Technische Universität Dresden • Fakultät Mathematik

Gewichtete Zählung von Eigenwerten auf einem Intervall

Bachelorarbeit

zur Erlangung des ersten Hochschulgrades

Bachelor of Science (B.Sc.)

vorgelegt von

Noah GÖPEL

(geboren am 01.02.2003 in Riesa)

Tag der Einreichung: 26. 09. 2024

Prof. Dr. Oliver Sander (Institut für Numerische Mathematik)

Inhaltsverzeichnis

Symbolverzeichnis

s	Design-Parameter
I_n	Einheitsmatrix
$A - \lambda B$	Element des Matrix-Pencils (A, B)
$\lambda(A, B)$	Eigenwerte des Matrix-Pencils $A - \lambda B$
$\lambda(A)$	Eigenwerte der Matrix A
M	Massenmatrix
K	Steifigkeitsmatrix
$\overrightarrow{F_L}$	Federkraft
$\overrightarrow{F_T}$	Trägheitskraft
x_k	Auslenkung des k-ten Massepunkts aus der Ruhelage
ω	Eigenkreisfrequenz
$[\omega_a, \omega_b]$	vorgegebenes Intervall
λ	Eigenwert des Matrix-Pencils, definiert durch $\lambda := \omega^2$
$[\lambda_a, \lambda_b]$	Intervall, definiert durch $[\lambda_a, \lambda_b] := [\omega_a^2, \omega_b^2]$
$J(s)$	Funktion, die Eigenwerte auf $[\lambda_a, \lambda_b]$ gewichtet zählt
γ	positiv orientierter Kreis in der komplexen Ebene
$J^*(s)$	durch Quadraturformel approximierte Funktion $J(s)$

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Berechnung von Eigenfrequenzen hat in der Mechanik einen besonderen Stand:

Falls eine Erregerfrequenz nahe einer Eigenfrequenz des mechanischen Systems liegt, tritt Resonanz auf (vgl. [1, S. 435]). Aufgrund der Resonanz reichen kleine Kräfte aus, um ein mechanisches System in so große Schwingung zu versetzen, dass die Materialien beschädigt werden können. In Extremfällen kann Resonanz auch zur Zerstörung des Systems führen, ein Beispiel dafür ist die Broughton Suspension Bridge, welche einstürzte, weil eine Armee im Gleichschritt über sie marschierte (vgl. [2]).

Daher ist es von entscheidender Relevanz, durch bestimmte Maßnahmen die Eigenfrequenzen eines mechanischen Systems so weit weg von den Bereichen der möglichen Erregerfrequenzen zu legen, wie möglich (vgl. [1, S. 520]). Man definiere daher ein Intervall $[\omega_a, \omega_b]$, in welchem zum Schluss keine Eigenfrequenzen liegen sollen.

Um diese Systeme formal zu beschreiben, kann man die Massen- und Steifigkeitsmatrix eines Systems bestimmen, anhand derer können durch die Berechnung von bestimmten Eigenwerten die Eigenfrequenzen ermittelt werden. Da durch die Maßnahmen auch diese Matrizen verändert werden, besitzen diese Matrizen eine Abhängigkeit von einem Parameter, der hier „Design-Parameter“ [3, S. 2][1] genannt wird und mit s bezeichnet wird. Der Design-Parameter soll dabei aus einer Menge Ω an zulässigen Parametern stammen.

Diese Ausarbeitung beschäftigt sich daher mit der Zählung von Eigenwerten auf einem vorgegebenen Intervall. Dazu werden in Kapitel 2 wichtige Aussagen vorgestellt, welche die Berechnung der Eigenwerte und somit auch der Eigenfrequenzen vereinfachen. In Kapitel 3 werden zwei einfache Beispielsysteme aufgestellt und die zugehörigen Massen- und Steifigkeitsmatrizen berechnet.

Anschließend werden in Kapitel 4 Formeln zur Zählung der Eigenwerte hergeleitet und so umgestellt, dass diese Formel direkt vom Design-Parameter s abhängt.

In Kapitel 5 wird die Zählung der Eigenwerte implementiert und ein Minimierungsverfahren angewendet, welches den Parameter s so verändern soll, dass $s \in \Omega$ gilt und alle Eigenwerte außerhalb des vorgegebenen Intervalls liegen.

Verbesserungen der Implementierung folgen in Kapitel 6 und die Zusammenfassung der Erkenntnisse in Kapitel 7.

[1] Übersetzung des Autors

Kapitel 2

Das allgemeine Eigenwertproblem und die Identität von Futamura

Zu Beginn der Ausarbeitung werden einige Resultate über Matrizen und lineare Algebra behandelt. Diese werden in Kapitel 4 verwendet, um die Zählung der Eigenwerte in eine Funktion $J(s)$ umzuformen, die direkt von s abhängt.

2.1 Der Matrix-Pencil

Betrachte zunächst das Konzept des „Matrix-Pencils“ [4, S. 32], welches in dieser Ausarbeitung eine entscheidende Rolle spielt, denn viele Aussagen aus Kapitel 2 beziehen sich auf eine Menge von Matrizen dieser Form.

Definition 1. (pencil, vgl. [5, S. 375])
Seien A und B Matrizen in $\mathbb{C}^{n\times n}$, dann wird die Menge aller Matrizen der Form $A - \lambda B, \lambda \in \mathbb{C}$ Matrix-Pencil genannt.

Diese Menge vom Matrizen wird auch mit (A, B) bezeichnet (siehe [6, S. 373]).
Laut [5, S. 375] seien die Eigenwerte von $A - \lambda B$ diejenigen $\lambda \in \mathbb{C}$, für die

$$(A - \lambda B)x = 0, \quad x \neq 0\,. \tag{2.1}$$

gelte, $x \in \mathbb{C}^n$ wird dann „Eigenvektor“ [5, S. 375][1] genannt. Man beachte, dass für $B = I_n$ die Gleichung (2.1) zu

$$(A - \lambda I_n)x = 0 \Leftrightarrow Ax = \lambda x$$

wird, also genau die Form des „speziellen Eigenwertproblems“ [1, S. 381]. Da für ein spezielles Eigenwertproblem die Menge $\lambda(A)$ gesucht wird, wird die Suche nach den $\lambda \in \mathbb{C}$, die (2.1) erfüllen, auch „allgemeines Eigenwertproblem“ [1, S. 380] genannt, da sie eine größere Klasse von Gleichungen abdeckt. Falls man $A - \lambda B$ durch die Matrix C ersetzt, so entsteht aus (2.1) das Gleichungssystem

$$Cx = 0, \quad x \neq 0\,.$$

[1] Übersetzung des Autors

Da

$$\det(C) \neq 0 \Leftrightarrow Cx = 0 \text{ hat als einzige Lösung } x = 0$$

gilt[2], folgt durch Kontraposition:

$$\det(C) = 0 \Leftrightarrow \exists x \neq 0 : Cx = 0\,.$$

Somit kann durch Finden einer Nullstelle λ' des Polynoms $\det(A - \lambda B)$ sichergestellt werden, dass ein zugehöriger Vektor $x' \neq 0$ existiert, sodass λ' und x' Gleichung (2.1) lösen. Man findet die Eigenwerte des Matrix-Pencils also durch die Berechnung der Nullstellen dieses Polynoms.

Daher wird in [5, S. 375] die Menge der Eigenwerte des Matrix-Pencils $A - \lambda\, B$ auch als

$$\lambda(A, B) := \{z \in \mathbb{C} :\ \det(A - zB) = 0\} \tag{2.2}$$

definiert. Laut [5, S. 375] gelte zudem

$$A - \lambda\, B \text{ hat } n \text{ Eigenwerte} \Leftrightarrow \operatorname{rang}(B) = n\,. \tag{2.3}$$

Dieses Resultat ist für Kapitel 4 relevant und kann mithilfe von Theorem 5 leicht gezeigt werden.

Man benötigt für die Identität von Futamura aus Kapitel 2.3 obendrein:

Definition 2. (Regulärer Matrix-Pencil, vgl. [6, S. 376])
Ein Matrix-Pencil $A - \lambda B$ wird regulär genannt, wenn $A, B \in \mathbb{C}^{n\times n}$ und

$$\exists \lambda \in \mathbb{C} : \det(A - \lambda B) \neq 0$$

gilt.

