



**Fachbereich Maschinenbau und Verfahrenstechnik**  
**Lehrstuhl für Maschinenelemente und Getriebetechnik**  
**Prof. Dr.-Ing. Bernd Sauer**

## **Studienarbeit**

---


**Implementierung eines effizienten numerischen  
Eigenwertlösers in C++**

---

vorgelegt von  
Herrn Fahad Khan  
413060

Betreuer:  
Markus Hofmann, M.Sc.  
Jun. Prof. Dr.-Ing. Manuel Oehler

Kaiserslautern, Juli 2021

 <b>TECHNISCHE UNIVERSITÄT KAISERSLAUTERN</b>	Fachbereich Maschinenbau und Verfahrenstechnik <b>Lehrstuhl für Maschinenelemente und Getriebetechnik</b> Prof. Dr.-Ing. B. Sauer	<b>Nr. 1138/21</b>
--	---	------------------------

# Studienarbeit

VON

**Herrn cand.-Ing. Fahad Khan**

MATR.-NR. 413060

## **THEMA:**

**Implementierung eines effizienten numerischen Eigenwertlösers in C++**

Eigenfrequenzen von technischen Systemen sind von zentraler Bedeutung im Maschinenbau. Zur rechnerischen Bestimmung von Eigenfrequenzen ist es notwendig, Eigenwerte und Eigenvektoren von großen Matrizen effizient bestimmen zu können. Hierzu sollen ausgehend von der bestehenden Literatur verschiedene Eigenwertlöser in C++ implementiert werden und ihre Laufzeit anhand eines Beispielproblems aus der Schwingungstechnik verglichen werden.

Folgende Aufgaben sind Bestandteil der Arbeit:

- Recherche zum Stand der Technik
  - o Dies betrifft sowohl die mathematische als auch die maschinenbauliche Literatur
- Implementierung von Eigenwertalgorithmen in C++
- Anwendung und Bewertung der Algorithmen anhand eines Schwingungsproblems
- Dokumentation der Ergebnisse

**Betreuer: Markus Hofmann, M.Sc.**

Kaiserslautern, den 22. Juli 2021

Jun. Prof. Dr.-Ing. Manuel Oehler

## Erklärung

**Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig, ohne unerlaubte Hilfe, ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel einschließlich Internet und nach wissenschaftsethischen Kriterien verfasst habe. Die den benutzen Quellen wörtlich oder inhaltlich entnommenen Stellen sind als solche kenntlich gemacht.**

**Kaiserslautern, 22.Juli.2021**

**Ort, Datum, Unterschrift**

A handwritten signature in black ink, appearing to read 'Friedrich' followed by a flourish.

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung .....</b>	<b>1</b>
1.1	Definition von Eigenwerten und Eigenvektoren .....	1
1.2	Relevanz des Themas und Motivation.....	1
1.2.1	Schwingungstechnik.....	1
1.2.1	Eigenwertanalyse des Balkens .....	4
<b>2</b>	<b>Berechnung von Eigenwerten und Eigenvektoren .....</b>	<b>8</b>
2.1	Analytisch.....	8
2.2	Numerische Eigenwert-Algorithmen.....	10
2.2.1	Power-Iteration.....	10
2.2.1.1	Beispiel.....	11
2.2.2	QR Algorithmus .....	12
2.2.2.1	QR-Zerlegung mittel Gram Schmidt .....	12
	Beispiel.....	13
2.2.2.2	Basic QR-Algorithmus .....	14
2.2.2.3	Laufzeitanalyse.....	14
2.2.3	Beschleunigter QR-Algorithmus.....	15
2.2.3.1	Hessenberg-Form .....	15
2.2.3.2	Hessenberg-QR-Algorithmus .....	17
2.2.3.3	Laufzeitanalyse.....	18
2.2.3.4	Hessenberg-QR-Algorithmus mit Shift.....	19
<b>3</b>	<b>Implementierung in C++.....</b>	<b>21</b>
3.1	Mathematischer Datentypen und Matrix/Vektor-Operationen.....	21
3.2	Implementierung Eigenwert-Algorithmen .....	25
<b>4</b>	<b>Schwingungstechnik Beispiel.....</b>	<b>29</b>
<b>5</b>	<b>Zusammenfassung und Ausblick .....</b>	<b>34</b>
<b>6</b>	<b>Literaturverzeichnis .....</b>	<b>35</b>

## Formelzeichen

Zeichen	Bedeutung	Einheit
$A$	beliebige Matrix	
$G(i, j, \vartheta)$	Givens-Rotation	
$H$	Hessenbergmatrix	
$I$	Identitätsmatrix	
$\ell$	Abstand	m
$M$	Masse	kg
$e$	normalisierter Vektor	
$Q$	orthogonale Matrix	
$R$	obere dreieckmatrix	
$T$	Kraft	N
$U$	Transformationmatrix	
$v$	Eigenvektor	
$\lambda$	Eigenwerte	
$\sigma_k$	Rayleigh-Quotientverschiebung	

# 1 Einführung

In den Ingenieur- und Naturwissenschaften besteht das Eigenwertproblem hauptsächlich darin, Eigenwerte und Eigenvektoren des Systems zu finden. Sie geben einen Einblick in die Eigenschaften eines Systems und helfen, das Verhalten des Systems zu identifizieren.

## 1.1 Definition von Eigenwerten und Eigenvektoren

Sei  $A$  eine quadratische  $(n \times n)$ -Matrix, mit  $n$  Spalten und  $n$  Zeilen. Gibt es eine Zahl  $\lambda$  und einen vom Nullvektor verschiedenen Vektor  $v$ , sodass die Gleichung

$$Av = \lambda v \quad (1.1)$$

erfüllt ist, so nennt man  $\lambda$  einen **Eigenwert** von  $A$  und  $v$  einen zugehörigen **Eigenvektor** von  $A$ .

## 1.2 Relevanz des Themas und Motivation

Die Eigenwertanalyse hat eine breite Anwendung im Bereich des Maschinenbaus. Fast jede mechanische Struktur ist anfällig für Schwingungen. In diesem Zusammenhang ist es sehr wichtig, die Eigenfrequenzen zu bestimmen. Wenn eine Struktur bei einer bestimmten Eigenfrequenz schwingt, verformt sie sich in eine entsprechende Form, die als Eigenmode bezeichnet wird.

Die Hauptziele der Eigenwertanalyse sind:

1. Um sicherzustellen, dass keine übermäßigen Spannungen oder Geräuschemissionen aufgrund von Resonanzen durch periodische Erregung entstehen können.
2. Prüfen, ob alle Eigenfrequenzen einer Struktur im Vergleich zu den Frequenzen der Belastung hoch sind.
3. Untersuchung der geeigneten Auswahl von Frequenzen für eine dynamische Response-Analyse.
4. Einblick in die Auswirkung von Konstruktionsänderungen auf eine bestimmte Eigenfrequenz durch Untersuchung ihrer Modenform.

Im Folgenden werden einige Beispiele für die Eigenwertanalyse besprochen.

### 1.2.1 Schwingungstechnik

Eigenwerte und Eigenvektoren finden in vielen Anwendungsfeldern Verwendung. Die Schwingungsanalyse ist eine häufige Anwendung des Eigenwertproblems. Eigenfrequenzen von Schwingungen und Schwingungsformen werden aus Eigenwerten bzw. Eigenvektoren abgeleitet.

Als Beispiel ist unten ein System mit fünf Massen dargestellt. Die Massen sind über eine Schnur mit gleichmäßiger Spannung  $T$  verbunden. Die Massen sind in gleichem Abstand  $\ell$  voneinander angeordnet. Jede Masse befindet sich zunächst in Gleichgewichtslage an der Position  $x_i$ .

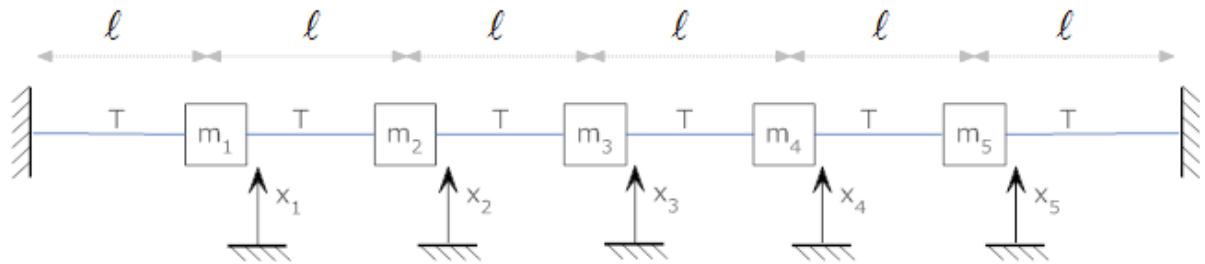


Abbildung 1.2: Schwingungssystem mit fünf Massen (Cheever 2005)

Die Massen werden aus dem Gleichgewicht gebracht, um Bewegungsgleichungen herzuleiten.

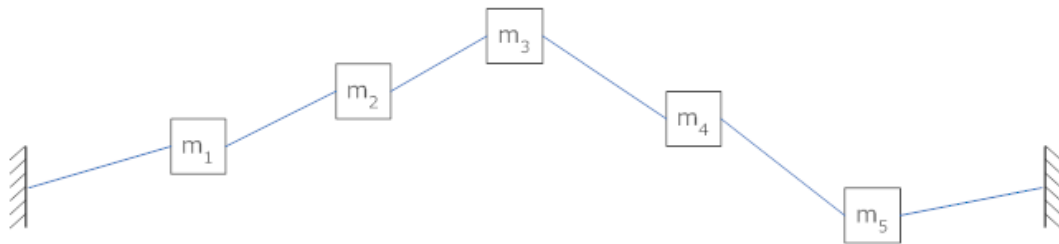


Abbildung 1.3: aus dem Gleichgewicht gebrachte Massen (Cheever 2005)

Man sieht, was mit  $m_1$  unter der Verschiebung geschieht.

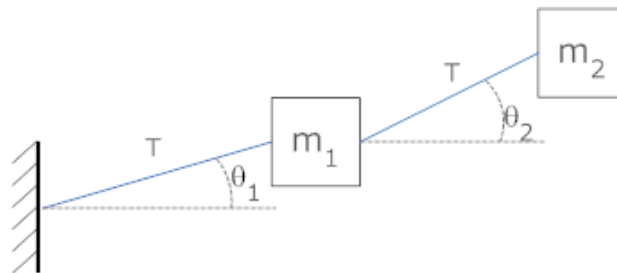


Abbildung 1.4:  $m_1$  unter Verschiebung (Cheever 2005)

Die auf  $m_1$  wirkenden Kräfte können aus der folgenden Abbildung abgeleitet werden.

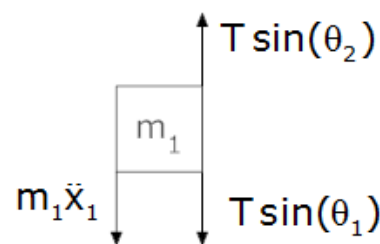


Abbildung 1.5: Auf  $m_1$  wirkenden Kräfte (Cheever 2005)

Wenn man annimmt, dass sowohl die Breite der Massen als auch ihre vertikale Verschiebung klein sind im Vergleich zum Abstand  $\ell$ , dann kann man die auf das Massenelement wirkenden Kräfte nach dem zweiten newtonschen Bewegungsgesetz schreiben als:

$$\sin(\theta_1) \approx \frac{x_1}{\ell} \quad \sin(\theta_2) \approx \frac{x_2 - x_1}{\ell}$$

### 3 Einführung

$$m_1 \ddot{x}_1 + T \sin(\theta_1) - T \sin(\theta_2) = 0$$

$$m_1 \ddot{x}_1 + T \frac{x_1}{\ell} - T \frac{x_2 - x_1}{\ell} = 0$$

$$\ddot{x}_1 = -2 \frac{T}{m_1} x_1 + \frac{T}{m_1} x_2$$

Dasselbe Verfahren kann für  $m_2$  wiederholt werden, wie unten dargestellt.

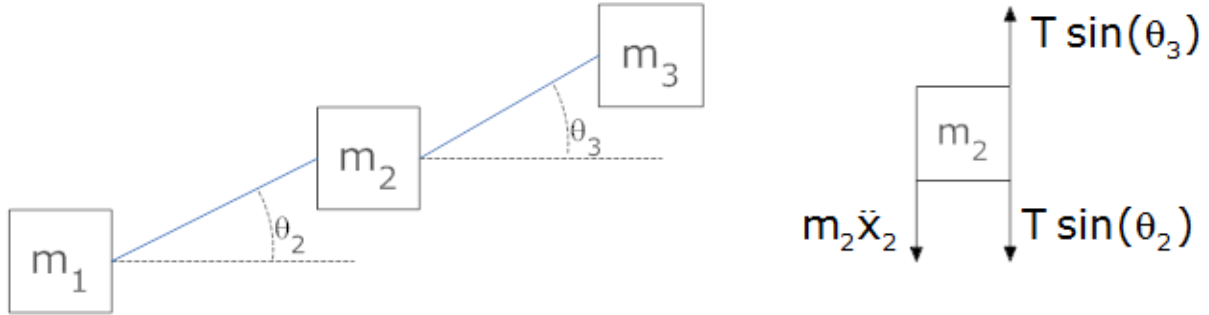


Abbildung 1.6: Auf  $m_2$  wirkenden Kräfte (Cheever 2005)

$$m_2 \ddot{x}_2 + T \sin(\theta_2) - T \sin(\theta_3) = 0$$

$$m_2 \ddot{x}_2 + T \frac{x_2 - x_1}{\ell} - T \frac{x_3 - x_2}{\ell} = 0$$

$$\ddot{x}_2 = \frac{T}{m_2} x_1 - 2 \frac{T}{m_2} x_2 + \frac{T}{m_2} x_3$$

Gleiches gilt für  $m_3$ ,  $m_4$  und  $m_5$ .

$$\ddot{x}_3 = \frac{T}{m_3} x_2 - 2 \frac{T}{m_3} x_3 + \frac{T}{m_3} x_4$$

$$\ddot{x}_4 = \frac{T}{m_4} x_3 - 2 \frac{T}{m_4} x_4 + \frac{T}{m_4} x_5$$

$$\ddot{x}_5 = \frac{T}{m_5} x_4 - 2 \frac{T}{m_5} x_5$$

Wenn man  $m_1 = m_2 = m_3 = m_4 = m_5 = T = 1$  wählt, erhält man

$$\ddot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \mathbf{x} \quad (1.2)$$

Das Verhalten des Systems kann durch Eigenwerte und dann durch Eigenvektoren dieser Matrix dargestellt werden. Die Eigenwerte sind für die Schwingungsfrequenzen verantwortlich und die Eigenvektoren führen zur Modenform des Systems (Cheever 2005).

