-}16 & 2.0708e{-}13 & 5.9716e{-}09 \\
0.12958 & -0.34247 & 0.17101 & 0.26135 \\
0.12994 & -0.059674 & -0.59822 & 0.60022 \\
1.9854e{-}17 & -5.5937e{-}16 & 1.5458e{-}13 & 4.4648e{-}09 \\
0.16335 & -0.34333 & 0.0095698 & 0.34013 \\
0.13989 & 0.054473 & -0.66764 & 0.0059066 \\
1.6207e{-}17 & -6.3173e{-}16 & 9.6541e{-}14 & 2.7922e{-}09 \\
0.19936 & -0.31484 & -0.14956 & 0.26451 \\
0.14793 & 0.17335 & -0.57937 & -0.58596 \\
1.988e{-}18 & -5.3898e{-}16 & 3.8435e{-}14 & 1.1358e{-}09 \\
0.23716 & -0.25708 & -0.26824 & 0.070202 \\
0.15418 & 0.2872 & -0.34968 & -0.90545 \\
1.615e{-}18 & -6.0846e{-}16 & -1.4069e{-}14 & -3.8254e{-}10 \\
0.27632 & -0.17238 & -0.31661 & -0.15252 \\
0.15882 & 0.38755 & -0.027107 & -0.80314 \\
1.0581e{-}17 & -5.9385e{-}16 & -5.939e{-}14 & -1.6855e{-}09 \\
0.31645 & -0.064965 & -0.27971 & -0.2988 \\
0.16203 & 0.46804 & 0.32039 & -0.3158 \\
6.4072e{-}18 & -5.6436e{-}16 & -9.5534e{-}14 & -2.727e{-}09 \\
0.35723 & 0.059687 & -0.16016 & -0.29494 \\
0.16404 & 0.52516 & 0.62353 & 0.35507 \\
1.1652e{-}17 & -6.1806e{-}16 & -1.2148e{-}13 & -3.482e{-}09 \\
0.39839 & 0.19565 & 0.024 & -0.12828 \\
0.1651 & 0.55881 & 0.8312 & 0.94437 \\
3.9604e{-}18 & -6.9965e{-}16 & -1.3702e{-}13 & -3.9384e{-}09 \\
0.43973 & 0.33744 & 0.24603 & 0.15412 \\
0.16551 & 0.57279 & 0.92755 & 1.2631 \\
9.3319e{-}18 & -4.8995e{-}16 & -1.4322e{-}13 & -4.0909e{-}09 \\
0.48112 & 0.48105 & 0.48096 & 0.48088 \\
0.16557 & 0.575 & 0.94394 & 1.3225
\end{bmatrix}
\tag{C.1}
$$