Bermerkung 3. Diese Definition bedeutet insbesondere, dass der Matrix-Pencil regulär ist, falls B regulär ist.

Beweis:

$$\begin{aligned} B \text{ regulär } &\Leftrightarrow \operatorname{rang}(B) = n \Leftrightarrow |\lambda(A, B)| = n < |\mathbb{C}| \\ &\Rightarrow \exists z \in \mathbb{C} \setminus \lambda(A, B) : \det(A + zB) \neq 0 \Leftrightarrow A - \lambda\, B \text{ regulär} \quad \square \end{aligned}$$

wobei (2.3) und die Überabzählbarkeit von $\mathbb{C}$ verwendet wurde. Diese Aussage kann auch in [6, S. 376] gefunden werden.

2.2 Die Schur-Zerlegung

In diesem Abschnitt wird eine wichtige Zerlegung vorgestellt, die für Kapitel 2.3 benötigt wird. Dazu benötigt man noch die Definition der unitären Matrizen:

Definition 4. (Unitary Matrices, [5, S. 73])
Sei $Q \in \mathbb{C}^{n\times n}$, dann wird Q unitär genannt, wenn

$$Q^H Q = Q Q^H = I_n\,.$$

[2]Beweis: $\det(C) \neq 0 \Leftrightarrow C$ regulär $\Leftrightarrow$ Lösung von $Cx = b :\ x = C^{-1}b$

Hier ist Q^H die Adjungierte von Q, also die Matrix, die durch komplexe Konjugation und Transponierung von Q entsteht. Man bemerke, dass die komplexe Konjugation auf jeden Eintrag der Matrix Q einzeln angewendet wird (vgl. [5, S. 14]).

Mithilfe dieser Definition kann nun die allgemeine Schur-Zerlegung vorgestellt werden, welche für den Beweis der Identität von Futamura in Kapitel 2.3 benötigt wird.

Theorem 5. *(Generalized Schur Decompositon, [5, S. 377])*
Seien $A, B \in \mathbb{C}^{n\times n}$, dann existieren Q und Z unitär, sodass

$$Q^H AZ = T, \quad Q^H BZ = S \tag{2.4}$$

mit T und S obere Dreiecksmatrizen.

Falls ein $k \in \{1, \ldots, n\}$ existiert, sodass $t_{kk} = s_{kk} = 0$, dann gilt $\lambda(A, B) = \mathbb{C}$, sonst

$$\lambda(A, B) = \{t_{ii}/s_{ii} : \ s_{ii} \neq 0\}\,. \tag{2.5}$$

Beweis: siehe [5, S. 377]. □

2.3 Die Identität von Futamura

Nach den Vorbereitungen der vorherigen Abschnitte kann nun die Identität von Futamura beschrieben und gezeigt werden.

Für den Beweis von Theorem 9 benötigt man noch einige Aussagen über obere Dreiecksmatrizen. X_{ij} bezeichne in diesem Abschnitt den Eintrag der Matrix X in Zeile i und Spalte j.

Lemma 6. *Die Inverse einer oberen Dreiecksmatrix A ist eine obere Dreiecksmatrix.*

Beweis: Anwenden des Gauß-Jordan-Algorithmus zur Bestimmung der Inversen auf $(A|I_n)$ mit A rechte obere Dreiecksmatrix. □

Lemma 7. *Seien X und Y obere Dreiecksmatrizen, dann gilt:*

$$(XY)_{ii} = X_{ii}\, Y_{ii}\,.$$

Beweis:

$$(XY)_{ii} = \sum_{k=1}^{n} X_{ik}Y_{ki} = \sum_{k=I}^{n} X_{ik}Y_{ki} = \sum_{k=I}^{i} X_{ik}Y_{ki} = X_{ii}Y_{ii},$$

wobei verwendet wurde, dass für eine obere Dreiecksmatrix A nach Definition $A_{ij} = 0$ für $i > j$ gilt. □

Lemma 8. *Für eine invertierbare obere Dreiecksmatrix X gilt*

$$(X^{-1})_{ii} = \frac{1}{X_{ii}}\,.$$

Beweis: Sei $i \in \{1, \ldots, n\}$ fest, dann gilt

$$I_n = X\,X^{-1} \Rightarrow 1 = X_{ii}(X^{-1})_{ii} \Rightarrow (X^{-1})_{ii} = \frac{1}{(X)_{ii}}\,.$$

Hier wurde Lemma 7 und

$$X \text{ invertierbar} \Rightarrow \det X = \prod_{k=1}^{n} X_{kk} \neq 0 \Rightarrow X_{kk} \neq 0 \quad \forall k = 1, \ldots, n$$

verwendet. □

Das folgende Theorem wird in Kapitel 4 von entscheidender Relevanz sein, da durch diese Aussage in dieser Arbeit die Verteilung der Eigenwerte von (A, B) auf den Design-Parameter s zurückgeführt werden kann. Dieses Theorem lehnt sich stark an die Identität von Futamura aus [7, S. 127] an, hier wurde aber die Menge N eingeführt, um nicht beweisen zu müssen, dass

$$s_{ii} \begin{cases} \neq 0 & : i \leq n' \\ = 0 & : i > n' \end{cases}$$

gilt.

Theorem 9. *(Identität von Futamura)*
Seien $A, B \in \mathbb{C}^{n\times n}, z \in \mathbb{C}$, sodass $(zB - A)$ ein regulärer Matrix-Pencil ist, dann gibt es unitäre Matrizen Q und $U \in \mathbb{C}^{n\times n}$, sodass

$$A = QTU^H, \quad B = QSU^H$$

und

$$tr((zB - A)^{-1}B) = \sum_{j\in N} \frac{1}{z - \lambda_j}, \tag{2.6}$$

gilt, wobei $N := \{i \in \{1, \ldots, n\} : s_{ii} \neq 0\}$ und λ_j die Eigenwerte des Matrix-Pencils sind. S und T sind hier obere Dreiecksmatrizen.

Bermerkung 10. Es gilt folgende Äquivalenz:

$$(\lambda B - A) \text{ regulär} \Leftrightarrow (A - \lambda B) \text{ regulär}$$

Damit ist auch $(A - \lambda B)$ regulär.

Beweis:
Offenbar gilt:

$$(\lambda B - A) = -(A - \lambda B) = (-I_n)(A - \lambda B)$$

Nach Satz 4.24 und Korollar 4.23 aus [8, S. 80] gilt:

$$\begin{aligned} \det(\lambda B - A) &= \det((-I_n)(A - \lambda B)) = \det(-I_n)\det(A - \lambda B) \\ &= \underbrace{(-1)^n}_{\neq 0} \det(A - \lambda B) \end{aligned}$$

Es folgt für ein $\lambda' \in \mathbb{C}$:

$$\det(\lambda' B - A) \neq 0 \Leftrightarrow \det(A - \lambda' B) \neq 0$$

Nach Anwendung der Definition 2 folgt Behauptung. □

Beweis Theorem 9: Nach Theorem 5 existieren unitäre Matrizen Q und $U \in \mathbb{C}^{n\times n}$, sodass nach entsprechender Multiplikation gilt:

$$A = QTU^H, \quad B = QSU^H$$

mit $T, S \in \mathbb{C}^{n\times n}$ obere Dreiecksmatrizen.