Für  $n$  Massen kann die Systemmatrix geschrieben werden als



$$A = \begin{bmatrix} \overbrace{\begin{matrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{matrix}}^n \end{bmatrix}^n$$

Am Ende dieser Arbeit werden mit Hilfe der entwickelten Methoden die Eigenwerte und Eigenvektoren numerisch berechnet.

### 1.2.1 Eigenwertanalyse des Balkens

Die Bestimmung der Eigenfrequenzen von Strukturen unter dynamischen Bedingungen ist sehr wichtig. Alle Objekte haben eine Eigenfrequenz oder einen Satz von Frequenzen, mit denen sie schwingen. Wenn diese Eigenfrequenzen mit den externen Anregungsfrequenzen übereinstimmen, beginnt die Struktur mit maximaler Amplitude zu schwingen, d. h. es tritt Resonanz auf, was zu einer Beschädigung der Struktur führt. Dies kann vermieden werden, wenn die Eigenfrequenzen der Struktur bekannt sind. Dann können Änderungen an der Struktur vorgenommen werden, um Resonanzen zu vermeiden.

Die Eigenfrequenzen von Strukturen können mit Hilfe von FE-Solver effizient ermittelt und später mit analytischen oder experimentellen Ergebnissen verglichen werden. In diesem Beispiel werden die Eigenfrequenzen eines Freitragers mit ANSYS-Workbench 2020 R2 ermittelt und die Ergebnisse anschließend mit analytisch ermittelten Eigenfrequenzen verglichen.

Als Ausgangspunkt, die diskrete Formulierung der Schwingungsbewegung einer elastischen Struktur als Funktion der Systemmasse, der Steifigkeit, der Dämpfung und der aufgebrachten Lasten lautet:

$$[M]\{\ddot{x}(t)\} + [B]\{\dot{x}(t)\} + [K]\{x(t)\} = \{P(t)\} \quad (1.3)$$

Wobei:

$[M]$  = globale Massenmatrix,  $[K]$  = globale Steifigkeitsmatrix

$[B]$  = globale Dämpfungsmatrix,  $\{P\}$  = globaler Lastvektor

Um den Eigenwert eines Systems zu finden, ist es notwendig, die aufgebrachten Lasten gleich Null zu setzen. Nur dann können die Eigenfrequenzen des Systems ermittelt werden. Die Bewegungsgleichung für freie Schwingungen ohne Dämpfung und aufgebrachte Lasten kann geschrieben werden als:

$$[M]\{\ddot{x}(t)\} + [K]\{x(t)\} = 0 \quad (1.4)$$

In den meisten Schwingungsfällen wird eine harmonische Schwingung beobachtet. Unter der Annahme einer sinusförmigen Schwingung kann die Verschiebung beschrieben werden durch:

$$\{\mathbf{x}(t)\} = \{v\}e^{i\omega t} \quad (1.5)$$

$\omega$  ist die Kreisfrequenz (Radiant pro Sekunde). Nach der Doppelableitung von Gleichung 1.5, die Gleichung für die Beschleunigung kann geschrieben werden als:

$$\{\ddot{\mathbf{x}}(\mathbf{t})\} = -\omega^2 \{v\} e^{i\omega t} \quad (1.6)$$

Die Bewegungsgleichung wird:

$$(-\omega^2[\mathbf{M}] + [\mathbf{K}])\{v\}e^{i\omega t} = 0 \quad (1.7)$$

## 5 Einführung

Da  $e^{i\omega t}$  niemals Null ist, kann die Gleichung in die Form eines allgemeinen Eigenwertproblems umgestellt werden:

$$[K] - \lambda[M]\{v\} = 0 \quad (1.8)$$

Wobei:

$\lambda$  ist der Eigenwert für jede Mode, der die Eigenfrequenz ergibt.

$\{v\}$  ist der Eigenvektor für jede Mode, der die Eigenmodenform darstellt

Der Eigenwert bezieht sich auf die Eigenfrequenz des Systems, basierend auf den Freiheitsgraden der Systeme, wird jede Eigenfrequenz wie folgt geschrieben:

$$\lambda_i = \omega_i^2 \quad i = 1, \dots, n \quad (1.9)$$

$\lambda = \omega^2$  wird als Eigenwert bezeichnet, und jedem Wert ist ein eindeutiges  $\{v\}$  zugeordnet, das als Eigenvektor bezeichnet wird.

Im Folgenden werden die Eigenfrequenzen und Eigenformen eines Freitragers in ANSYS Workbench 2020 R2 berechnet. Die Analyse wird im Modalanalysebereich von ANSYS durchgeführt.

Hier wird ein Träger mit folgenden Eigenschaften analysiert:

Material: Baustahl;  $E = 210\text{GPa}$

Dimension: Länge = 450mm, Breite = 20mm, Höhe = 3mm

Zwangsbedingung: An einem Ende fixiert

Wenn das Objekt komplex ist, wird es normalerweise in der CAD-Software erstellt. Im Fall des Freitragers ist es ziemlich unkompliziert und kann leicht mit ANSYS DesignModeler erstellt werden.

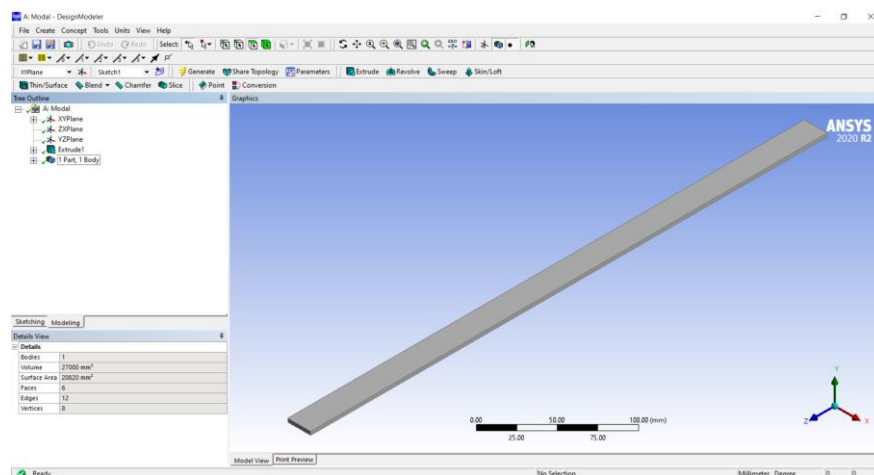


Abbildung 1.7: Träger modelliert in ANSYS DesignModeler

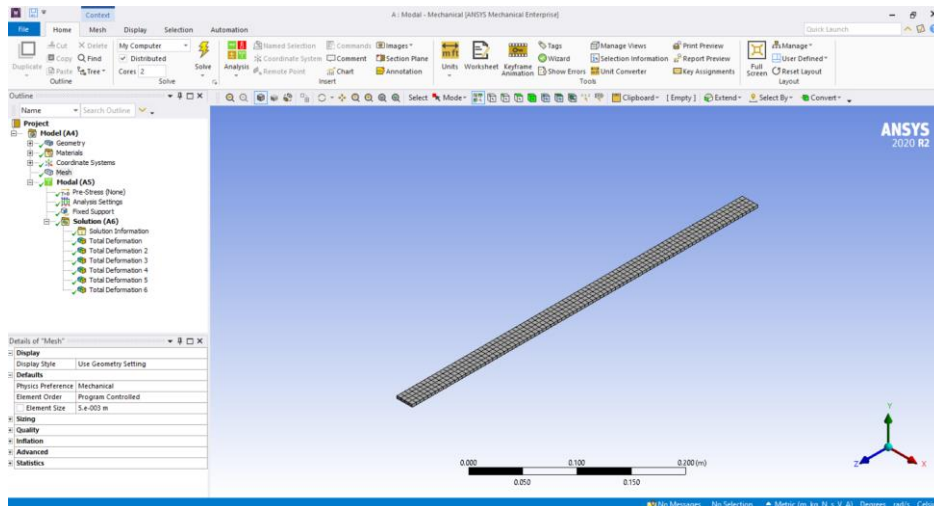
## 6 Einführung

Der Struktur werden dann Materialeigenschaften zugewiesen. Dies ist in Abbildung 1.8 unten dargestellt.

Outline of Schematic A2: Engineering Data				
	A	B	C	D
1	Contents of Engineering Data			Description
2	Material			
3	Structural Steel			Fatigue Data at zero mean stress comes from 1998 ASME BPV Code, Section 8, Div 2, Table S-110.1
Click here to add a new material				
Properties of Outline Row 3: Structural Steel				
	Property	Value	Unit	
1	Material Field Variables	Table		
3	Density	7850	kg m <sup>-3</sup>	
4	Isotropic Secant Coefficient of Thermal Expansion			
6	Isotropic Elasticity			
7	Derive from	Young's Modulus and Poisson...		
8	Young's Modulus	2.1E+11	Pa	
9	Poisson's Ratio	0.3		
10	Bulk Modulus	1.75E+11	Pa	
11	Shear Modulus	8.0769E+10	Pa	

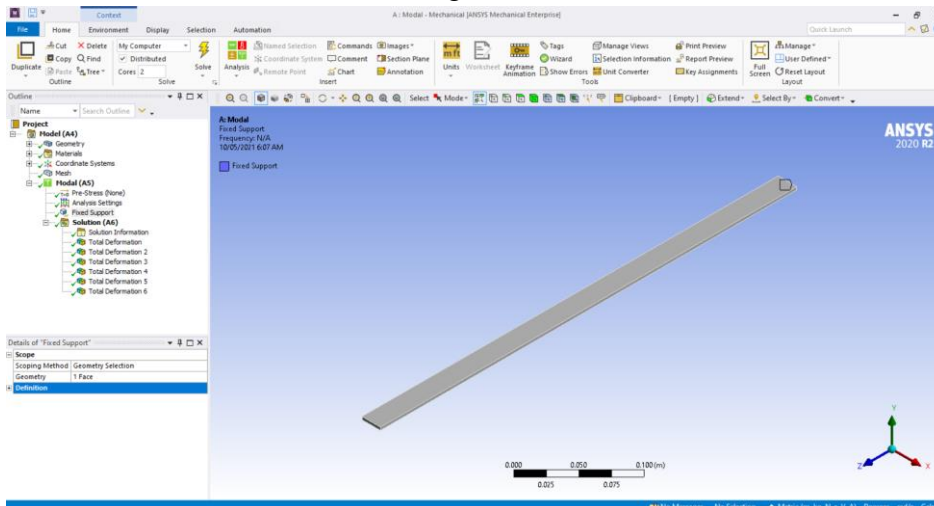
**Abbildung 1.8: Materialeigenschaften des Modells**

Nächster Schritt ist die Erstellung des Netzes auf der Struktur. Unter Vernetzung versteht man die Zerlegung der Geometrie in leicht beschreibbare Teilbereiche der Elemente. Die einzelnen Elemente werden an den Eckknoten und, falls vorhanden, an den Mittelknoten miteinander verbunden. Mit feiner Netzgröße kann eine höhere Genauigkeit auf Kosten einer höheren Berechnungszeit erreicht werden.



**Abbildung 1.9: Netz des Balkenelements**

Die Randbedingungen spielen in der FEM eine wichtige Rolle. Die Änderung der Randbedingung führt zu einer Änderung des Ergebnisses. Hier wird das Balkenelement an einem Ende festgehalten, während das andere frei schwingen kann.



**Abbildung 1.10: Modell nach Anwendung der Randbedingung**

Nach der Berechnung wird mit einem Klick auf LÖSUNG die Liste der Eigenfrequenzen in tabellarischer Form und als Grafik angezeigt (siehe Abbildung 1.11). Alle auftretenden Eigen-schwingungen (Eigenmoden) sind zu sehen und können animiert werden.

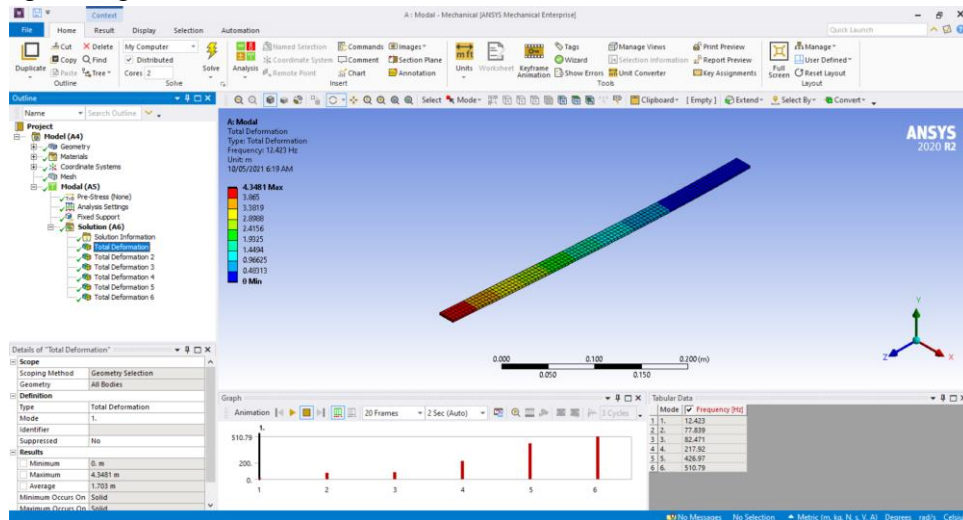


Abbildung 1.11: Verformung des Balkenelements

Die resultierenden Eigenfrequenzen sind in der folgenden Tabelle 1.1 dargestellt.

Tabelle 1.1: Eigenfrequenzen des freitragenden Trägers

Mode	Eigenfrequenz (Hz)
Erste Eigenmode	12,423
Zweite Eigenmode	77,839
Dritte Eigenmode	82,471
Vierte Eigenmode	217,92
Fünfte Eigenmode	426,97
Sechste Eigenmode	510,79

Analytisch kann die Eigenfrequenz mit einer aus der Euler-Bernoulli-Gleichung abgeleiteten Formel berechnet werden (Mekalke, Sutar. 2016)

Erste Eigenfrequenz:

$$\omega_1 = 1,875^2 \sqrt{\frac{Eh^2}{12\rho L^4}} = 1,875^2 \sqrt{\frac{2,1 \times 10^{11} \cdot 0,003^2}{12 \cdot 7850 \cdot 0,45^4}} = 77,76 \text{ rad/sec}$$

$$\omega_1 = 12,37 \text{ Hz}$$

Zweite Eigenfrequenz:

$$\omega_2 = 4,694^2 \sqrt{\frac{Eh^2}{12\rho L^4}} = 4,694^2 \sqrt{\frac{2,1 \times 10^{11} \cdot 0,003^2}{12 \cdot 7850 \cdot 0,45^4}} = 487,34 \text{ rad/sec}$$

$$\omega_2 = 77,53 \text{ Hz}$$

Die analytischen Ergebnisse stimmen also mit der mit ANSYS Workbench 2020 R2 durchgeführten Berechnung überein.