Es folgt:

$$\begin{aligned}(zB - A)^{-1}B =& (z\,QSU^H - QTU^H)^{-1}QSU^H = (Q(zS - T)U^H)^{-1}QSU^H \\ =& U(zS - T)^{-1}Q^H\,QSU^H = U(zS - T)^{-1}S\,U^{-1}\,.\end{aligned}$$

Man sieht, dass $(zB - A)^{-1}B$ und $(zS - T)^{-1}S$ ähnlich zueinander sind.
Da die Spur invariant unter Ähnlichkeitstransformation ist (vgl. [9, Exercise unter Example 1.3.5]), gilt

$$\operatorname{tr}((zB - A)^{-1}B) = \operatorname{tr}((zS - T)^{-1}S) = \sum_{i=1}^{n}((zS - T)^{-1}S)_{ii}\,. \tag{2.7}$$

Es sei

$$N := \{i = 1, \ldots, n : s_{ii} \neq 0\}\,.$$

Mit den Lemmata 6, 7 und 8 folgt für (2.7):

$$\begin{aligned}\operatorname{tr}((zB - A)^{-1}B) =& \sum_{i=1}^{n}((zS - T)^{-1})_{ii}\ S_{ii} = \sum_{i=1}^{n}((zS - T)_{ii})^{-1}\ S_{ii} \\ =& \sum_{i=1}^{n}\frac{s_{ii}}{z\,s_{ii} - t_{ii}}\,.\end{aligned}$$

Für $i \notin N$ gilt nach Definition von N:

$$\frac{s_{ii}}{z\,s_{ii} - t_{ii}} = \frac{0}{-t_{ii}} = 0\,.$$

Beweis:
Nach Voraussetzung ist der Matrix-Pencil regulär, daher folgt:

$$\begin{aligned}\text{Matrix-Pencil regulär} \Rightarrow& \exists\lambda' \in \mathbb{C} : \det(A - \lambda B) \neq 0 \Rightarrow \lambda' \notin \lambda(A, B) \\ \Rightarrow& \mathbb{C} \neq \lambda(A, B) \Rightarrow \nexists i \in \{1, \ldots, n\} : s_{ii} = t_{ii} = 0 \\ \Rightarrow& t_{ii} \neq 0\ \forall i \notin N\,,\end{aligned}$$

wobei in der ersten Implikation Definition 2 und in der vierten Implikation Theorem 5 verwendet wurde. □

Mit

$$\lambda_i = \frac{t_{ii}}{s_{ii}}, \quad i \in N$$

folgt

$$\begin{aligned}\operatorname{tr}((zB - A)^{-1}B) &= \sum_{i=1}^{n} \frac{s_{ii}}{z\, s_{ii} - t_{ii}} = \sum_{i \in N} \frac{s_{ii}}{s_{ii}(z - t_{ii}\, s_{ii}^{-1})} \\ &= \sum_{i \in N} \frac{1}{z - \lambda_i}\,.\end{aligned}$$

□

Bermerkung 11. Hier wurde nicht die originale Identität von Futamura gezeigt, da nicht bewiesen wurde, dass

$$s_{ii} \begin{cases} \neq 0 & : i \leq |N| \\ = 0 & : i > |N| \end{cases}.$$

Für diese Ausarbeitung spielt es aber keine Rolle, da man immer einen Matrix-Pencil (A, B) mit regulärer Matrix B verwenden wird. Nach Bemerkung 3 ist daher immer $N = \{1, \ldots, n\}$.

Nach Theorem 5 und der Definition von N gilt für einen regulären Matrix-Pencil:

$$\lambda(A, B) = \left\{ \frac{t_{ii}}{s_{ii}} : s_{ii} \neq 0 \right\} = \{\lambda_i : i \in N\}\,.$$

Somit sind die Eigenwerte eines regulären Matrix-Pencils genau die Polstellen der Funktion $\operatorname{tr}((zB - A)^{-1}B)$.

In Kapitel 4 wird daher das Residuentheorem auf ein Kurvenintegral über $\operatorname{tr}((zB - A)^{-1}B)$ angewendet, um die Anzahl der Eigenwerte zu bestimmen. Genau dies ist auch die Grundidee in [3] und [7].

Kapitel 3

Massen- und Steifigkeitsmatrizen

Um die Eigenkreisfrequenzen eines mechanischen Systems vorherzusagen, benötigt man die Massen- und Steifigkeitsmatrizen dieser Systeme. Diese Systeme werden nun vorgestellt.

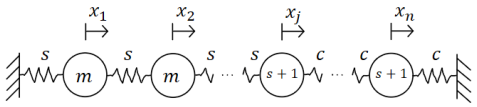

Abbildung 3.1: Schema System 1

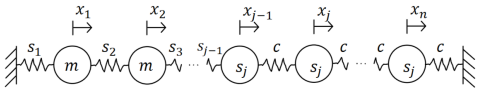

Abbildung 3.2: Schema System 2

Der Vektor $x = (x_1, \ldots, x_n)^T$ enthält hier die Auslenkungen der Massen aus ihrer Ruhelage und $s \in \mathbb{R}^l$ wird Design-Parameter genannt.

Falls in dieser Arbeit je von „Systemen“ die Rede ist, so sind damit die Systeme aus Abb. 3.1 und 3.2 gemeint.

Die Systeme aus Abb. 3.1 und 3.2 werden „Schwingerketten“ [1, S. 236] genannt, da sie aus Massen und Federn bestehen, die linear miteinander verbunden sind. Es existiert zudem in diesen Systemen keine Dämpfung.

Laut [1, S. 362-365] könne man für diese Systeme die „Differentialgleichung der freien Schwingungen“ [1, S. 365] erhalten, wenn man für jede Masse ein Kräftegleichgewicht aufstellt und geeignet umformt. Diese Differentialgleichung besitzt die Form

$$Kx + M\ddot{x} = 0, \tag{3.1}$$

wobei K die Steifigkeitsmatrix und M die Massenmatrix des Systems sind. Da genau diese Matrizen benötigt werden, um die Eigenkreisfrequenzen zu berechnen, ist das Ziel dieses Kapitels die Systeme von Kräftegleichgewichten aufzustellen und anschließend so umzuformen, dass eine Gleichung der Form (3.1) entsteht. Anhand dieser Gleichung werden die Matrizen bestimmt. Dafür wird der erste Abschnitt die im System wirkenden Kräfte und die Anwendung des Prinzips von d'Alembert behandeln. Im zweiten Abschnitt werden die Matrizen der Systeme bestimmt und anschließend wird in Abschnitt 3.3 die Formel für die Berechnung der Eigenkreisfrequenzen hergeleitet und umgeformt.

3.1 Herleitung durch das Prinzip von d'Alembert

Nach [10] besage das Prinzip von d'Alembert, dass die Summe aller wirkenden Kräfte in einem beschleunigten System verschwinde. Wir betrachten hier ein beschleunigtes System, da man in den Systemen die Massen von einem festen Standpunkt beobachtet. Daher gibt es für jede Masse eine Auslenkung aus der Ruhelage[1].

Somit wird zuerst ein Kräftegleichgewicht aufgestellt und anschließend in eine Gleichung über Matrizen umgeformt, wodurch man die Massen- und Steifigkeitsmatrix erhält. Da diese Herleitung auf Kräftegleichgewichten beruht, werden nun die agierenden Kräfte kurz erläutert:

Die Federkraft $\overrightarrow{F_L}$ wird laut [11] durch

$$\overrightarrow{F_L} = c \cdot s \tag{3.2}$$

berechnet, wobei c die Federkonstante und s die Auslenkung der Feder aus der Ruhelage ist.

Beachte, dass nach Definition die Federkraft entgegen der Auslenkung wirkt, da bei Auslenkung die Feder in die Ruhelage zurückkehren will.

Der Betrag F_T der Trägheitskraft ist nach [12] durch

$$F_T = m \cdot a \tag{3.3}$$

mit Masse m und Beschleunigungsvektor a definiert.

Beachte, dass laut [12] auch diese Kraft der Auslenkung entgegenwirkt. Für den Kraftvektor $\overrightarrow{F_T}$ gilt daher

$$\overrightarrow{F_T} = -m \cdot a\,.$$

Um die Darstellung der Kräfte klarer zu gestalten, werden die Kraftpfeile immer so gezeichnet, dass ihre Werte nichtnegativ sind. Das bedeutet, dass beide Kräfte entgegen der Auslenkung gezeichnet werden, dies kann man auch in Abb. 3.3 erkennen.

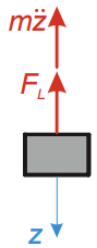

Abbildung 3.3: wirkende Kräfte an einer Masse mit Feder, Ausschnitt aus [13]

Laut [13] gelte zwar $F_L = c \cdot z$, aber auch hier werden F_L und die Auslenkung z in entgegengesetzte Richtungen eingezeichnet. Da in dieser Ausarbeitung nur Feder-Masse-Systeme behandelt werden, kann man s und a auf die Auslenkungen $(x_k(t))_k$

[1] man könnte auch ein System betrachten, in dem die Auslenkung einer bestimmten Masse immer gleich Null ist, in diesem System würden alle anderen Auslenkungen von dieser einen abhängen.

zurückführen: Im Allgemeinen wird hier s durch $(x_{k+1} - x_k)$ und a, ähnlich zu Abb. 3.3, durch die zweite Ableitung $\ddot{x_k}(t)$ beschrieben.

Betrachte nun ein System mit 3 Massen, wie es in Abb. 3.4 dargestellt wird. Anhand dieses Systems soll eine allgemeines Kräftegleichgewicht aufgestellt werden.

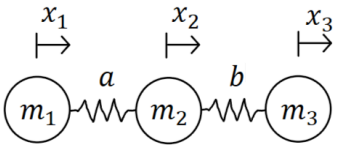

Abbildung 3.4: System mit 3 Massen

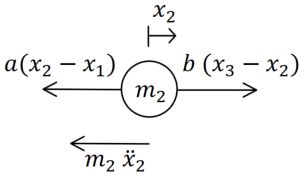

Abbildung 3.5: Kräfte an frei geschnittener Masse

Für m_2 erhält man die wirkenden Kräfte, wie sie in Abb. 3.5 dargestellt wurden. Ein ähnliches Schema findet man auch in [14] und Bild 4.6 d) aus [1, S. 237]. In letzterem sind einige Kraftpfeile entgegengesetzt eingezeichnet, dafür wurden aber die Beträge mit -1 multipliziert[2].

Mit dem Prinzip von d'Alembert führt dies auf die Gleichung

$$m_2\,\ddot{x}_2 + a\,(x_2 - x_1) - b\,(x_3 - x_2) = 0\,. \tag{3.4}$$

Nehme man an, es gelte $x_1 \equiv 0$, dann würde (3.4) sich zu

$$m_2\,\ddot{x}_2 + a\,x_2 - b\,(x_3 - x_2) = m_2\,\ddot{x}_2 + (a+b)\,x_2 - b\,x_3 = 0 \tag{3.5}$$

reduzieren.

In (3.5) kann die Masse m_1 als unbeweglicher Rand betrachtet werden. Eine Gleichung dieser Form wird daher in den Systemen für den Punkt $k = 1$ gelten. Analog kann mit $x_3 \equiv 0$ verfahren werden, um für die Systeme aus den Abbildungen 3.1 und 3.2 eine Gleichung für den Punkt $k = n$ zu erhalten.

3.2 Ermittlung der Matrizen der Systeme

Nun werden die Massen- und Steifigkeitsmatrizen der Systeme bestimmt.

Beginne dazu mit dem ersten System, siehe Abb. 3.1.

Die Form der Gleichung für die linke Masse ist durch (3.5) gegeben. Für $k = 2, \ldots, n-1$ gilt eine Gleichung der Form (3.4).

Da die Masse m_n durch die rechte Feder mit dem Rand verbunden ist, gilt also $x_{n+1} \equiv 0$. Und analog zu (3.5) erhält man

$$(s+1)\,\ddot{x}_n + c\,(x_n - x_{n-1}) + c\,x_n = 0$$

Nach Einsetzen aller Massen, Federkonstanten und Auslenkungen erhält man nach Umstellen das Gleichungssystem

[2] in diesem Bild aus [1] wurde folgender Fehler gemacht: Die linke Kraft wurde mit $c_k(x_{k-1} - x_k)$ bezeichnet, richtig wäre $c_{k-1}(x_{k-1} - x_k)$

$$\begin{cases} m\,\ddot{x}_1 + 2s\,x_1 - s\,x_2 & = 0 \\ m\,\ddot{x}_k - s\,x_{k-1} + 2s\,x_k - s\,x_{k+1} & = 0, \quad k = 2, \ldots, j-1 \\ (s+1)\,\ddot{x}_j - s\,x_{j-1} + (s+c)\,x_j - c\,x_{j+1} & = 0 \\ (s+1)\,\ddot{x}_k - c\,x_{k-1} + 2c\,x_k - c\,x_{k+1} & = 0, \quad k = j+1, \ldots, n-1 \\ (s+1)\,\ddot{x}_n - c\,x_{n-1} + 2c\,x_n & = 0 \end{cases}.$$

Definiere nun n und j explizit, damit auch M und K explizit beschrieben werden können:

$$n = 8, \quad j = n/2 = 4\,. \tag{3.6}$$

Dann kann dieses System in eine Gleichung der Form (3.1) zusammengefasst werden. Hierbei gilt

$$M = \operatorname{diag}(m, m, m, s+1, s+1, s+1, s+1, s+1) \text{ und} \tag{3.7}$$

$$K = \begin{pmatrix} 2s & -s & & & & & & \\ -s & 2s & -s & & & & 0 & \\ & -s & 2s & -s & & & & \\ & & -s & s+c & -c & & & \\ & & & -c & 2c & -c & & \\ & & & & -c & 2c & -c & \\ & 0 & & & & -c & 2c & -c \\ & & & & & & -c & 2c \end{pmatrix}. \tag{3.8}$$

Wende dieses Vorgehen auch für das System aus Abb. 3.2 an und erhalte

$$\begin{cases} m\,\ddot{x}_1 + (s_1+s_2)\,x_1 - s_2\,x_2 & = 0 \\ m\,\ddot{x}_k - s_k\,x_{k-1} + (s_k+s_{k+1})\,x_k - s_{k+1}\,x_{k+1} & = 0, \quad k = 2, \ldots, j-2 \\ s_j\,\ddot{x}_{j-1} - s_{j-1}\,x_{j-2} + (s_{j-1}+c)\,x_{j-1} - c\,x_{j+1} & = 0 \\ s_j\,\ddot{x}_k - c\,x_{k-1} + 2c\,x_k - c\,x_{k+1} & = 0, \quad k = j, \ldots, n-1 \\ s_j\,\ddot{x}_n - c\,x_{n-1} + 2c\,x_n & = 0 \end{cases}.$$

Dieses System kann man analog zu oben in eine Gleichung der Form (3.1) umformen. Hier gilt nun aber, mit n und j wie in (3.6)

$$M = \operatorname{diag}(m, m, s_4, s_4, s_4, s_4, s_4, s_4) \tag{3.9}$$

$$K = \begin{pmatrix} s_1+s_2 & -s_2 & & & & & & \\ -s_2 & s_2+s_3 & -s_3 & & & & 0 & \\ & -s_3 & s_3+c & -c & & & & \\ & & -c & 2c & -c & & & \\ & & & -c & 2c & -c & & \\ & & & & -c & 2c & -c & \\ & 0 & & & & -c & 2c & -c \\ & & & & & & -c & 2c \end{pmatrix}. \tag{3.10}$$

Wie man sieht, gibt es noch zwei unbekannte Parameter: m[3] und c. Beide seien positiv und werden in Kapitel 5 festgelegt.

Man beachte, dass für positive Parameter $m \in \mathbb{R}$ und s[4] beide Massenmatrizen regulär ist.

Damit auch s positiv ist, wähle man einen positiven Startwert und eine zulässige Menge $\Omega \subset (0, \infty]^l$. Dies wird ebenfalls später festgelegt.

3.3 Zusammenhang Eigenwert und Eigenkreisfrequenz

Nach der Ermittlung der zugehörigen Matrizen bestimme nun die Formel für die Eigenkreisfrequenzen eines Systems.

Die Schritte in diesem Abschnitt folgen [1, S. 380] und [3, S. 2]. Nach [1, S. 380] gelte für die Auslenkungen der Massen mit x statt q:

$$x(t) = v\, \exp(i\omega t), \quad \ddot{x}(t) = -\omega^2\, v \exp(i\omega t)$$

mit $v = (v_1, \ldots, v_n)^T$ Vektor der Amplituden und der „noch unbekannten Eigenkreisfrequenz ω“ [1, S. 380]. Dieser Ansatz wird in (3.1) eingesetzt und durch Division mit $\exp(i\omega t)$[5] folgt:

$$(K - \omega^2\, M)\, v = 0\,. \tag{3.11}$$

Die Gleichung kann ebenfalls in [1, S. 380] gefunden werden.

Man will diese Gleichung für $v \neq 0$ lösen, da man sonst eine triviale Lösung erhält, in der keine Masse schwingt.