Jede Eigenfrequenz hat einen entsprechenden Eigenvektor, der die Modenformen darstellt. Die entstehenden Modenformen sind im Folgenden dargestellt:

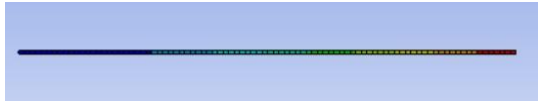


Abbildung 1.7: Erste Eigenmode



Abbildung 1.8: Zweite Eigenmode



Abbildung 1.9: Dritte Eigenmode

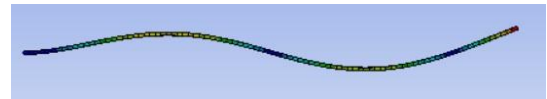


Abbildung 1.10: Vierte Eigenmode



Abbildung 1.11: Fünfte Eigenmode

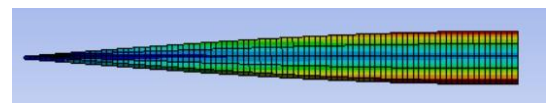


Abbildung 1.12: Sechste Eigenmode

Hier wurden nur sechs anfängliche Eigenfrequenzen berechnet. Es könnten jedoch auch höhere Eigenfrequenzen berechnet werden, aber dafür sollte das Netz fein genug sein, um korrekt zu lösen. Infolgedessen wird mehr Rechenzeit benötigt.

## 2 Berechnung von Eigenwerten und Eigenvektoren

Eigenwerte und Eigenvektoren können sowohl analytisch als auch mit Hilfe numerischer Methoden berechnet werden. Die analytische Methode ist sehr begrenzt und für komplexe Probleme werden numerische Methoden verwendet. Im Folgenden werden beide Methoden im Detail behandelt.

### 2.1 Analytisch

Sei  $A$  eine  $n \times n$ -Matrix. Sei  $v$  ein Eigenvektor von  $A$ , der zu dem Eigenwert  $\lambda$  gehört. Dann

$$A \cdot v = \lambda \cdot v$$

$$A \cdot v = (\lambda I) \cdot v$$

$$(A - \lambda I)v = 0 \tag{2.1}$$

Da  $v = 0$  das Gleichungssystem immer löst, wird eine Lösung  $v \neq 0$  gesucht. Damit eine solche weitere Lösung existiert, darf das Gleichungssystem  $A - \lambda I$  nicht eindeutig lösbar sein. Dies ist gleichbedeutend mit der Aussage, dass die Determinante von  $A - \lambda I$  den Wert 0 hat. Um eine Lösung des Problems zu berechnen müssen zunächst die Werte für  $\lambda$  (Eigenwerte) bestimmt werden, für welche die Determinante 0 ist. Dies geschieht über das charakteristische Polynom.

$|A - \lambda I| = 0$  wird als charakteristisches Polynom der Matrix  $A$  bezeichnet (Beutelspacher, S. 249). Die Eigenwerte  $\lambda$  von  $A$  sind Nullstellen des charakteristischen Polynoms. Zugehörige Eigenvektoren von  $A$  sind Nicht-Null Lösungen der Gleichung  $(A - \lambda I)v = 0$  (Beutelspacher, S. 251).

### Beispiel

$$\begin{aligned}
 A &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \\
 |A - \lambda I| &= \begin{vmatrix} a - \lambda & b \\ c & d - \lambda \end{vmatrix} \\
 |A - \lambda I| &= (a - \lambda)(d - \lambda) - bc \\
 &= \lambda^2 - (a + b)\lambda + (ad - bc)
 \end{aligned} \tag{2.2}$$

dies wird als charakteristische Gleichung der Matrix bezeichnet.

Analytisch kann man Eigenvektor und Eigenwerte mit folgendem Beispiel berechnen.

Gegeben sei die Matrix

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

Die charakteristische Gleichung kann geschrieben werden als

$$\begin{aligned}
 |A - \lambda I| &= \left| \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0 \\
 \left| \begin{bmatrix} -\lambda & 1 \\ -2 & -3 - \lambda \end{bmatrix} \right| &= \lambda^2 + 3\lambda + 2 = 0
 \end{aligned}$$

die beiden Eigenwerte sind (Anwendung der PQ-Formel zur Bestimmung der Eigenwerte).

$$\begin{aligned}
 \lambda &= \frac{-3 \pm \sqrt{(3)^2 - 4(1)(2)}}{2(1)} \\
 \lambda_1 &= -1, \quad \lambda_2 = -2
 \end{aligned}$$

Jetzt können die zu den Eigenwerten zugehörigen Eigenvektoren berechnet werden, indem der Kern der Matrix  $(A - \lambda I)v$  bestimmt wird. Dies soll zunächst für  $\lambda_1 = -1$  geschehen.

$$A \cdot v_1 = \lambda_1 \cdot v_1$$

$$(A - \lambda_1) v_1 = 0$$

$$\begin{bmatrix} -\lambda_1 & 1 \\ -2 & -3 - \lambda_1 \end{bmatrix} \cdot v_1 = 0$$

$$\begin{bmatrix} 1 & 1 \\ -2 & -2 \end{bmatrix} \cdot v_1 = \begin{bmatrix} 1 & 1 \\ -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$$

Aus der ersten Zeile erhält man die Gleichung.

$$v_{1,1} + v_{1,2} = 0$$

$$v_{1,1} = -v_{1,2}$$

wenn die zweite Zeile ausgewählt wird, wäre das Ergebnis dasselbe.

$$-2 \cdot v_{1,1} - 2 \cdot v_{1,2} = 0$$

$$v_{1,1} = -v_{1,2}$$

In beiden Fällen wird herausgefunden, dass der erste Eigenvektor ein beliebiger 2-Element Spaltenvektor ist, in dem die beiden Elemente gleich groß sind und entgegengesetzte Vorzeichen haben. Der Eigenvektor kann geschrieben werden als

$$\mathbf{v}_1 = k_1 \begin{bmatrix} +1 \\ -1 \end{bmatrix}$$

wobei  $k_1$  eine willkürliche Konstante ist.

Dasselbe Verfahren kann für den zweiten Eigenvektor wiederholt werden.

Wenn  $\lambda_2 = -2$ ,

$$\mathbf{A} \cdot \mathbf{v}_2 = \lambda_2 \cdot \mathbf{v}_2$$

$$(\mathbf{A} - \lambda_2) \mathbf{v}_2 = 0$$

$$\begin{bmatrix} -\lambda_2 & 1 \\ -2 & -3 - \lambda_2 \end{bmatrix} \cdot \mathbf{v}_2 = 0$$

$$\begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix} \cdot \mathbf{v}_2 = \begin{bmatrix} 2 & 1 \\ -2 & -1 \end{bmatrix} \cdot \begin{bmatrix} v_{2,1} \\ v_{2,2} \end{bmatrix} = 0$$

$$2 \cdot v_{2,1} + 1 \cdot v_{2,2} = 0$$

$$2 \cdot v_{2,1} = -v_{2,2}$$

$$\mathbf{v}_2 = k_2 \begin{bmatrix} +1 \\ -2 \end{bmatrix}$$

Man kann Eigenvektoren nur bis zu einer Matrixgröße von  $4 \times 4$  auf diese analytische Weise berechnen, da für eine  $5 \times 5$  Matrix das charakteristische Polynom Grad 5 hat und die Nullstellen (Eigenwerte) eines Polynoms von Grad 5 im Allgemeinen nicht algebraisch lösbar sind.

## 2.2 Numerische Eigenwert-Algorithmen

Im Folgenden werden verschiedene numerische Methoden zur Berechnung von Eigenwerten und Eigenvektoren ausführlich besprochen.

### 2.2.1 Power-Iteration

Die Power-Iteration ist die grundlegendste Methode zur (numerischen) Berechnung eines Eigenvektors. Bei dieser Methode wird angenommen, dass die Matrix  $A$  einen dominanten Eigenwert (einen Eigenwert mit einem größten absoluten Wert) mit einem entsprechenden dominanten Eigenvektor besitzt. Es erfolgt eine erste Approximation  $v$  eines der Eigenvektoren von  $A$ . Diese erste Approximation ist ein zufälliger Vektor und ein Nicht-Null-Vektor im  $\mathbb{R}^n$ . Der Vektor wird normalisiert. Schließlich wendet man die folgende Sequenz an.

$$\begin{aligned}
v_1 &= Av_0, & v_1 &= \frac{v_1}{\|v_1\|} \\
v_2 &= Av_1 = A(Av_0) = A^2v_0, & v_2 &= \frac{v_2}{\|v_2\|} \\
v_3 &= Av_2 = A(A^2v_0) = A^3v_0, & v_3 &= \frac{v_3}{\|v_3\|} \\
&\vdots \\
v_k &= Av_{k-1} = A(A^{k-1}v_0) = A^kv_0, & v_k &= \frac{v_k}{\|v_k\|}
\end{aligned}$$

Das vollständige Verfahren wird im folgenden Beispiel veranschaulicht.

### 2.2.1.1 Beispiel

Die Methode dieses Beispiels ist entnommen aus (Chapra 2014, S.797). In diesem Beispiel wird ein dominanter Eigenvektor der gegebenen Matrix approximiert. Zu diesem Zweck wird im Folgenden eine Power-Iteration durchgeführt.

$$A = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 5 & 1 \\ -1 & 2 & 3 \end{bmatrix}$$

Das Verfahren beginnt mit der Auswahl eines anfänglichen Nicht-Null-Vektors  $v_0$ .

$$v_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Anschließend werden Iterationen durchgeführt, um die folgende Annäherung zu erhalten.

Iteration	Normalisierung
$v_1 = Av_0 = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 5 & 1 \\ -1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 4 \end{bmatrix}$	$\blacktriangleright v_1 = \frac{v_1}{\ v_1\ _2} = \begin{bmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{bmatrix}$
$v_2 = Av_1 = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 5 & 1 \\ -1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{bmatrix} = \begin{bmatrix} 2.8557 \\ 5.3072 \\ 2.4495 \end{bmatrix}$	$\blacktriangleright v_2 = \frac{v_2}{\ v_2\ _2} = \begin{bmatrix} 0.4392 \\ 0.8157 \\ 0.3765 \end{bmatrix}$
$v_3 = Av_2 = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 5 & 1 \\ -1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0.4392 \\ 0.8157 \\ 0.3765 \end{bmatrix} = \begin{bmatrix} 2.8863 \\ 5.3334 \\ 2.3216 \end{bmatrix}$	$\blacktriangleright v_3 = \frac{v_3}{\ v_3\ _2} = \begin{bmatrix} 0.4445 \\ 0.8213 \\ 0.3575 \end{bmatrix}$
$v_4 = Av_3 = \begin{bmatrix} 1 & 3 & 0 \\ 2 & 5 & 1 \\ -1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0.4445 \\ 0.8213 \\ 0.3575 \end{bmatrix} = \begin{bmatrix} 2.9085 \\ 5.3532 \\ 2.2708 \end{bmatrix}$	$\blacktriangleright v_4 = \frac{v_4}{\ v_4\ _2} = \begin{bmatrix} 0.4473 \\ 0.8233 \\ 0.3493 \end{bmatrix}$

$$\lambda_1 = \frac{(Av_4) \cdot v_4}{v_4 \cdot v_4} = 6.5036$$

Es ist zu erkennen, dass der Approximationsvektor sich  $\begin{bmatrix} 0.4473 \\ 0.8233 \\ 0.3493 \end{bmatrix}$  annähert. Dies ist der dominante Eigenvektor von Matrix  $A$  und der entsprechende Eigenwert ist 6.5036.

Der zweitgrößte Eigenwert kann herausgefunden werden, indem derselbe Algorithmus auf die

Matrix



$$A := A - \lambda_1 \cdot v_1 \cdot v_1^T$$

angewendet wird.

Dieses Verfahren ist relativ einfach, jedoch auch sehr ineffizient. Die Konvergenz ist dann besonders langsam, wenn zwei Eigenwerte denselben Absolutbetrag haben.

### 2.2.2 QR Algorithmus

Der QR-Algorithmus wurde in den späten 1950er Jahren von John G. F. Francis und Vera N. Kublanowskaja unabhängig voneinander entwickelt. Es ist eine beliebte numerische Methode zur Berechnung aller Eigenwerte und schließlich aller Eigenvektoren einer quadratischen Matrix. Eine QR-Zerlegung von einer reellen quadratischen Matrix  $A$  ist eine Zerlegung von  $A$  als

$$A = QR \quad (2.3)$$

wobei  $Q$  eine orthogonale Matrix (d.h.  $Q^T Q = I$ ) und  $R$  eine obere Dreiecksmatrix ist.

Der QR-Algorithmus kann als eine weiterentwickelte Variante der grundlegenden Power-Iterationsmethode angesehen werden. Der Power-Algorithmus multipliziert die Matrix mit einem einzelnen Vektor und normalisiert nach jeder Iteration. Der Vektor konvergiert zu einem Eigenvektor mit dem größten Eigenwert. Stattdessen arbeitet der QR-Algorithmus mit einer vollständigen Basis von Vektoren und verwendet die QR-Zerlegung zur Renormierung und Orthogonalisierung.

Es gibt verschiedene QR-Methoden mit mehreren Änderungen, um Zeiteffizienz zu erreichen. Diese Methoden werden nacheinander in den kommenden Abschnitten behandelt. Zunächst muss jedoch geklärt werden, wie man eine QR-Zerlegung erhält.

#### 2.2.2.1 QR-Zerlegung mittel Gram Schmidt

Das Gram Schmidt Verfahren wird verwendet, um eine nicht-orthogonalen Basis in eine orthogonale Basis zu überführen. Dies wird Schritt für Schritt durchgeführt. Die Gleichungen in diesem Abschnitt sind zitiert aus (Ford 2015, S.284).