Um die Theorie aus Kapitel 2 anzuwenden, so definiere wie in [3, S. 2]

$$\lambda_i := \omega_i^2, \quad i = 1, \ldots, n\,.$$

Es folgt das allgemeine Eigenwertproblem

$$(K - \lambda\, M)\, v = 0\,. \tag{3.12}$$

Man beachte, dass diese Gleichung genau die Form aus (2.1) besitzt. Man kann für die vorgestellten Systeme folgern, dass für N aus Kapitel 2.3

$$N = \{1, \ldots, n\}$$

gilt, da die Massenmatrizen aus (3.7) und (3.9) Diagonalmatrizen mit positiven Einträgen auf der Diagonalen sind. Die Eigenwerte der Massenmatrix sind daher ebenfalls positiv.

Für reguläre Massenmatrizen gelte nach [5, S. 376] für (3.12):

$$(K - \lambda\, M)\, v = 0 \Leftrightarrow (M^{-1}K - \lambda I_n)\, v = 0$$

[3] es gibt hier eine Doppelbelegung von m: einerseits ist es ein Parameter der Massenmatrix, andererseits gibt es die Anzahl an Teilintervallen in Kapitel 5 an

[4] mit $s \in \mathbb{R}^l > 0$ ist gemeint, dass $s_i > 0\ \forall i = 1, \ldots, l$

[5] offensichtlich ist die Division wohldefiniert, da $\exp(i\omega t) \neq 0\ \forall \omega, t \in \mathbb{R}$,

und folglich

$$\lambda(K, M) = \lambda(M^{-1}K, I_n) = \lambda(M^{-1}K)\,. \tag{3.13}$$

Somit sind die Eigenwerte des Matrix-Pencils $K - \lambda M$ und der Matrix $M^{-1}K$ gleich.

Da auch M^{-1} eine Diagonalmatrix ist, so ist $M^{-1}K$ für die vorgestellten Systeme symmetrisch. Es gilt daher für die Systeme aus Abb. 3.1 und 3.2

$$\lambda(K, M) = \lambda(M^{-1}K) \subset \mathbb{R}\,, \tag{3.14}$$

wobei [5, S. 393] angewendet wurde.

Durch Bemerkung 3 folgt ferner, dass die Matrix-Pencil der Systeme regulär sind.

Kapitel 4

Zählung von Eigenwerten

Nun wende man sich dem Finden einer Zielfunktion zu, die die Anzahl der Eigenwerte auf einem Intervall beschreibt. Diese gesuchte Funktion soll von dem Design-Parameter s abhängen und wird offenbar minimal, wenn kein Eigenwert des Matrix-Pencils in dem Intervall liegt.

4.1 Vorüberlegungen

Die in diesem Abschnitt erwähnten Schritte stammen aus [3, S. 2-4].

Betrachte Gleichung (3.12), durch die Definition $\lambda = \omega^2$ verändert sich auch das Intervall, welches man betrachtet. Das neue Intervall sei definiert als:

$$[\lambda_a, \lambda_b] := [\omega_a, \omega_b] .$$

Definiere nun die Funktion h wie folgt:

$$h(\lambda) := \mathbb{1}_{[\lambda_a, \lambda_b]}(\lambda) .$$

Da man erstmal daran interessiert ist, die Anzahl an Eigenwerten auf dem Intervall zu erfassen, definiere

$$\mu := \sum_{j \in N} h(\lambda_j) = \sum_{j=1}^{n} h(\lambda_j) ,$$

da nach Kapitel 3.3 $N = \{1, \ldots, n\}$ gilt.

Obwohl diese Funktion die Anzahl der Eigenwerte korrekt beschreibt, gibt es einige Schwachstellen, sollte man diese Funktion minimieren wollen: Die Funktion ist eine Treppenfunktion, damit ist sie nicht stetig und alle Ableitungen sind gleich 0, also unbrauchbar für Minimierungsalogrithmen, die eine Ableitung verwenden.

Diese Probleme kann man teilweise umgehen, indem man die Funktion h auf dem Intervall $[\lambda_a, \lambda_b]$ gewichtet.

Um die Eigenwerte effektiv aus dem Intervall zu entfernen, nutze eine Gewichtungsfunktion, die auf dem ganzen Intervall positiv und konkav ist (vgl. [3, S. 3]). Diese Funktion sorgt dafür, dass Eigenwerte in der Mitte des Intervalls stärker ins

Gewicht fallen und somit die Eigenwerte zuerst aus der Mitte des Intervalls zum Rand gehen und ihn anschließend überqueren. Nach [3, S. 3] sei die Funktion

$$g(z) = -(z - ((1+\alpha)\lambda_a - \alpha\lambda_b))(z - ((1+\alpha)\lambda_b - \alpha\lambda_a))$$

sehr geeignet, man könnte aber auch andere Funktionen definieren, die diese Eigenschaften besitzen. Hierbei ist $\alpha > 0$ einen Parameter, der sicherstellt, dass die Funktion $g(z)$ auf $[\lambda_a, \lambda_b]$ positiv ist. Der Parameter α wird „Inflationsparameter“ [3, S. 3][1] genannt.

Man findet so die erste Funktion, welche man mit Verfahren der Optimierung sinnvoll minimieren könnte:

$$J(s) = \sum_{j=1}^{n} g(\lambda_j) h(\lambda_j)\,. \tag{4.1}$$

Für diese Formel benötigt man allerdings alle Eigenwerte, diese müssten vorher berechnet werden. Auch hängen weder g noch h explizit von s ab, daher kann die Funktion $J(s)$ in dieser Form zwar zur gewichteten Zählung der Eigenwerte benutzt werden, aber nur über Umwege kann man $\nabla J(s)$ berechnen, was nötig wäre, um die gewichtete Eigenwertzählung schnellstmöglich zu minimieren.

Das Ziel ist daher eine Funktion, die explizit von s abhängt, aber trotzdem die Eigenwerte gewichtet zählt. Dazu wird die Identität von Futamura aus Theorem 9 benutzt.

Betrachte also wie in Theorem 9 eine Funktion, deren Polstellen die Eigenwerte sind.

Die Funktion sei durch

$$L(z,s) = g(z) \sum_{j \in N} \frac{1}{z - \lambda_j} = \sum_{j=1}^{n} \frac{g(z)}{z - \lambda_j} \tag{4.2}$$

definiert, ein Kurvenintegral über diese Funktion sei

$$\frac{1}{2\pi i} \int_\gamma L(z,s)\; dz\,. \tag{4.3}$$

Um den Residuensatz sinnvoll anwenden zu können, muss die Kurve γ das Intervall $[\lambda_a, \lambda_b]$ einschließen. Da aber alle Eigenwerte reell sind, so ist für γ ein Kreis, welcher durch die Punkte λ_a und λ_b geht, gut geeignet[2].

Allerdings muss zunächst gezeigt werden, dass diese Gleichung tatsächlich die Eigenwerte in dem Intervall gewichtet zählt, dafür benötigt man den Residuensatz, welcher im nächsten Abschnitt vorgestellt und angewendet wird.

[1] Übersetzung des Autors

[2] dadurch wird sichergestellt, dass kein Eigenwert, der außerhalb des Intervalls $[\lambda_a, \lambda_b]$ liegt, mit in die Zählung eingeht

4.2 Der Residuensatz und dessen Anwendung

Dieses Kapitel behandelt ausschließlich den Residuensatz, die Argumente, warum er auf (4.3) anwendbar ist und die Anwendung selbst. Obwohl L durch die Verteilung der Eigenwerte auch von dem Design-Parameter s abhängt, vernachlässige diese Abhängigkeit in diesem Abschnitt.

Theorem 12. *(Theorem 9.2 (Residue theorem), [15, S. 141])*
Sei f analytisch auf Ω, mit Ausnahme der isolierten Singularitäten $a_1, \ldots, a_n$. Wenn für einen Kreis $\gamma \subseteq \mathbb{C}$ die Bedingungen $\gamma \sim 0$ und $a_j \notin \gamma,\ j = 1, \ldots, n$ erfüllt sind, dann

$$\int_\gamma f(z)\ dz = 2\pi i \sum_{k=1}^{n} n(\gamma, a_k) Res_{a_k} f\,.$$

Beweis: siehe [15, S. 142], □

Die folgenden Aussagen erklären die Voraussetzungen und das Resultat des Theorems und stammen aus [15]. Die Aussagen und Definitionen werden nur kurz angeschnitten, da man sie in dieser Ausarbeitung nur für den Residuensatz (Theorem 12) und Gleichung (4.3) verwendet. Die Beweise sind ebenfalls in [15] wiederzufinden.