Hierzu werden die Spalten der Matrix  $A$  als Vektoren betrachtet, d.h.

$$A = [a_1 \mid a_2 \mid \dots \mid a_n]$$

Anschließend wird von jedem Vektor der Anteil der vorherigen Vektoren abgezogen und der Vektor normiert.

$$u_1 = a_1, \quad e_1 = \frac{u_1}{\|u_1\|}$$

$$u_2 = a_2 - (a_2 \cdot e_1)e_1, \quad e_2 = \frac{u_2}{\|u_2\|}$$

$$u_{k+1} = a_{k+1} - (a_{k+1} \cdot e_1)e_1 - \dots - (a_{k+1} \cdot e_k)e_k, \quad e_{k+1} = \frac{u_{k+1}}{\|u_{k+1}\|}$$

Die resultierende QR-Faktorisierung kann geschrieben werden als

$$A = [a_1 \mid a_2 \mid \dots \mid a_n] = [e_1 \mid e_2 \mid \dots \mid e_n] \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \dots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \dots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_n \cdot e_n \end{bmatrix} = QR$$

**Beispiel**

Die Methode dieses Beispiels ist entnommen aus (Ford 2015, S.284)

Gegeben sei die Matrix

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

mit den Vektoren  $a_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ ,  $a_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ ,  $a_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$ .

Bei der Durchführung des Gram Schmidt Verfahrens erhält man:

$$u_1 = a_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},$$

$$e_1 = \frac{u_1}{||u_1||} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix},$$

$$u_2 = a_2 - (a_2 \cdot e_1)e_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} - \frac{1}{\sqrt{2}} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ -1/2 \\ 1 \end{bmatrix},$$

$$e_2 = \frac{u_2}{||u_2||} = \frac{1}{\sqrt{3/2}} \begin{bmatrix} 1/2 \\ -1/2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{6} \\ -1/\sqrt{6} \\ 2/\sqrt{6} \end{bmatrix},$$

$$u_3 = a_3 - (a_3 \cdot e_1)e_1 - (a_3 \cdot e_2)e_2$$

$$u_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} - \frac{1}{\sqrt{2}} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix} - \frac{1}{\sqrt{6}} \begin{bmatrix} 1/\sqrt{6} \\ -1/\sqrt{6} \\ 2/\sqrt{6} \end{bmatrix} = \begin{bmatrix} -1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix},$$

$$e_3 = \frac{u_3}{||u_3||} = \begin{bmatrix} -1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix},$$

So,

$$Q = [e_1 | e_2 | \dots | e_n] = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{bmatrix},$$

$$R = \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \cdots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{bmatrix}.$$

### 2.2.2.2 Basic QR-Algorithmus

Die Gleichungen und der Algorithmus 1 in diesem Abschnitt sind zitiert aus (Arbenz 2016, S.63). Der grundlegende QR-Algorithmus lautet wie folgt: Gegeben sei eine Matrix  $A$ , deren Eigenwerte bestimmt werden sollen. Sei  $A_0 = A$ , dann bestimme iterativ  $A_k$  aus  $A_{k-1}$ , indem die QR-Zerlegung  $Q_k$  und  $R_k$  von  $A_{k-1}$  bestimmt wird, so dass  $A_{k-1} = Q_k R_k$  gilt. Definiere anschließend  $A_k$  als  $A_k = R_k Q_k$ . Der grundlegende Algorithmus der QR-Zerlegung kann geschrieben werden als

$$A_{k-1} = Q_k R_k, \quad A_k = R_k Q_k = Q_k^* A_{k-1} Q_k,$$

#### Algorithmus 1: Basic QR Algorithmus

1. Sei  $A \in \mathbb{C}^{n \times n}$ . Dieser Algorithmus berechnet eine obere Dreiecksmatrix  $T$  und eine einheitliche Matrix  $U$ , so dass  $A = UTU^*$ .
2. Setze  $A_0 := A$  und  $U_0 = I$ .
3. **for**  $k = 1, 2, \dots$  **do**
4.    $A_{k-1} := Q_k R_k$ ; /\* QR Zerlegung \*/
5.    $A_k := R_k Q_k$ ;
6.    $U_k := U_{k-1} Q_k$ ; /\* Transformationsmatrix aktualisieren \*/
7. **end for**
8. Setzte  $T := A_\infty$  and  $U := U_\infty$

Der Algorithmus kann geschrieben werden als:

$$A_k = Q_k^* A_{k-1} Q_k = Q_k^* Q_{k-1}^* A_{k-2} Q_{k-1} Q_k = \cdots = Q_k^* \cdots Q_1^* A_0 \underbrace{Q_1 \cdots Q_k}_{U_k}.$$

wobei  $A_k$  zu einer oberen Dreiecksmatrix konvergiert. Die Diagonaleinträge der Matrix  $A_k$  entsprechen den Eigenwerten und die Matrix  $U_k$  enthält die dazugehörigen Eigenvektoren.

### 2.2.2.3 Laufzeitanalyse

Die Konvergenz des Basic QR Algorithmus ist sehr langsam. Wenn die Eigenwerte ähnlich groß sind, ist sie besonders langsam. Jeder Iterationsschritt im Algorithmus erfordert die Berechnung der QR Zerlegung einer vollen  $n \times n$  Matrix, d.h. jeder einzelne Iterationsschritt hat eine Komplexität  $\mathcal{O}(n^3)$  und es werden viele Iterationen benötigt (Arbenz 2016, S.67).

In den kommenden Abschnitten werden deshalb Algorithmen mit verbesserter Laufzeit implementiert.

### 2.2.3 Beschleunigter QR-Algorithmus

Die Konvergenz des oben beschriebenen Algorithmus ist langsam. Im Folgenden soll die Konvergenz des Algorithmus verbessert werden. Es ist eine gute Idee, eine Matrixstruktur zu finden, die durch den QR-Algorithmus erhalten bleibt und die die Kosten für einen einzelnen Iterationsschritt senkt.

#### 2.2.3.1 Hessenberg-Form

Eine Matrixstruktur, die der oberen Dreiecksform nahek kommt und die durch den QR-Algorithmus erhalten bleibt, ist die Hessenberg-Form. Ziel ist es, die Matrix A, deren Eigenwerte bestimmt werden sollen, durch Ähnlichkeitstransformationen (Ähnlichkeitstransformationen verändern die Eigenwerte nicht) auf Hessenberg-Form zu bringen.

Die folgende Diskussion und Algorithmus 2 ist hauptsächlich aus (Arbenz 2016, S.71-73) entnommen.

**Definition:** Eine Matrix H ist eine Hessenberg-Matrix, wenn deren Einträge unterhalb der ersten Nebendiagonalen gleich Null sind,

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1n} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2n} \\ 0 & h_{32} & h_{33} & \cdots & h_{3n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{nn-1} & h_{nn} \end{bmatrix} \quad h_{ij} = 0, \quad i < j + 1$$

#### Die Householder-Reduktion auf Hessenberg-Form

Durch Householder-Reflexionen lässt sich jede Matrix A auf Hessenberg-Form transformieren. Eine Matrix der Form

$$P = I - 2\mathbf{u}\mathbf{u}^* / \mathbf{u}^*\mathbf{u}, \quad \|\mathbf{u}\| = 1,$$

wird als Householder-Reflektor bezeichnet. Wobei,

$$\mathbf{u} = \mathbf{x} + \operatorname{sgn}(x_1) \cdot \|\mathbf{x}\|_2 \cdot \mathbf{e}$$

Hier ist  $\mathbf{x}$  der Spaltenvektor der Matrix A, dessen Einträge auf Null gesetzt werden. Ziel ist es, Spalte für Spalte die Nulleinträge zu erzeugen. Im Falle des ersten Vektors  $\mathbf{u}_1$  ist  $\mathbf{x}$  der Spaltenvektor unter dem ersten Element in der ursprünglichen Matrix:  $\mathbf{x} = [a_{21} \ a_{31} \ \dots \ a_{n1}]^T$  und  $\mathbf{e}$  ist auf die Formel  $\mathbf{e} = [1 \ 0 \ \dots \ 0]^T$ , die Größe von  $\mathbf{e}$  ist gleich wie die Größe von  $\mathbf{x}$ .

Nun wird gezeigt, wie eine quadratische Matrix mit Householder-Reflektoren auf die Hessenberg-Form reduziert werden kann. Im ersten Schritt werden Nullen in der ersten Spalte unterhalb des zweiten Elements eingeführt.

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1^*} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{*P_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = P_1^* A P_1$$

Es gilt  $P_1 = P_1^*$ , da es sich um einen Householder-Reflektor handelt, welcher eine orthogonale Matrix ist.  $P_1$  hat die Struktur

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = \begin{bmatrix} 1 & 0^T \\ 0 & I_4 - 2\mathbf{u}\mathbf{u}^*/\mathbf{u}^*\mathbf{u} \end{bmatrix}$$

Die Multiplikation von  $P_1$  von links ergibt das gewünschte Nullen in Spalte 1 von  $A$ . Um die Ähnlichkeit zu erhalten, ist die Multiplikation von rechts erforderlich. Wegen der Nicht-Null-Struktur von  $P_1$  ist die erste Spalte von  $P_1 A$  nicht betroffen. Daher bleibt die Nullen erhalten.

Auf analoge Weise, werden anschließend die restlichen Einträge auf 0 transformiert:

$$P_1 A P_1 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_2^*/*P_2} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix} \xrightarrow{P_3^*/*P_3} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix} = P_3 P_2 P_1 A P_1 P_2 P_3$$

Der Algorithmus lässt sich in Pseudocode wie folgt schreiben:

### Algorithmus 2: Reduktion auf Hessenberg-Form

1. Dieser Algorithmus reduziert eine Matrix  $A \in \mathbb{C}^{n \times n}$  zu Hessenberg Form  $H$  durch eine Sequenz von Householder-Reflektoren.  $H$  überschreibt  $A$ .
2. **for**  $k = 1$  to  $n - 2$  **do**
3.   Erzeugung eines Householder-Reflektors  $P_k$ ;
4.   /\* Anwenden  $P_k = I_k \oplus (I_{n-k} - 2\mathbf{u}_k\mathbf{u}_k^*)$  von links nach  $A$  \*/
5.    $A_{k+1:n,k:n} := A_{k+1:n,k:n} - 2\mathbf{u}_k(\mathbf{u}_k^* A_{k+1:n,k:n})$ ;
6.   /\* Anwenden  $P_k$  von rechts,  $A := A P_k$  \*/
7.    $A_{1:n,k+1:n} := A_{1:n,k+1:n} - 2(A_{1:n,k+1:n} \mathbf{u}_k) \mathbf{u}_k^*$ ;
8. **end for**
9. **Wenn** Eigenvektoren gewünscht werden von  $U = P_1 \cdots P_{n-2}$  **dann**
10.    $U := I_n$ ;
11. **for**  $k = n - 2$  down to 1 **do**
12.   /\* Update  $U := P_k U$  \*/
13.    $U_{k+1:n,k+1:n} := U_{k+1:n,k+1:n} - 2\mathbf{u}_k(\mathbf{u}_k^* U_{k+1:n,k+1:n})$ ;
14. **end for**
15. **end if**

### 2.2.3.2 Hessenberg-QR-Algorithmus

Die folgende Diskussion und Algorithmus 3 ist hauptsächlich aus (Arbenz 2016, S.67-70) entnommen.

Im vorherigen Abschnitt wird besprochen, wie man eine Hessenberg-Form mittels einer Householder Reduktion erhält. Die Hessenberg-Form hilft, die Kosten für einen QR-Schritt von  $\mathcal{O}(n^3)$  auf  $\mathcal{O}(n^2)$  zu senken, wodurch die Konvergenzrate des QR-Algorithmus verbessert wird. Dieser Abschnitt ist dem Hessenberg-QR-Algorithmus gewidmet.

Ausgehend von einem konstruktiven Beweis, dass die Hessenberg-Matrix  $H$  durch QR-Zerlegung, d.h.  $H = QR$ , zerlegt werden kann. Es wird gezeigt, dass  $\hat{H} = RQ$  wieder eine Hessenberg-Matrix ist.

Zur effizienten Berechnung von  $\hat{H} = RQ$  aus  $H$  werden Givens-Rotationen verwendet

Die Givens-Rotation  $G(i, j, \vartheta)$  kann definiert werden als

$$G(i, j, \vartheta) := \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \leftarrow i \\ \leftarrow j \end{matrix}$$

$\uparrow \qquad \qquad \uparrow$   
 $i \qquad \qquad j$

Wobei  $c = \cos(\vartheta)$  und  $s = \sin(\vartheta)$ . Die Multiplikation durch  $G(i, j, \vartheta)$  ergibt eine Drehung im Gegenuhrzeigersinn um  $\vartheta$  in der  $(i, j)$ -Koordinatenebene. Falls  $x \in \mathbb{R}^n$  und  $y = G(i, j, \vartheta)^* x$ , dann

$$y_k = \begin{cases} cx_i - sx_j & k = i \\ sx_i + cx_j & k = j \\ x_k & k \neq i, j \end{cases}$$

$y_j$  wird auf Null gesetzt, indem

$$c = \frac{x_i}{\sqrt{|x_i|^2 + |x_j|^2}}, \quad s = \frac{-x_j}{\sqrt{|x_i|^2 + |x_j|^2}}.$$

Nun soll das prinzipielle Vorgehen anhand eines  $4 \times 4$  Beispiels veranschaulicht werden.

$$H = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(1,2,\vartheta_1)^*} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(2,3,\vartheta_2)^*} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(3,4,\vartheta_3)^*} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} = R$$

Mit  $G_k = G(k, k+1, \vartheta_k)$ , erhält man

$$\underbrace{G_3^* G_2^* G_1^*}_{Q^*} H = R \quad \Leftrightarrow \quad H = QR.$$

Multiplikation von  $Q$  und  $R$  in umgekehrter Reihenfolge ergibt

$$\hat{H} = RQ = RG_1 G_2 G_3,$$

oder, detaillierte Ansicht,

$$R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{.G(1,2,\vartheta_1)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{.G(2,3,\vartheta_2)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{.G(3,4,\vartheta_3)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} = H$$

Im Allgemeinen gilt, dass, wenn  $n \times n$  die Größe von  $H$  ist,  $n - 1$  Givens Rotationen  $G_1, \dots, G_{n-1}$  benötigt, um  $H$  in die obere Dreiecksform zu transformieren. Die Anwendung der Rotationen von rechts stellt die Hessenberg-Form wieder her.

### Algorithmus 3: Ein Hessenberg-QR-Schritt

1.  $H \in \mathbb{C}^{n \times n}$  sei eine obere Hessenberg-Matrix. Dieser Algorithmus überschreibt  $H$  mit  $\hat{H} = RQ$ , wobei  $H = QR$  eine QR-Zerlegung von  $H$  ist.
2. **for**  $k = 1, 2, \dots, n - 1$  **do**
3.   /\* Erzeugen von  $G_k$  und dann anwenden:  $H = G(k, k + 1, \vartheta_k)^* H$  \*/
4.    $[c_k, s_k] := \text{givens}(H_{k,k}, H_{k+1,k});$
5.    $H_{k:k+1,k:n} = \begin{bmatrix} c_k & -s_k \\ s_k & c_k \end{bmatrix} H_{k:k+1,k:n};$
6. **end for**
7. **for**  $k = 1, 2, \dots, n - 1$  **do**
8.   /\* Wenden Sie die Rotationen  $G_k$  von rechts an \*/
9.    $H_{1:k+1,k:k+1} = H_{1:k+1,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix};$
10. **end for**
11. **for**  $k = 1, 2, \dots, n - 1$  **do**
12.    $U_{1:n,k:k+1} = U_{1:n,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix};$
13. **end for**

#### 2.2.3.3 Laufzeitanalyse

Zunächst wird die Reduktion der Vollmatrix auf die Hessenberg-Form betrachtet. In diesem Verfahren werden die Householder-Vektoren an den Stellen der Nullen gespeichert. Daher wird die Matrix  $U = P_1 \cdots P_{n-2}$  berechnet, nachdem alle Householder-Vektoren erzeugt wurden, wodurch  $\binom{2}{3}n^3$ -Flops eingespart werden. Die Gesamtkomplexität der Reduktion ist

- Anwendung des  $P_k$  von links:  $\sum_{k=1}^{n-2} 4(n-k-1)(n-k) \approx \frac{4}{3}n^3$
- Anwendung des  $P_k$  von rechts:  $\sum_{k=1}^{n-2} 4(n)(n-k) \approx 2n^3$
- Von  $U = P_1 \cdots P_{n-2}$ :  $\sum_{k=1}^{n-2} 4(n-k)(n-k) \approx \frac{4}{3}n^3$

So kostet die Reduktion auf Hessenberg-Form  $\frac{10}{3}n^3$ -Flops ohne Bildung der Transformationsmatrix und  $\frac{14}{3}n^3$  mit der Bildung der Transformationsmatrix.

Nun wird die Laufzeitanalyse von Hessenberg-QR-Schritt diskutiert. Betrachtet man den Algorithmus 2, so ist bei Vernachlässigung der Bestimmung von  $c_k$  und  $s_k$ , jede der beiden Schleifen braucht

$$\sum_{i=1}^{n-1} 6i = \frac{n(n-1)}{2} \approx 3n^2\text{-Flops}$$

Optional, wenn die Operation  $U_k := U_{k-1}Q_k$  des Algorithmus 2 ausgeführt wird, kostet es

$$\sum_{i=1}^{n-1} 6n \approx 6n^2\text{-Flops}$$

Insgesamt erfordert ein  $QR$ -Schritt mit einer Hessenberg-Matrix  $12n^2$  Gleitkommaoperationen, wozu auch die Aktualisierung der einheitlichen Transformationsmatrix gehört. Dies muss in Bezug zu einem  $QR$ -Schritt mit einer Vollmatrix gesetzt werden, die  $\frac{7}{3}n^3$  kostet.

Folglich wird ein Faktor von  $\mathcal{O}(n)$  in Bezug auf die Operationen beim Übergang von der Dichten zur Hessenberg-Form gewonnen. Es kann jedoch immer noch eine sehr langsame Konvergenz bestehen, wenn einer der Quotienten  $|\lambda_k|/|\lambda_{k+1}|$  nahe bei 1 liegt.

Die Konvergenz des Hessenberg- $QR$ -Schritts wird sich durch die Einführung der Rayleigh-Quotientenverschiebung dramatisch verbessern. Die oben diskutierte Laufzeitanalyse ist hauptsächlich entnommen aus (Arbenz 2016, S.72)

### 2.2.3.4 Hessenberg- $QR$ -Algorithmus mit Shift

Die folgende Diskussion und Algorithmus 4 ist hauptsächlich aus (Arbenz 2016, S.74-77) entnommen.

Bislang wurde diskutiert, dass die Umwandlung einer Matrix in die Hessenberg-Form günstiger ist. In diesem Abschnitt wird diskutiert, wie die Konvergenz des  $QR$ -Algorithmus verbessert werden kann, indem **(spektrale) Verschiebungen** in den Algorithmus eingeführt werden. Schon beim normalen  $QR$ -Algorithmus wurde beobachtet, dass die Konvergenz sehr langsam ist, wenn das Verhältnis von zwei Eigenwerten  $\frac{|\lambda_1|}{|\lambda_2|} \approx 1$  ist (Arbenz 2016, S.70). Durch die Transformation  $H - \lambda I$  lassen sich die Eigenwerte um  $\lambda$  verschieben, wodurch bei geschickter Wahl von  $\lambda$  erreicht werden kann, dass  $\frac{|\lambda_1 - \lambda|}{|\lambda_2 - \lambda|} \neq 1$  gilt, wodurch die Konvergenz beschleunigt wird.

Die Idee ist es:

1. Die Matrix in Hessenberg-Form zu transformieren.
2. Die perfekte Verschiebung Hessenberg- $QR$  mit den Eigenwerten, von denen man wusste, dass sie nacheinander existieren, durchzuführen.

Sei  $\lambda$  ein Eigenwert der irreduziblen Hessenberg-Matrix  $H$ . Es wird geprüft, was geschieht, wenn folgende Operationen durchgeführt werden

---

1:  $H - \lambda I$  /\*  $QR$ -Zerlegung \*/

2:  $\hat{H} = RQ + \lambda I$

---

Zuerst wird festgestellt, dass  $\hat{H} \sim H$  und zwar

$$\hat{H} = Q^*(H - \lambda I)Q + \lambda I = Q^*H Q$$

Zweitens,

$$H - \lambda I = QR, \quad \text{mit} \quad R = \begin{bmatrix} \times & & \\ & \times & \\ & & \times \\ & & & 0 \end{bmatrix}.$$



Also,

$$RQ = \begin{bmatrix} \diagup & & \\ & \diagup & \\ & & 0 & 0 \end{bmatrix}$$

und

$$\hat{H} = RQ + \lambda I = \begin{bmatrix} \diagup & & \\ & \diagup & \\ & & 0 & \lambda \end{bmatrix} = \begin{bmatrix} \hat{H}_1 & h_1 \\ 0^T & \lambda \end{bmatrix}$$

Wenn also ein  $QR$ -Schritt mit einer perfekten Verschiebung auf die Hessenberg-Matrix angewendet wird, fällt der Eigenwert aus. Dann fährt der Algorithmus mit einer kleineren Matrix  $\hat{H}_1$  fort.

Bis zu diesem Punkt könnte es eine gute Idee sein, Verschiebungen in den  $QR$ -Algorithmus einzuführen. Da die Eigenwerte der Matrix unbekannt sind, können die perfekten Verschiebungen nicht gewählt werden! Daher ist eine Heuristik zur Schätzung der Eigenwerte erforderlich. Die **Rayleigh-Quotientenverschiebung** ist eine der Heuristiken: Setze die Verschiebung  $\sigma_k$  im  $k$ -ten Schritt des  $QR$ -Algorithmus gleich dem letzten Diagonalelement:

$$\sigma_k := h_{n,n}^{(k-1)} = e_n^* H^{(k-1)} e_n.$$

**Algorithmus 4: Der Hessenberg-QR-Algorithmus mit Rayleigh Quotientenverschiebung**

1.  $H_0 = H \in \mathbb{C}^{n \times n}$  sei eine obere Hessenberg-Matrix. Dieser Algorithmus berechnet seine  $H = UTU^*$ .
2.  $k := 0$ ;
3. **for**  $m = n, n-1, \dots, 2$  **do**
4.   **repeat**
5.      $k := k + 1$ ;
6.      $\sigma_k := h_{m,m}^{(k-1)}$ ;
7.      $H_{k-1} - \sigma_k I =: Q_k R_k$ ;
8.      $H_k := R_k Q_k + \sigma_k I$ ;
9.      $U_k := U_{k-1} Q_k$ ;
10.   **bis**  $|h_{m,m-1}^{(k)}|$  ausreichend klein ist
11. **end for**
12.  $T := H_k$ ;

Diese Heuristik ist in Algorithmus 4 implementiert. Es fällt auf, dass Verschiebungsänderungen in jedem Iterationsschritt existieren. Sobald das letzte niedrigere off-diagonal Element ausreichend klein ist, wird es zu Null gesetzt, und der Algorithmus fährt mit einer kleineren Matrix fort. Im Algorithmus 4.4 ist der 'aktive Teil' der Matrix  $m \times m$ .

### 3 Implementierung in C++

Alle oben besprochenen Eigenwert-Algorithmen sind im Rahmen dieser Arbeit in der Programmiersprache C++ implementiert worden. Dieser Abschnitt befasst sich hauptsächlich mit der Implementierung dieser Algorithmen.

Programmiersprachen können gemäß mehreren Kategorien aufgeteilt werden. Eine dieser Kategorien ist das verwendete Programmierparadigma, wobei sich mit der Entwicklung der Programmiersprachen neue Programmierparadigmen durchsetzen.

In den Jahren 1945 bis 1970 wurde gab es noch keine wirklichen Programmierparadigmen. Anfang der 1970er Jahre waren die prozedurale oder strukturelle Programmierung (Sprachen z.B. C, später Matlab) populär. Sie dient dazu, Klarheit, Qualität und Entwicklungszeit eines Programms zu verbessern. Ab 1990 wird die objektorientierte Programmierung (Sprachen z.B. C++, Java, C#) eingesetzt, um die Komplexität eines Programms zu reduzieren (Knuth 1993, S. 419).

#### 3.1 Mathematischer Datentypen und Matrix/Vektor-Operationen

Für den Eigenwert-Algorithmen werden Matrix- und Vektoroperationen benötigt. Da C++ keinen eingebauten Datentyp für Matrix- und Vektoroperationen bereitstellt, werden benutzerdefinierte Klassen verwendet. Im Rahmen dieses Projekts wurden die Klassen vorher schon implementiert und bereitgestellt. Lediglich die von diesen Klassen durchgeführten Operationen und auch Eigenwert-Algorithmen wurden in C++ implementiert. Diese Operationen sind Matrix-Multiplikation, Matrix-Vektor-Multiplikation, Transponieren, Matrix-Addition/Subtraktion, Vektor-Addition/Subtraktion, Vektor-Punktprodukt usw.

##### 3.1.1 Matrix-Addition/Subtraktion

Zwei Matrizen können nur addiert werden, wenn sie die gleichen Dimensionen haben. Das Ergebnis ist eine Matrix mit den gleichen Dimensionen. Um die Addition durchzuführen, werden Zahlen an gleichen Positionen in den Eingangsmatrizen addiert und das Ergebnis wird an der gleichen Position in der Ausgangsmatrix gespeichert. Der Algorithmus für die Matrixaddition ist im Folgenden angegeben:

##### Algorithmus 3.1

```

1. input:
2.   A, B: matrices
3.   N: number of rows
4.   M: number of columns
5. output:
6.   C: matrix(N, M)
7. assert:
8.       N==N && M==M
9. begin
```

```

10.         for i := 1 to N do
11.             for j := 1 to M do
12.                 C[i, j] := A[i, j] + B[i, j]
13.             endfor
14.         endfor
15.         return C
16.     end

```

Die Implementierung der Matrix-Subtraktionsoperation ist analog zur Matrix-Addition.

### 3.1.2 Matrix-Multiplikation

Zwei Matrizen können nur dann multipliziert werden, wenn die Anzahl der Spalten der ersten Matrix gleich der Anzahl der Zeilen der zweiten Matrix sein muss. Die Ergebnismatrix hat die Anzahl der Zeilen der ersten und die Anzahl der Spalten der zweiten Matrix. Um die Multiplikation durchzuführen, sind drei Schleifen erforderlich. Die beiden äußeren Schleifen definieren die Positionen der Elemente in den Matrizen, während die innere Schleife für die Multiplikationsoperation zuständig ist. Der Algorithmus für die Matrixmultiplikation ist im Folgenden angegeben:

#### Algorithmus 3.2

```

1. input:
2.   A, B: matrices
3.   N: number of rows of A matrix
4.   M: number of columns of A matrix
5.   J: number of rows of B matrix
6.   K: number of columns of B matrix
7. output:
8.   C: matrix(R=N, S=K)
9.   R: number of rows of C matrix
10.  S: number of columns of C matrix
11.  assert:
12.      M==J
13.  begin
14.      for i := 1 to N do
15.          for j := 1 to K do
16.              for k := 1 to M do
17.                  C[i, j] += A[i, k] * B[k, j]
18.              endfor
19.          endfor
20.      endfor
21.      return C
22.  end

```

### 3.1.3 Skalar-Matrix-Multiplikation/Division

Bei diesem Algorithmus führt der Skalar eine Multiplikations-/Divisionsoperation an jedem Element in der Matrix durch und das Ergebnis wird in der Ausgangsmatrix an der gleichen Stelle wie in der Eingangsmatrix gespeichert.

#### Algorithmus 3.3

```

1. input:
2.   A: matrix
3.   N: number of rows
4.   M: number of columns
5.   S: Scalar
6. output:
7.   C: matrix(N, M)
8. begin
9.   for i := 1 to N do
10.    for j := 1 to M do
11.      C[i, j] := S * A[i, j]
12.    endfor
13.  endfor
14.    return C
15.  end

```

Die Divisionsoperation ist analog zur Multiplikation.

### 3.1.4 Matrix-Transposition

Die Transponierung einer Matrix ist eine neue Matrix, deren Zeilen die Spalten des Originals sind. Für die Implementierung werden 2 Schleifen benötigt.

#### Algorithmus 3.4

```

1. input:
2.   A: matrix
3.   N: number of rows
4.   M: number of columns
5. output:
6.   C: matrix(M, N)
7. begin
8.   for i := 1 to N do
9.    for j := 1 to M do
10.      C[j, i] := A[i, j]
11.    endfor
12.  endfor

```

```
13.          return C
14.      end
```

### 3.1.5 Vektor-Addition/Subtraktion

Die Vektoraddition kann durchgeführt werden, wenn die Anzahl der Elemente der beiden Vektoren gleich ist. Die Elemente werden addiert und dann im Ausgangsvektor an der gleichen Stelle gespeichert.