- **analytisch in z** : Ist eine Eigenschaft von komplexen Funktionen, die besagt, dass man die Funktion als eine Potenzreihe um z entwickeln kann.

  Für $L(z, s)$ reicht aber folgende Aussage aus [15, S. 24]: Für Polynome $p(z)$ und $q(z)$ sei die rationale Funktion $L(z) := \frac{p(z)}{q(z)}$ analytisch auf $\{z \in \mathbb{C} : q(z) \neq 0\}$.

  Offenbar ist $L(z)$ eine rationale Funktion

  $$L(z) = \frac{p(z)}{q(z)}$$

  mit

  $$p(z) = \sum_{j=1}^{n} \prod_{\substack{i=1 \\ i \neq j}}^{n} z - \lambda_i \text{ und}$$

  $$q(z) = \prod_{i=1}^{n} z - \lambda_i\,.$$

  Offenbar sind p und q Polynome und es gilt $\{z : q(z) \neq 0\} = \mathbb{C} \setminus \{\lambda_i, i = 1, \ldots, n\}$. Daher ist L analytisch auf $\mathbb{C}$ bis auf $\lambda_1, \ldots, \lambda_n$.

- **b isolierte Singularität**: Laut [15, S. 74] habe L eine isolierte Singularität bei λ_i, wenn $L(\lambda_i)$ nicht definiert und L analytisch für $0 < |z - \lambda_i| < \epsilon$ mit $\epsilon > 0$ sei.

  Offenbar ist $L(\lambda_i)$ nicht definiert für $i = 1, \ldots, n$ und da L nur eine endliche Anzahl an Singularitäten besitzt, gilt:

  $$L \text{ analytisch für } 0 < |z - \lambda_i| < \widetilde{\epsilon}, \quad i = 1, \ldots, n \tag{4.4}$$

mit

$$\widetilde{\epsilon} := \frac{1}{2} \min_{\substack{i,j \in \{1,\dots,n\} \\ \lambda_i \neq \lambda_j}} |\lambda_i - \lambda_j| \,.$$

Sollte ein einzelner Eigenwert mehrfach vorkommt, so ist dieser Eigenwert trotzdem eine isolierte Singularität, da die Definition dies erlaubt.

- $\frac{\mathbf{1}}{\mathbf{2\pi i}} \int_\gamma \frac{\mathbf{1}}{\mathbf{z-a}}\, \mathbf{dz}$: Dieses Integral ist zwar nicht direkt für Theorem 12 von Bedeutung, allerdings wird es hier mehrmals benötigt.

  Sei $z_0 \in \mathbb{C},\ r > 0$ und $\gamma = \{z_0 + re^{it} : t \in [0, 2\pi]\}$ (vgl. [15, S. 48]), laut Proposition 4.13 aus [15, S. 48] gelte

$$\frac{1}{2\pi i} \int_\gamma \frac{1}{z-a}\, dz = \begin{cases} 1 & : |a - z_0| < r \\ 0 & : |a - z_0| > r \end{cases} . \tag{4.5}$$

- $\mathbf{n}(\gamma, \mathbf{a})$: Sei γ ein Kreis und $a \in \mathbb{C} \setminus \gamma$. Nach Definition 5.4 aus [15, S. 65] ist die Windungsnummer $n(\gamma, a)$ von γ über a wie folgt definiert:

$$n(\gamma, a) := \frac{1}{2\pi i} \int_\gamma \frac{1}{\xi - a}\, d\xi \,.$$

  Man erkenne, dass mit (4.5) gilt:

$$n(\gamma, a) = \begin{cases} 1 & : a \in B_r(z_0) \\ 0 & : a \notin \overline{B_r(z_0)} \end{cases} . \tag{4.6}$$

  Hierbei benötigt man das γ wie in (4.5), damit der Kreis in positive Richtung durchlaufen wird und kein Punkt (bis auf $z = z_0 + r$) mehrmals vorkommt.

- $\gamma \sim \mathbf{0}$: Nach Definition 5.5 aus [15, S. 67] folgt

$$\gamma \sim 0 \Leftrightarrow n(\gamma, a) = 0 \ \forall a \notin \mathbb{C} \,.$$

  Offenbar gilt für alle $\alpha \in \mathbb{C}^C$

$$n(\gamma, \alpha) = 0 \,.$$

- $\mathrm{Res}_{\mathbf{a}}\mathbf{f}$: Nach Beispiel 9.3 aus [15, S. 142] gelte für eine einfache Polstelle[3]:

$$\mathrm{Res}_a f = \lim_{z \to a} (z - a) f(z) \,. \tag{4.7}$$

  Anwenden auf $\mathrm{Res}_{\lambda_i} L$ ergibt:

$$\mathrm{Res}_{\lambda_i} L = \lim_{z \to \lambda_i} \sum_{j=1}^{n} (z - \lambda_i) \frac{g(z)}{z - \lambda_j} = g(z) \underbrace{\sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{z - \lambda_i}{z - \lambda_j}}_{\to 0} + g(z) \frac{z - \lambda_i}{z - \lambda_i}$$

$$= g(\lambda_i) \,.$$

[3] es gibt hier nur einfache Polstellen, da nach (4.2) kein Term der Form $\frac{1}{(z-\nu)^r},\ r > 1$ vorkommen kann

Damit wurde gezeigt, dass der Residuensatz auf die Funktion L angewendbar ist.

Mit dem Residuensatz (Theorem 12) gilt folglich für (4.3)

$$\frac{1}{2\pi i}\int_\gamma L(z,s)\ dz = \sum_{k=1}^{n} n(\gamma,\lambda_k)\mathrm{Res}_{\lambda_k} L = \sum_{k=1}^{n} n(\gamma,\lambda_k) g(\lambda_k)$$

mit

$$\begin{aligned}\gamma =& \{\tilde{z_0} + \tilde{r}\exp(it) : t \in [0,2\pi]\}, \\ \tilde{z_0} =& \frac{\lambda_a+\lambda_b}{2}, \quad \tilde{r} = \frac{\lambda_b-\lambda_a}{2}\,.\end{aligned} \tag{4.8}$$

Mit (4.6) folgt für $n(\gamma,\lambda_k)$

$$n(\gamma,\lambda_k) = \begin{cases} 1 & : \lambda_k \in B_{\tilde{r}}(\tilde{z_0}) \\ 0 & : \lambda_k \notin \overline{B_{\tilde{r}}(\tilde{z_0})} \end{cases}$$

Da aber $\lambda_k \in \mathbb{R}$ für $k = 1,\dots,n$, ist diese Gleichung äquivalent zu

$$n(\gamma,\lambda_k) = \begin{cases} 1 & : \lambda_k \in (\lambda_a,\lambda_b) \\ 0 & : \lambda_k \notin [\lambda_a,\lambda_b] \end{cases} = \mathbb{1}_{[\lambda_a,\lambda_b]}(\lambda_k) = h(\lambda_k)\,.$$

Beachte, dass für den Fall $\lambda_k \in \{\lambda_a,\lambda_b\}$ diese Theorie und auch der Residuensatz aus Theorem 12 keine Aussage trifft, dieser Fall tritt aber fast sicher nicht ein.

Für (4.3) folgt

$$\frac{1}{2\pi i}\int_\gamma L(z,s)\ dz = \sum_{k=1}^{n} h(\lambda_k) g(\lambda_k) = J(s)\,,$$

womit gezeigt wurde, dass tatsächlich (4.3) die Eigenwerte auf dem Intervall gewichtet zählt.

4.3 Anwendung der Identität von Futamura und Differenzierung

Um nun die Abhängigkeit von s zu offenbaren, nutze die Identität von Futamura (Theorem 9), wodurch folgt

$$\begin{aligned} J(s) =& \frac{1}{2\pi i}\int_\gamma L(z,s)\ dz = \frac{1}{2\pi i}\int_\gamma g(z)\sum_{j\in N}\frac{1}{z-\lambda_j}\ dz \\ =& \frac{1}{2\pi i}\int_\gamma g(z)\mathrm{tr}((zM(s)-K(s))^{-1}M(s))\ dz\,. \end{aligned} \tag{4.9}$$

Diese Funktion hängt nun explizit von s ab und kann daher in Kapitel 5 minimiert werden.