#### Algorithmus 3.5

```
1. input:
2.   A, B: Vectors
3.   N: number of rows of A Vector
4.   M: number of rows of B Vector
5. output:
6.   C: Vector(N)
7. assert:
8.   N==M
9. begin
10.   for i := 1 to N do
11.       C[i] := A[i] + B[i]
12.   endfor
13.   return C
14. end
```

Die Vektorsubtraktion ist analog zur Addition.

### 3.2 Implementierung Eigenwert-Algorithmen

Alle oben besprochenen Eigenwert-Algorithmen sind erfolgreich in C++ implementiert. Zu diesem Zweck werden häufig bereits erstellte Matrix- und Vektorklassen und deren Operatoren verwendet. Es werden auch einige Zusatzfunktionen der linearen Algebra erstellt, die in den Algorithmen verwendet wurden. Diese Algorithmen können zur Berechnung von Eigenwerten und Eigenvektoren einer Matrix verwendet werden. Im Folgenden werden die Algorithmen in Blockdiagrammen dargestellt, um einen Überblick über den Aufbau des Programms in C++ zu geben.

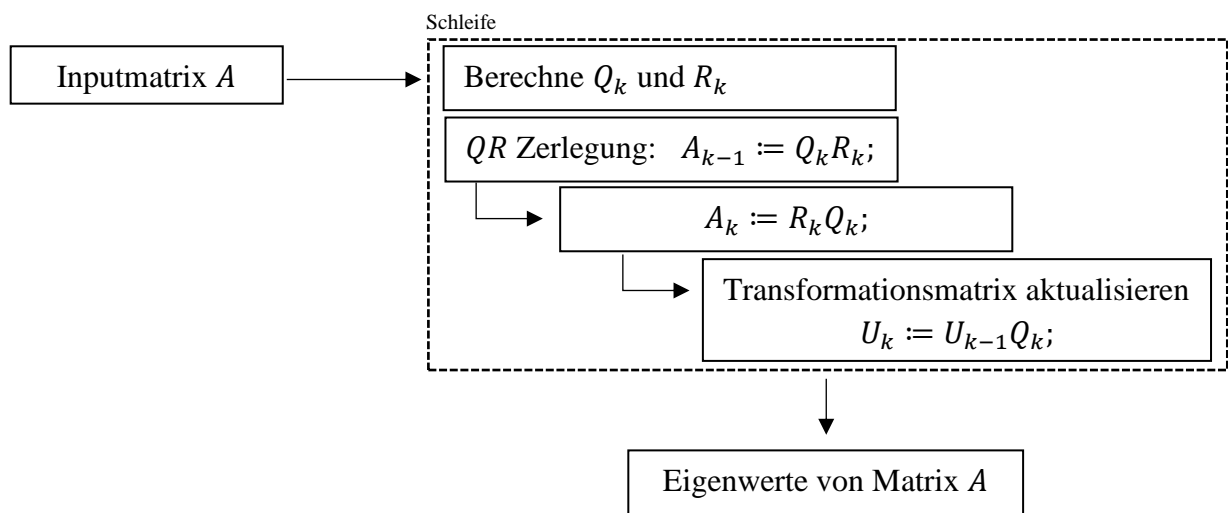
#### Algorithmus 1: Basic QR-Algorithmus

Der grundlegende Eigenwert-Algorithmus, der in Abschnitt 2.2.2.2 vorgestellt wurde, lautet wie folgt: Gegeben sei eine Matrix  $A$  (quadratische Eingabematrix), deren Eigenwerte bestimmt werden sollen. Sei  $A_0 = A$ , dann bestimme iterativ  $A_k$  aus  $A_{k-1}$ , indem die QR-Zerlegung  $Q_k$  und  $R_k$  von  $A_{k-1}$  bestimmt wird, so dass  $A_{k-1} = Q_k R_k$  gilt. Definiere anschließend  $A_k$  als  $A_k = R_k Q_k$ . Der grundlegende Algorithmus der QR-Zerlegung kann geschrieben werden als

$$A_{k-1} = Q_k R_k, \quad A_k = R_k Q_k = Q_k^* A_{k-1} Q_k,$$

Nach mehreren Iterationen können die Eigenwerte aus der Diagonalen der Matrix  $A$  gelesen werden, wenn diese zur oberen Dreiecksmatrix konvergiert.

In diesem Algorithmus werden Operationen wie Matrixmultiplikation und Matrixtransponierung verwendet.

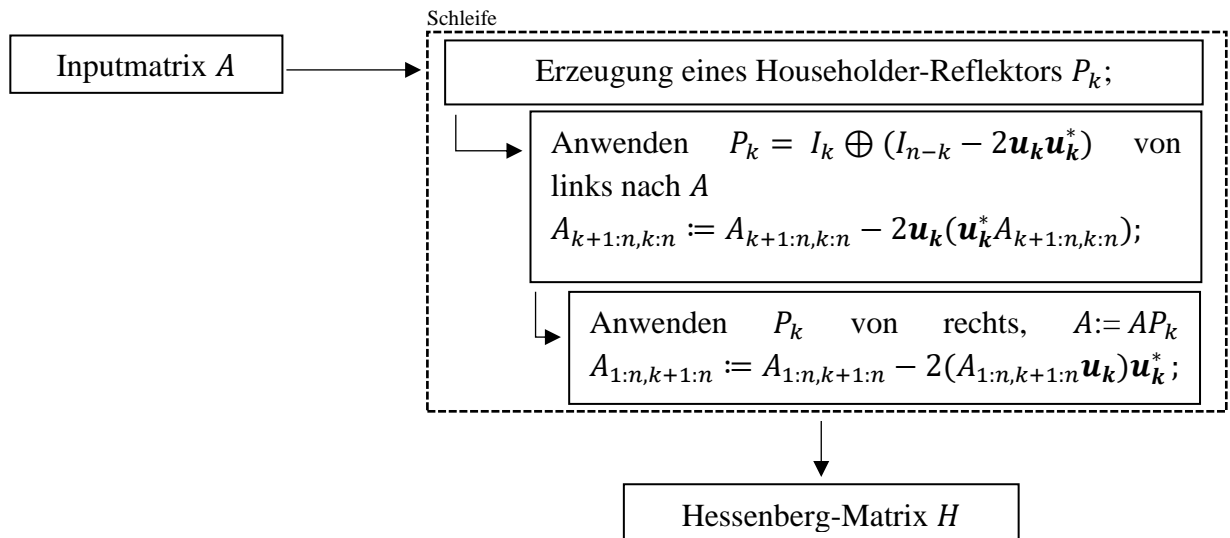


#### Algorithmus 2: Reduktion auf Hessenberg-Form

Dieses Algorithmus wurde bereits in Abschnitt 2.2.3.1 vorgestellt. Es reduziert die quadratische Eingabematrix mit Hilfe des Householder-Reduktions auf die Hessenberg-Matrix (eine obere Hessenberg-Matrix hat null Einträge unterhalb der ersten Subdiagonale und eine untere Hessenberg-Matrix hat null Einträge oberhalb der ersten Superdiagonale). Zu diesem Zweck wird der Householder-Reflektor verwendet, um eine geeignete Householder-Matrix zu konstruieren und diese mit der ursprünglichen Matrix  $A$  zu multiplizieren,

um die Hessenberg-Matrix  $H$  zu erhalten. Matrizen in Hessenbergform ermöglichen eine verbesserte Effizienz des Eigenwert-Algorithmus.

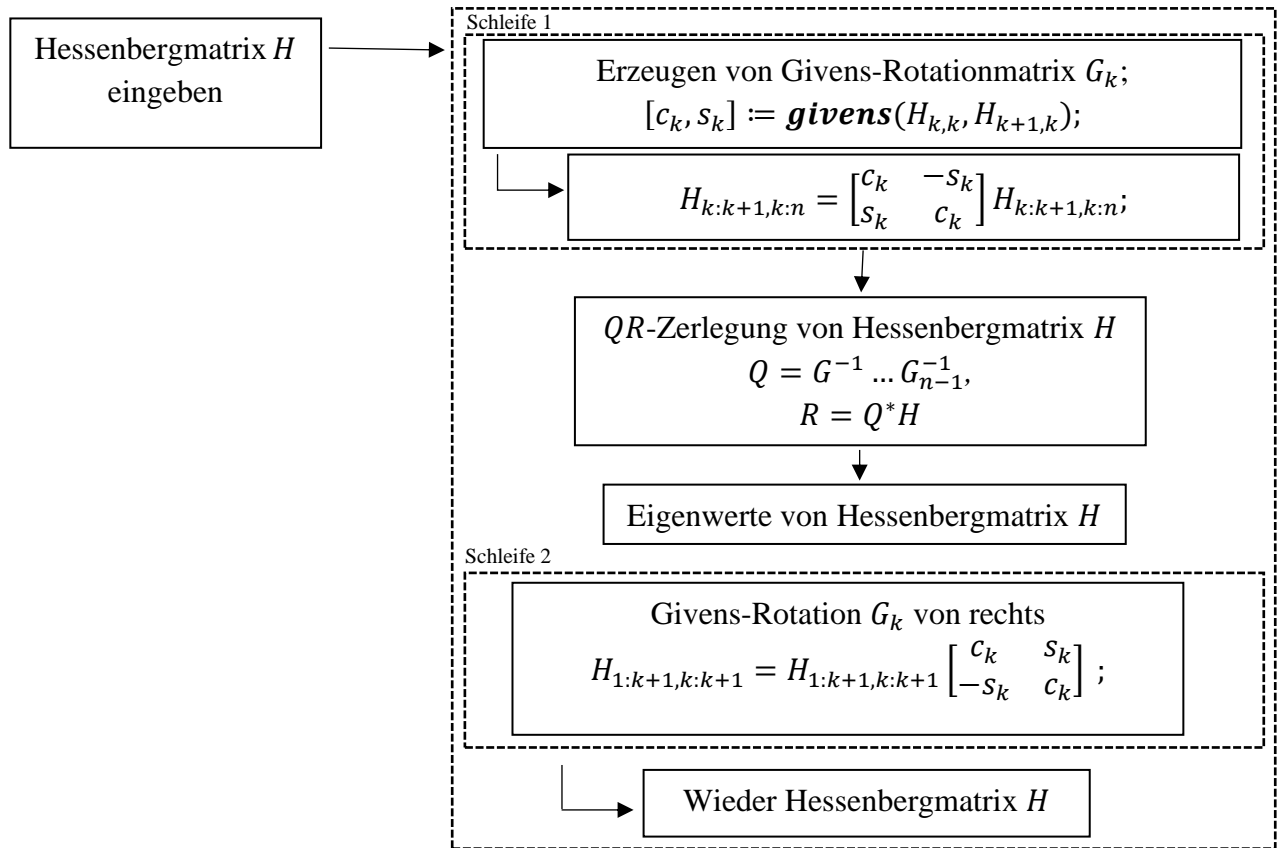
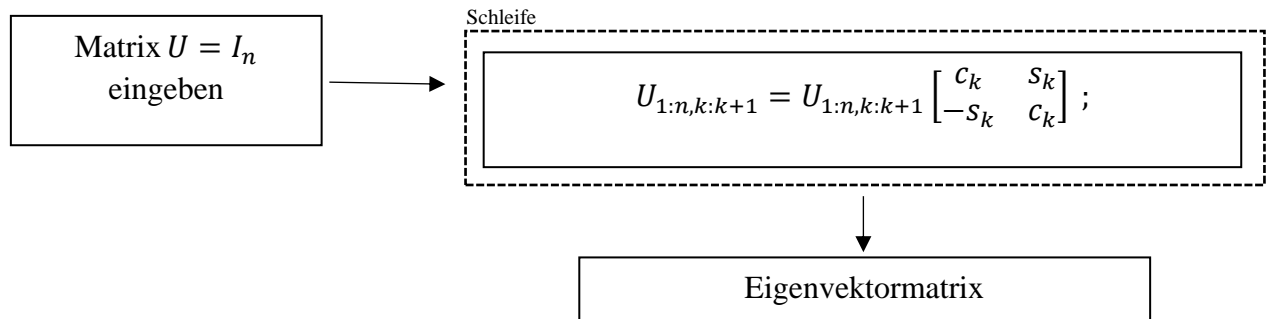
In diesem Algorithmus werden Operationen wie die Matrixmultiplikation und die Skalar-Matrix-Multiplikation verwendet.



### Algorithmus 3: Ein Hessenberg-QR-Schritt

Dieser Algorithmus wird in Abschnitt 2.2.3.2 detailliert erläutert. Bei diesem Algorithmus wird die Hessenberg-Matrix anstelle der Vollmatrix als Eingabematrix verwendet. Die QR-Zerlegung wird mit Hilfe der Givens-Rotation durchgeführt. Die Givens-Rotationsmatrix ist eine orthogonale Matrix, die oft verwendet wird, um eine reelle Matrix in eine äquivalente Matrix zu transformieren, typischerweise durch Löschen der Einträge unterhalb ihrer Hauptdiagonale. Nach der Anwendung der Givens-Rotation können die Eigenwerte auf der Diagonalen der Ausgabematrix abgelesen werden. Die Konvergenz dieses Algorithmus verbessert sich, aber die Iteration kann immer noch sehr langsam konvergieren, so dass zusätzliche Modifikationen erforderlich sind, um die QR-Iteration zu einem praktischen Algorithmus für die Berechnung der Eigenwerte einer allgemeinen Matrix zu verwenden.

In diesem Algorithmus werden Operationen wie Matrixmultiplikation und Matrixtransponierung verwendet.