Gehe zuletzt auf $\frac{\partial L(z,s)}{\partial s}$ ein, da diese Ableitung in Kapitel 5 benötigt wird:

$$\frac{\partial L(z,s)}{\partial s} = g(z)\frac{\partial}{\partial s}\left(\operatorname{tr}(\underbrace{(zM-K)^{-1}M}_{=:B(z,s)})\right) \tag{4.10}$$

und es gilt für $\frac{\partial}{\partial s}\left(\operatorname{tr}(B(z,s))\right)$

$$\frac{\partial}{\partial s}\left(\operatorname{tr}(B(z,s))\right) = \frac{\partial}{\partial s}\left(\sum_{i=1}^{n}(B(z,s))_{ii}\right) = \sum_{i=1}^{n}\frac{\partial}{\partial s}(B(z,s))_{ii} = \operatorname{tr}\left(\frac{\partial}{\partial s}B(z,s)\right).$$

Ferner gilt für die Ableitung von $B(z,s)$

$$\begin{aligned}\frac{\partial}{\partial s}B(z,s) =& \frac{\partial D}{\partial s}\ M + D\ \frac{dM}{ds}\\ =& -D\frac{d}{ds}(zM-K)D + D\ \frac{dM}{ds}\\ =& D\ \frac{dM}{ds} - D\left(z\frac{dM}{ds} - \frac{dK}{ds}\right)D\,.\end{aligned}$$

Wobei der Lesbarkeit halber $D := (zM-K)^{-1}$ gilt und im 2. Schritt die Formel

$$\nabla A(t)^{-1} = -A(t)^{-1}\ \nabla A(t)\ A(t)^{-1}$$

angewendet wurde.
Beweis: siehe [16]. □

Die endgültige Formel für die Ableitung von L ist

$$\frac{\partial L(z,s)}{\partial s} = g(z)\ \left[\operatorname{tr}\left(D\ \frac{dM}{ds}\right) - \operatorname{tr}\left(D\left(z\frac{dM}{ds} - \frac{dK}{ds}\right)D\right)\right]. \tag{4.11}$$

Kapitel 5

Implementierung

Man wende sich nun der Implementierung des Minimierungsverfahrens zu. Wie schon in (3.6) erwähnt gilt für beide Systeme aus Kapitel 3

$$n = 8, \quad j = 4\,.$$

Es gibt aber noch mehr Parameter, die zuerst definiert werden müssen:

- für System 1 aus Abb. 3.1 gelte
  - $\lambda_a = 1.5$,
  - $\lambda_b = 2.5$,
  - $s_0 = 3.5 \in \mathbb{R}$,
  - $\Omega = [2, 5] \subset \mathbb{R}$,
  - $m = 4$ und
  - $c = 1.5$
- für System 2 aus Abb. 3.2 gelte
  - $\lambda_a = 0.9$,
  - $\lambda_b = 1.5$,
  - $s_0 = (0.7, 0.7, 0.7, 1.5)^T \in \mathbb{R}^4$,
  - $\Omega = [1, 2]^3 \times [0.5, 3.5] \subset \mathbb{R}^4$,
  - $m = 2$ und
  - $c = 0.75$

Hierbei ist Ω die zulässige Menge des Design-Parameters s und s_0 der Startwert der Minimierung. Die Parameter m und c werden für die Massen- und Steifigkeitsmatrizen (3.7) - (3.10) verwendet.

Es folgt zunächst noch der verwendete Algorithmus zur Minimierung und die Anwendung einer Quadraturformel zur Approximation des Integrals aus (4.9).

Algorithmus 1 Verfahren des steilsten Abstiegs (vgl. [17, S. 285])

Eingabe: $J(s), \nabla J(s)$
Ausgabe: s^* mit $J(s^*) = 0$
  wähle $s \in \mathbb{R}^n$;
  repeat
    $d := \nabla J(s)$;
    $s := s - \lambda_* d$;
  until $J(s) = 0$;
  $s^* := s$;
  return s^*;

5.1 Theorie

Um die Funktion (4.9) zu minimieren, verwende eine Variation des Gradientenverfahrens aus [17, S. 285], diese sei in Algorithmus 1 beschrieben.

Die Schrittweite λ_* sei hier fest und da das Minimum der gewichteten Eigenwertzählung bekannt ist, benötigt man nur Information über $J(s)$, um ein Minimum zu erkennen. Diese Variation der Abbruchbedingung sorgt für eine schnellstmögliche Beendung des Minimierungsverfahrens.

Allerdings werden für einen Schritt des Verfahrens $\nabla J(s)$ und $J(s)$ benötigt. Durch (3.13) können die Eigenwerte des Matrix-Pencils $K - \lambda M$ einfach bestimmt werden, wodurch die Berechnung von $J(s)$ durch Gleichung (4.1) trivial wird. Aber für $\nabla J(s)$ müsste man die Funktion $J(s)$ aus (4.9) ableiten. Um dieses Parameterintegral zu umgehen, approximiere $J(s)$ zunächst durch eine Quadraturformel und differenziere dann diese Approximation $J^*(s)$.

Teile dazu $\gamma = \{\tilde{z_0} + \tilde{r}\exp(it) : t \in [0, 2\pi]\}$ in m Teile auf:

$$\gamma_k = \left\{ \tilde{z_0} + \tilde{r}\exp(it) : t \in \left[2\pi\frac{k}{m}, 2\pi\frac{k+1}{m} \right] \right\}, \quad k = 0, \ldots, m-1\,. \tag{5.1}$$

Es folgt

$$\begin{aligned} \int_{\gamma_k} f(z)\, dz &= \int_{2\pi\frac{k}{m}}^{2\pi\frac{k+1}{m}} f(\tilde{z_0} + \tilde{r}\exp(it))\, i\tilde{r}\exp(it)\, dz \\ &\approx \frac{f(z_k)\, i\tilde{r}\exp(\frac{2\pi ik}{m}) + f(z_{k+1})\, i\tilde{r}\exp(\frac{2\pi i(k+1)}{m})}{2}\frac{2\pi}{m} \\ &= \left(f(z_k) + f(z_{k+1})\, \exp\left(\frac{2\pi i}{m}\right) \right) \frac{\pi i\tilde{r}}{m}\, \exp(\frac{2\pi ik}{m})\,, \end{aligned} \tag{5.2}$$

wobei $z_k := \tilde{z_0} + \tilde{r}\exp\left(\frac{2\pi ik}{m}\right), \quad k = 0, \ldots, m$. Es wurde hier eine Transformation zur Berechnung von Kurvenintegralen verwendet, siehe [18, S. 21]. Ferner wurde in der zweiten Zeile die Trapezregel angewendet, wie sie in [19, S. 498] beschrieben ist.

Es gilt daher

$$\begin{aligned} J(s) =& \frac{1}{2\pi i} \sum_{k=0}^{m-1} \int_{\gamma_k} L(z,s)\ dz \\ \approx& \frac{1}{2\pi i} \left[\frac{\pi i \tilde{r}}{m} \sum_{k=0}^{m-1} \left(L(z_k,s) + L(z_{k+1},s)\ \exp\left(\frac{2\pi i}{m}\right)\right) \exp\left(\frac{2\pi i k}{m}\right)\right] \\ =& : J^*_{\mathrm{Tr}}(s)\,. \end{aligned} \tag{5.3}$$

$J^*(s)$ steht hierbei für die approximierte gewichtete Eigenwertzählung durch eine nicht näher bestimmte Quadraturformel. Der Index „Tr“ steht hier für die verwendete Trapezregel. Man kann leicht sehen, dass mit der Summen- und Faktorregel für Ableitungen

$$\nabla J^*_{\mathrm{Tr}}(s) = \frac{1}{2\pi i} \left[\frac{\pi i \tilde{r}}{m} \sum_{k=0}^{m-1} \left(\partial_s L(z_k,s) + \partial_s L(z_{k+1},s)\ \exp\left(\frac{2\pi i}{m}\right)\right) \exp\left(\frac{2\pi i k}{m}\right)\right]$$

gilt.

Die Faktoren vor $J^*(s)$ und $\nabla J^*(s)$ werden nicht zusammengefasst, da die eckige Klammer die Quadraturformel beschreibt, der Faktor $\frac{1}{2\pi i}$ gehört aber nicht zur Quadraturformel, sondern ist wegen der Anwendung des Residuensatzes vorhanden. In der Implementierung wird nur eine Funktion „JStern“ definiert, diese kann aber jede beliebige Quadraturformel nutzen, um $J(s)$ zu approximieren.