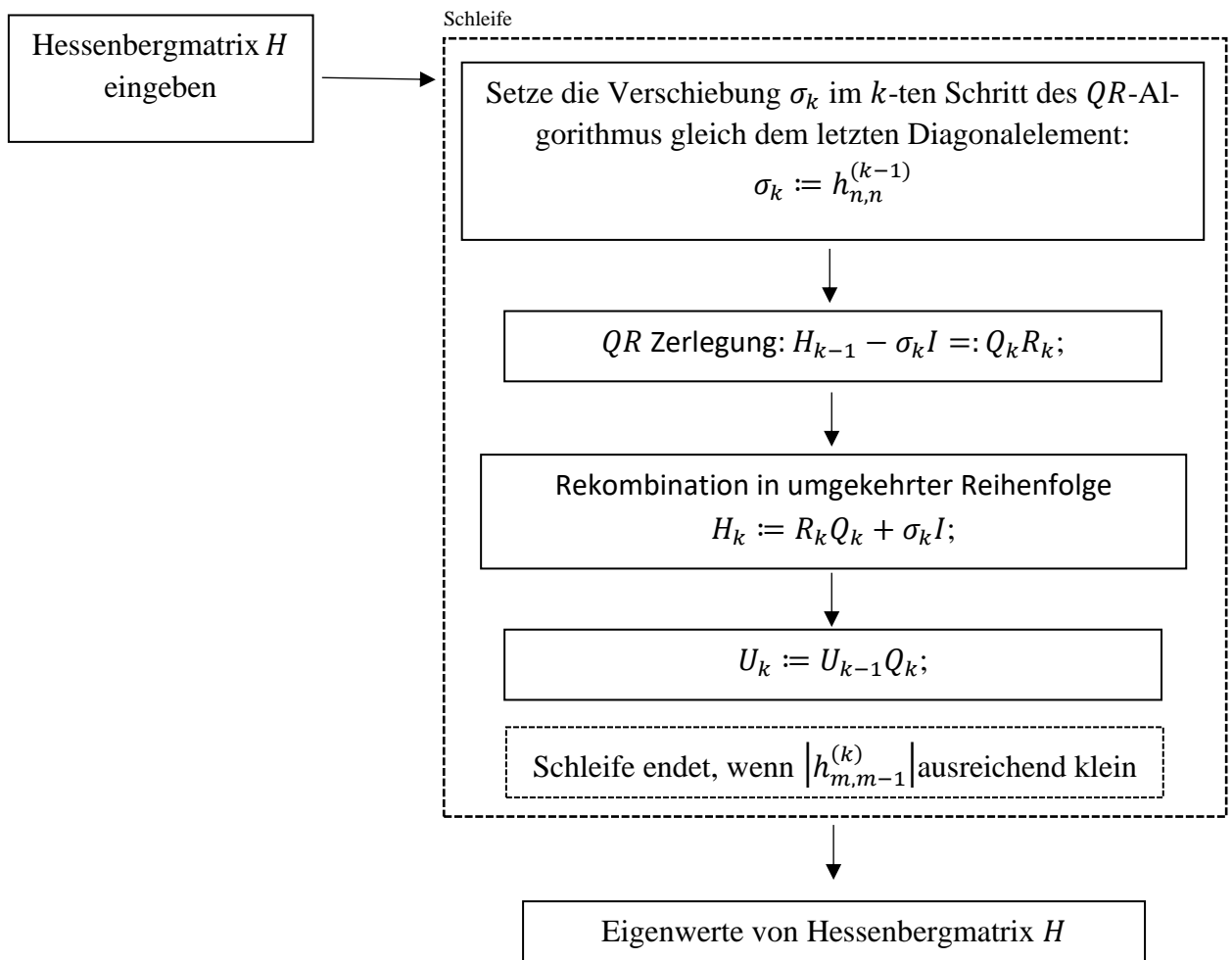
**Für Eigenvektoren**



**Algorithmus 4: Der Hessenberg-QR-Algorithmus mit Rayleigh Quotientenverschiebung**

Dieser Algorithmus wird in Abschnitt 2.2.3.4 behandelt. In diesem Algorithmus wird eine Verschiebungsstrategie eingeführt. Die perfekte Verschiebung kann jedoch nicht gewählt werden, da die genauen Eigenwerte der Eingangsmatrix nicht bekannt sind. Um die Eigenwerte zu schätzen, wird die Rayleigh-Quotienten-Verschiebung verwendet. Nach dem Ermitteln des Eigenwerts endet die Schleife und der Algorithmus fährt mit dem kleineren Wert für den nächsten Eigenwert fort. Die Einführung der Verschiebungsstrategie in diesem Algorithmus beschleunigt die Konvergenz des Algorithmus.

In diesem Algorithmus werden Operationen wie Matrixmultiplikation und Matrixtransponierung verwendet.



## 4 Schwingungstechnik Beispiel

In Abschnitt 1.2.1 wird das Beispielproblem analytisch bis zur Bildung der Systemmatrix gelöst. In diesem Abschnitt wird diese Matrix (um Eigenwerte und damit Eigenvektoren zu erhalten) mit Hilfe eines Eigenwert-Algorithmus gelöst, der in C++ implementiert ist. Dann werden die Ergebnisse grafisch analysiert. Am Ende dieses Abschnitts wird auch die Zeit analysiert, die jeder Eigenwert-Algorithmus (die in den vorhergehenden Kapiteln besprochen wurden) benötigt, um dieses Problem zu lösen.

Die in Abschnitt 1.2 gebildete Systemmatrix ist angegeben als:

$$\ddot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

Resultierende Eigenwerte sind

$$\lambda_1 = -3.7321, \quad \lambda_2 = -3.0000, \quad \lambda_3 = -2.0000, \quad \lambda_4 = -1.0000, \quad \lambda_5 = -0.2679$$

und Eigenvektoren sind

$$\mathbf{v}_1 = \begin{bmatrix} -0.2887 \\ 0.5000 \\ -0.5774 \\ 0.5000 \\ -0.2887 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} -0.5000 \\ 0.5000 \\ -0.5000 \\ -0.5000 \\ 0.5000 \end{bmatrix}, \mathbf{v}_3 = \begin{bmatrix} 0.5774 \\ -0.0000 \\ -0.5774 \\ -0.0000 \\ 0.5774 \end{bmatrix}, \mathbf{v}_4 = \begin{bmatrix} 0.5000 \\ 0.5000 \\ -0.0000 \\ -0.5000 \\ -0.5000 \end{bmatrix}, \mathbf{v}_5 = \begin{bmatrix} -0.2887 \\ -0.5000 \\ -0.5774 \\ -0.5000 \\ -0.2887 \end{bmatrix}$$

Diese Eigenvektoren stellen die Modenform des Systems dar.

Frequenzen  $\omega$  sind

$$\omega_1 = 1.93, \quad \omega_2 = 1.73, \quad \omega_3 = 1.41, \quad \omega_4 = 1.00, \quad \omega_5 = 0.52$$

Im Folgenden wird das Verhalten des Systems grafisch dargestellt

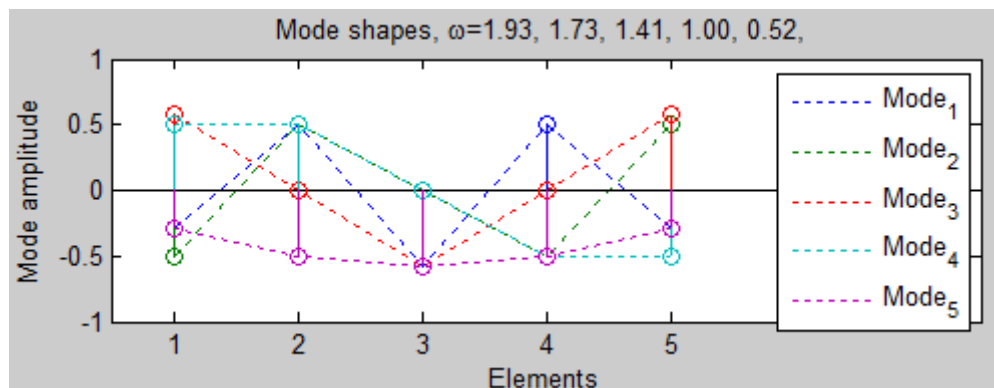
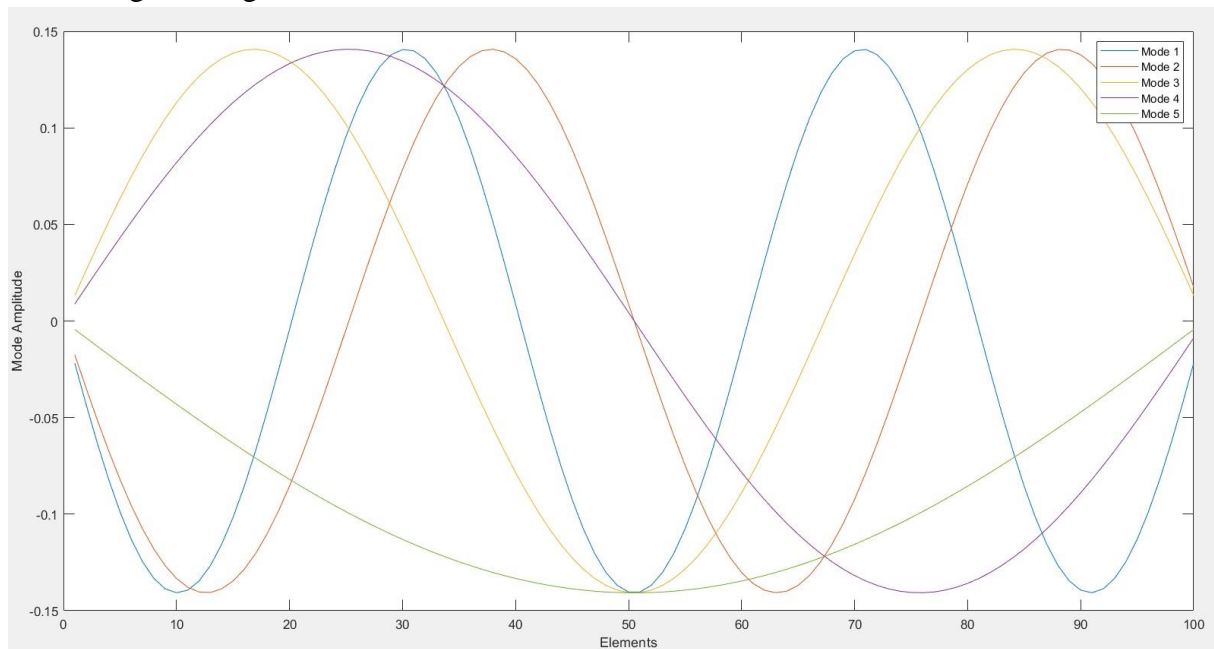


Abbildung 4.1: Verhalten des Systems (Cheever 2005)

Abbildung 4.1 gibt einen Einblick in das Verhalten des Systems. Es ist zu erkennen, dass der Modus mit der niedrigsten Frequenz (Modus 5) die "glatteste" Form hat. Die Formen werden von Modus 4 bis Modus 1 zunehmend oszillierend; in jedem Fall steigt die Frequenz der Oszillation an (Cheever 2005). Durch Veränderung der Anfangsverschiebung an den Massen können unterschiedliche Modenformen beobachtet werden.

Wenn z. B. das oben erläuterte Beispiel auf 100 Elemente erweitert wird, kann die Modenform für die 5 kleinsten Eigenvektoren (verbunden mit 5 kleinsten Eigenwerten) als Sinusform in Abbildung 4.2 dargestellt werden.



**Abbildung 4.2: Verhalten des Systems (100 Elemente)**

Nun wird die Leistung jedes Algorithmus zur Lösung dieses Problems analysiert. Microsoft Visual Studio 2017" wird auf dem privaten Rechner (Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz) zur Ausführung der Algorithmen verwendet. Die von jedem Algorithmus benötigte Zeitdauer wird mit Hilfe der Chrono-Bibliothek registriert. Sie dient zur Verwaltung von Datum und Uhrzeit. Sie wird auch verwendet, um die während der Ausführung eines C++-Programms verbrauchte Zeit zu messen. Es wird oft beobachtet, dass Zeitgeber und Uhren auf verschiedenen Systemen unterschiedlich sind. Um diesem Problem entgegenzuwirken und die Zeit in Bezug auf die Präzision zu verbessern, wurde die Bibliothek entworfen.

Die Ergebnisse zeigten eine Verbesserung der Leistung des Eigenwert-Algorithmus und die Lösung des Problems mit dem verbesserten Eigenwert-Algorithmus spart Rechenzeit. Im Folgenden wird die Leistung jedes Algorithmus in Bezug auf die benötigte Zeit dargestellt.

**Tabelle 4.2: Effizienzvergleich für 100-Massen-System**

Algorithmus	Zeitaufwand
Basic-Eigenwert-Algorithmus	3,721 sec
Hessenberg-QR-Algorithmus	1,986 sec
Hessenberg-QR-Algorithmus mit Shift	0,182 sec

**Tabelle 4.3: Effizienzvergleich für 150-Massen-System**

<b>Algorithmus</b>	<b>Zeitaufwand</b>
Basic-Eigenwert-Algorithmus	10,077 sec
Hessenberg-QR-Algorithmus	7,311 sec
Hessenberg-QR-Algorithmus mit Shift	0,592 sec

**Tabelle 4.4: Effizienzvergleich für 200-Massen-System**

<b>Algorithmus</b>	<b>Zeitaufwand</b>
Basic-Eigenwert-Algorithmus	19,505 sec
Hessenberg-QR-Algorithmus	15,505 sec
Hessenberg-QR-Algorithmus mit Shift	1,435 sec

Die Zeit kann sich mit jeder Ausführung ändern, aber sie ist relativ zueinander gleich.

Die Algorithmen werden auch mit 50, 100 und 150 Elementen getestet. Die Ergebnisse werden dann mit den aus Matlab berechneten Werten verglichen. Es wird festgestellt, dass aufgrund der Nachkommastellen die Genauigkeit sinkt, aber insgesamt ist sie akzeptabel. Mit zunehmender Anzahl der Elemente steigt auch die für die Lösung des Problems benötigte Rechenzeit deutlich an. Zum Beispiel wird ein Test mit dem Basic-Eigenwert-Algorithmus durchgeführt, um die Laufzeit für 5 bis 200 Elementen gegen eine unterschiedliche Anzahl von Iterationen zu berechnen. Aus dem Diagramm 4.3 geht hervor, dass bei kleinerer Elementanzahl die Anzahl der Iterationen keine große Rolle für die Laufzeit spielt, aber bei größeren Matrizen steigt die Laufzeit mit zunehmender Anzahl der Iterationen deutlich an. Weiterhin ist der Algorithmus teuer. Für jeden Iterationsschritt ist es erforderlich, die QR-Zerlegung der kompletten  $n \times n$ -Matrix zu berechnen, d. h. jeder einzelne Iterationsschritt hat eine Komplexität  $\mathcal{O}(n^3)$  (Arbenz 2016, S.67). Dies erhöht die Gesamtlaufzeit des Algorithmus, insbesondere bei größeren Matrizen.

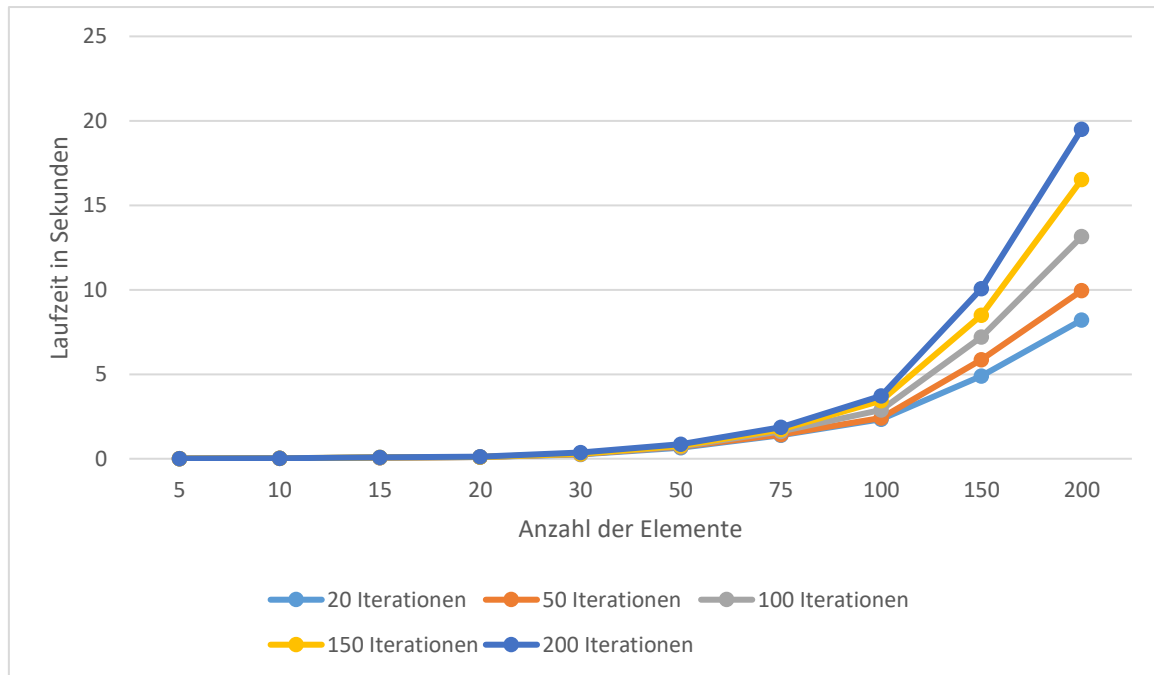


Abbildung 4.3: Laufzeitplot von Basic-Eigenwert-Algorithmus

Im Vergleich zum Basis-Eigenwert-Algorithmus zeigt der Hessenberg-QR-Algorithmus eine bessere Konvergenz. In Abbildung 4.4 ist zu erkennen, dass die Laufzeit quadratisch ansteigt. Das liegt daran, dass jeder Iterationsschritt die Komplexität von  $\mathcal{O}(n^2)$  erfordert.

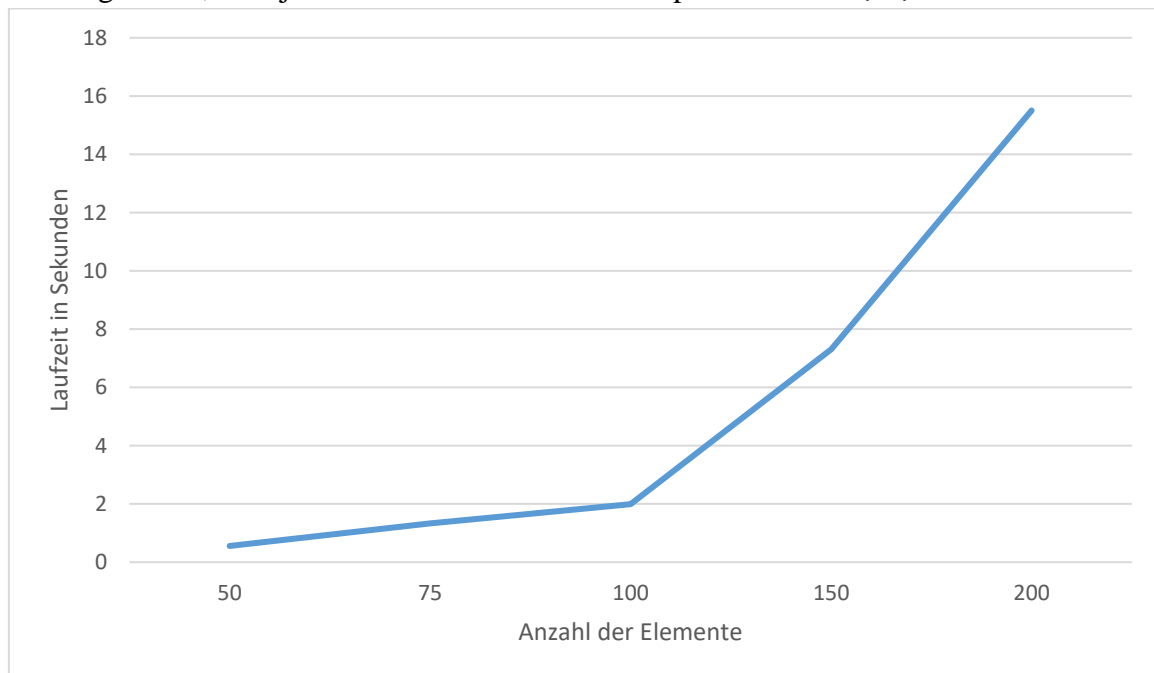
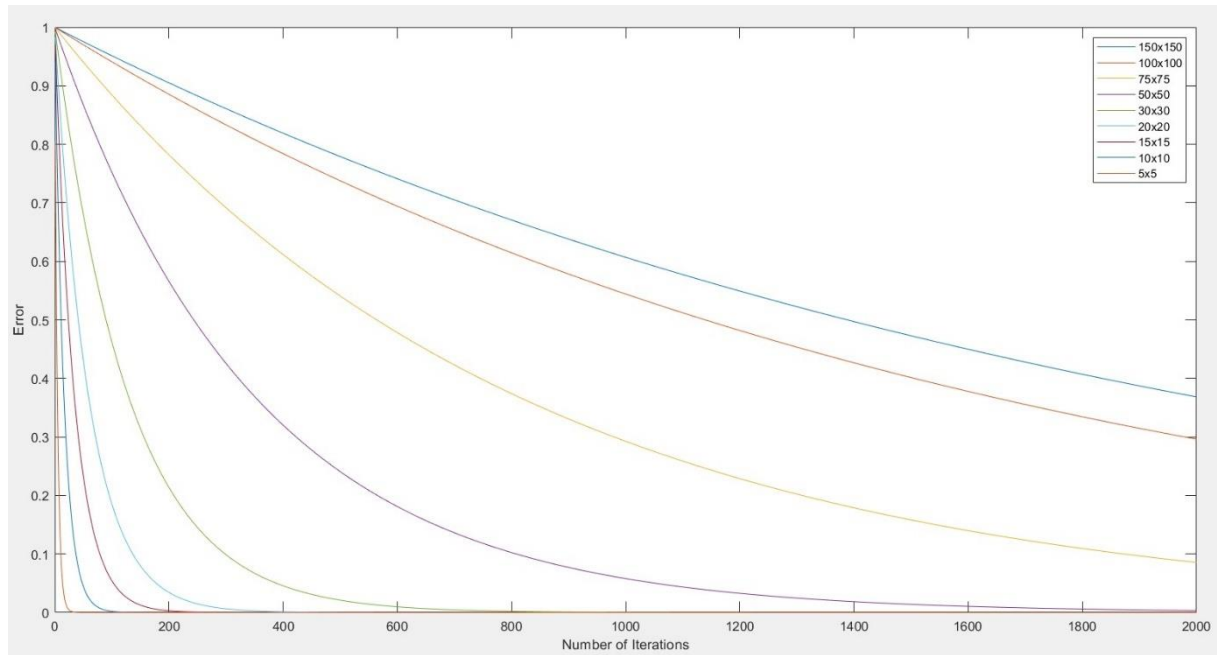


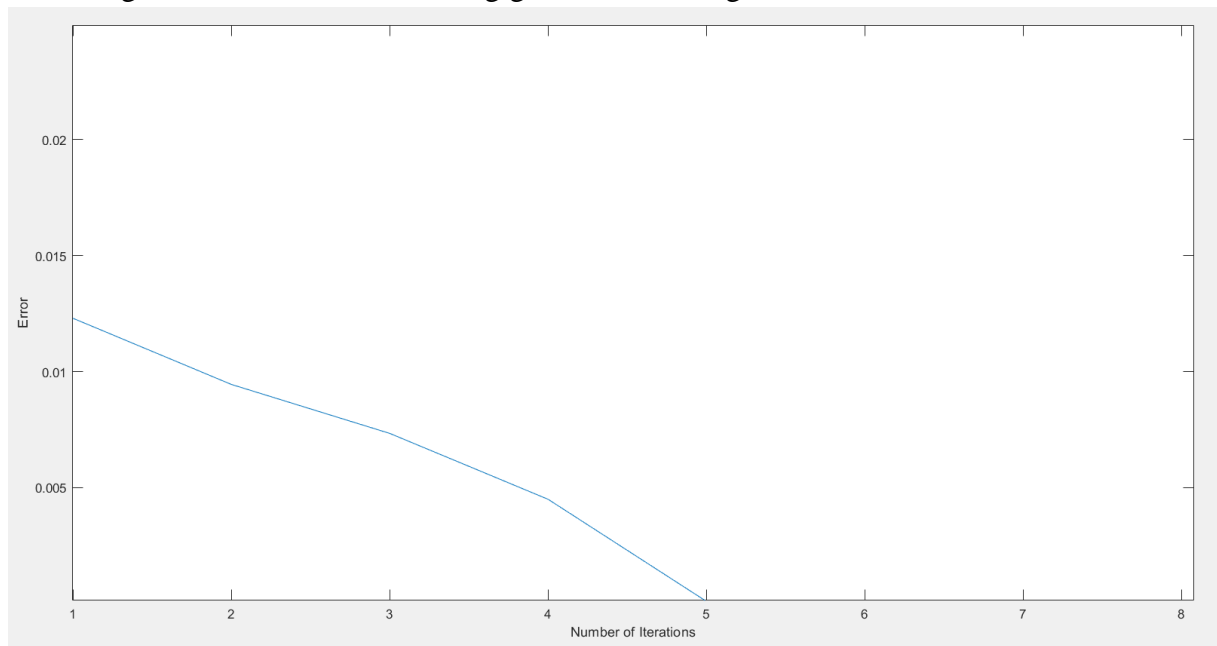
Abbildung 4.4: Laufzeitplot von Hessenberg-QR-Algorithmus

In Abbildung 4.5 ist zu erkennen, dass die Gesamtkonvergenzrate des Basic-Eigenwert-Algorithmus sehr langsam ist. Insbesondere für größere Matrizen wird eine hohe Anzahl von Iterationen benötigt, um zu konvergieren.



**Abbildung 4.5: Konvergenz des Basic-Eigenwert-Algorithmus für verschiedene Matrixgrößen**

Im Gegensatz dazu ist die Konvergenz des Hessenberg-QR-Algorithmus mit Shift sehr schnell. Durch die Einführung der Verschiebungsstrategie dauert es nur wenige Iterationen, um den ersten Eigenwert der Matrix unabhängig von der Matrixgröße zu berechnen.



**Abbildung 4.6: Konvergenz des Hessenberg-QR-Algorithmus mit Shift**

## 5 Zusammenfassung und Ausblick

Die Hauptaufgabe dieser Projektarbeit ist die Implementierung eines effizienten Eigenwertlösers unter Verwendung von C++. Die sekundäre Aufgabe umfasst die Lösung des Ingenieurproblems mit Hilfe des Eigenwertlösers.

Dieses Projekt kann in 3 Phasen unterteilt werden

Phase 1 umfasst die Planung des Projekts und die Auswahl der zu verwendenden Programmiersprache. Auch das Eigenwertproblem wurde analytisch gelöst.

Die zweite Phase ist hauptsächlich der Implementierung von Algorithmen in C++ gewidmet. In dieser Phase wurden die mathematischen Operatoren der Matrix- und Vektorklassen in C++ implementiert. Danach wurden Eigenwert-Algorithmen zur Berechnung von Eigenwerten und Eigenvektoren implementiert. Diese Algorithmen unterscheiden sich voneinander durch die benötigte Laufzeit. Konvergenzen der Algorithmen wurden ebenfalls diskutiert. Um die Algorithmen übersichtlicher zu gestalten und einen Einblick in das C++-Programm zu geben, wurden Boxdiagramme zu jedem Algorithmus erstellt.

Auch Eigenwertprobleme und ihre Bedeutung in der Technik werden diskutiert. Es wird auch gezeigt, wie Eigenwerte durch kommerzielle FE-Solver berechnet werden können. Die Analyse wird an einem freitragenden Träger durchgeführt und anschließend werden die Ergebnisse mit den analytischen Ergebnissen verglichen.

Die Konvergenz des Eigenwert-Algorithmus kann weiter verbessert werden, indem effizientere Algorithmen verwendet werden, wie z. B. "der Double-Shift-QR-Algorithmus" und "der Tridiagonal-QR-Algorithmus".

## 6 Literaturverzeichnis

1. Cheever, Eric.: Lecture Notes on Linear Physical Systems Analysis, Swarthmore College, 2005
2. Mekalke, C. Sutar, V.: Modal Analysis of Cantilever Beam for Various Cases and its Analytical and FEA Analysis, International Journal of Engineering Technology, Management and Applied Sciences, ISSN 2349-4476, Volume 4, Issue 2, Feb 2016
3. Beutelspacher, A. : Lineare Algebra: Eine Einführung in die Wissenschaft der Vektoren, Abbildungen und Matrizen, Springer, 2014
4. Chapra, S. Canale, R. : Numerical Methods for Engineers, McGraw Hill, 2014
5. Ford, W. : Numerical Linear Algebra with Applications: Using MATLAB, Academic Press; First edition, 2015
6. Arbenz, P. : Lecture Notes on Solving Large Scale Eigenvalue Problems, ETH Zürich, Chapter 4, Spring Semester 2016
7. Knuth, E. Pardo, Tr.: Early development of programming languages. Encyclopedia of Computer Science and Technology. Marcel Dekker.
8. Stroustrup, B. : Einführung in die Programmierung mit C++, München [u.a.], Pearson Studium, 2010

### Sonstige Quellen

9. Stroustrup, B. : Die C++-Programmiersprache [vom Erfinder von C++], München [u.a.], Addison Wesley, 2011
10. Kowalsky, H. : Einführung in die lineare Algebra, De Gruyter Lehrbuch, 1977
11. Tilo, A. Frank, H. : Mathematik, Berlin, Heidelberg, Springer Verlag, 4. Auflage. 2018
12. Deuflhard, Peter: Numerische Mathematik 1: Eine algorithmisch orientierte Einführung, Hanser, 2019