Die Funktion $\partial_s L(z,s)$ wurde schon in Kapitel 4 in Gleichung (4.10) definiert.

Die Ableitungen $\frac{dM}{ds}$ und $\frac{dK}{ds}$ werden durch eine Differenz, wie sie in [20, S. 16 f.] vorgestellt wird, approximiert.

Wähle hier die Vorwärtsdifferenz

$$(D^+u)(x) := [u(x+h) - u(x)]/h \tag{5.4}$$

mit $h = \frac{1}{10}$.

Da im Allgemeinen $s \in \mathbb{R}^l$, werden hier aber partielle Ableitungen approximiert. Für die i-te partielle Ableitung gelte

$$\frac{\partial M}{\partial s_i} \approx \frac{M(s + he_i) - M(s)}{h}, \quad i = 1, \ldots, l\,,$$

wobei e_i der i-te Einheitsvektor ist.

5.2 Implementation

Die folgenden Programme wurden mit Python geschrieben. Hauptsächlich wird das Paket „numpy“ genutzt, welches Berechnungen von Matrizen sehr vereinfacht.

Die verschiedenen übergreifenden Funktionen, die von allen Programmen verwendet werden, findet man in [21, *./Funktionen.py*].

Da die restlichen Dateien aus [21] nur eine Funktion aus *./Funktionen.py* aufrufen, werden kurz die verschiedenen Funktionen dieser Datei erwähnt.

Durch die Funktion *init()* können verschiedene Parameter definiert werden. Darunter fallen unter anderem die Anzahl an Freiheitsgraden n, die Anzahl an Teilintervallen m und der Inflationsparameter α.

systemAuswaehlen() wird verwendet, um zu entscheiden, ob System 1 aus Abb. 3.1 oder System 2 verwendet wird. Abhängig davon werden gemäß Kapitel 5 Variablen wie λ_a, λ_b und s_0 definiert.

Die vier Funktionen, die mit *quadratureContourIntegralCircle* beginnen, sind Quadraturformeln für einen Kurvenintegral.

ableitungDurchDifferenz() approximiert die Ableitung einer Funktion f an einer Stelle x durch ein Differenzenverfahren, wie es in (5.4) definiert ist. Für $\frac{dM}{ds}$ benötigt man aber l partielle Ableitungen, weshalb man die l partiellen Ableitungen einzeln berechnet und zu einer Matrix vereint. Eine partielle Ableitung besitzt als Array die Form $(n \times n)$. Um alle partiellen Ableitungen zu einem Array zusammenzuführen, werden dieses Arrays auf die Form $(n \times n \times 1)$ erweitert, um nun alle partiellen Ableitungen in der dritten Ebene zusammenzufügen (vgl. [22]). Das Ergebnis besitzt daher die Form $(n \times n \times l)$.

Die Funktion *schrittGradientenverfahren()* berechnet den Punkt s_{k+1} aus s_k, wohingegen *bedingungenPruefen()* prüft, ob ein Wert innerhalb der zulässigen Menge Ω liegt. Sollte dies nicht der Fall sein, wird der Design-Parameter mithilfe einer Projektion wie in [23, S. 314] wieder auf Ω abgebildet und im nächsten Schritt wird statt dem berechneten Wert die Projektion verwendet.

Die Funktion *EigenwerteMinimierenAufIntervall()* definiert wichtige Funktionen, wie *J(), J_Stern() und nablaJ_Stern()*, um dann wiederholt *schrittGradientenverfahren()* aufzurufen und somit die Eigenwertzählungen zu minimieren. Für die Ausführbarkeit von *nablaJ_Stern()* wird die Funktion „numpy.transpose“ [24] benötigt, damit die Multiplikation der Matrizen wie in 4.11 durchgeführt werden kann.

Schließlich wird *minimierenPlottenUndEckdatenAnzeigen()* genutzt, um vorherige Funktion aufzurufen, die benötigte Zeit zu stoppen, die Plots anzuzeigen und einige Kennzahlen des Durchlaufs auszugeben.

In Kapitel 5 werden ausschließlich die Ergebnisse des Programms [21, *./erste Implementierung.py*] behandelt. In dieser Datei werden sechs Durchläufe des Minimierungsverfahrens durchgeführt, beschrieben und die Plots gezeigt. Verbesserungen aus Kapitel 6 werden in den anderen Dateien zu finden sein.

Bei den Programmen gibt es eine Begrenzung für die Anzahl an Schritten pro Durchlauf der Minimierung. Falls ein Durchlauf mehr als 500 Schritte benötigen sollte, um zu einem Ergebnis zu kommen, wird er abgebrochen, damit ein einzelner Durchlauf nicht zu viel Zeit beansprucht. Diese Begrenzung kann man umgehen, indem man in Zeile 15 in *./erste Implementierung.py* den Wert des Arguments „maxIter“ beliebig verändert. Man beachte aber, dass für nicht-positive Werte keine Begrenzung vorhanden ist. Das Verfahren wird in diesem Fall solange ausgeführt, bis es zu einem Ergebnis kommt, oder manuell abgebrochen wird. Diese Sperre wurde auf 10000 erhöht, um die Ergebnisse in Tabelle 5.1 zu erhalten.

Es folgt nun eine Auflistung der Kennzahlen der 6 Durchläufe, die in diesem Programm durchgeführt werden.

Es gehören immer 2 nacheinander folgende Durchläufe zusammen, da sie sich

nur in der Anzahl der verwendeten Teilintervalle m und somit der Genauigkeit der Quadraturformel unterscheiden.

Beim Ausführen des Programms werden noch verschiedene andere Kennzahlen, wie der Endpunkt des Design-Parameters, ausgegeben, hier werden aber nur die Anzahl an benötigten Iterationen und die dafür benötigte Zeit betrachtet. Die Spalte „Zeit in s“ gibt nur an, wie viel Zeit der Aufruf der Funktion *EigenwerteMinimierenAufIntervall()* benötigte. Aufarbeitungen, wie das anschließende Berechnen der Eigenwerte, werden dabei nicht gemessen.

System	m	λ_*	Iterationen	Zeit in s
1	100	0.05	> 10000	> 584
1	150	0.05	> 10000	> 830
1	100	0.5	> 10000	> 561
1	150	0.5	15	1.2
2	100	0.05	150	15.2
2	150	0.05	> 10000	> 1592

Tabelle 5.1: Kennzahlen der ersten Implementierung

Man kann schon an Tabelle 5.1 erkennen, dass dieses Verfahren nur selten zum Erfolg führt.

Außerdem sind Zeiten über 9 Minuten für ein System mit nur 8 Freiheitsgraden und einem skalarem Design-Parameter völlig unzureichend.

Man bemerke aber, dass dieses Verfahren funktionieren kann, wie man an den Durchläufen 4 und 5 sieht.

5.3 Grafiken und Erklärung

Man gehe nun genauer auf die Verläufe der Eigenwertzählung und die Verläufe der Eigenwerte selbst ein. Betrachte zuerst die ersten beiden Durchläufe. Um die folgenden Ausführungen zu unterstützen, werden die folgenden zwei Abbildungen auf die Schritte 0 bis 200 begrenzt. Nach dem Schritt verändern sich in diesen beiden Durchläufen bis Schritt 10000 weder die Eigenwert-Zählungen noch die Verteilung der Eigenwerte wesentlich.

Der obere Plot eines Durchlaufs zeigt, wie sich die gewichteten und die ungewichtete Eigenwertzählung veränderten. Der untere Plot zeigt jeweils das vorgegebene Intervall und die Verteilung der Eigenwerte. Oft werden nur Ausschnitte gezeigt, um die Eigenwerte innerhalb des Intervalls deutlicher zu erkennen. Es ist daher möglich, dass nicht alle Eigenwerte abgebildet werden. Die kompletten Abbildungen kann man aber durch Ausführen des Programms *./erste Implementierung.py* erhalten.

Für diese beiden Durchläufe beschränke man sich auf die Verteilung der Eigenwerte, also den unteren Plot.

Die gepunkteten Linien stellen immer die Grenzen des Intervalls $[\lambda_a, \lambda_b]$ dar. Ferner werden die Eigenwerte in den Plots „Entwicklung der Eigenwerte“ nicht mithilfe

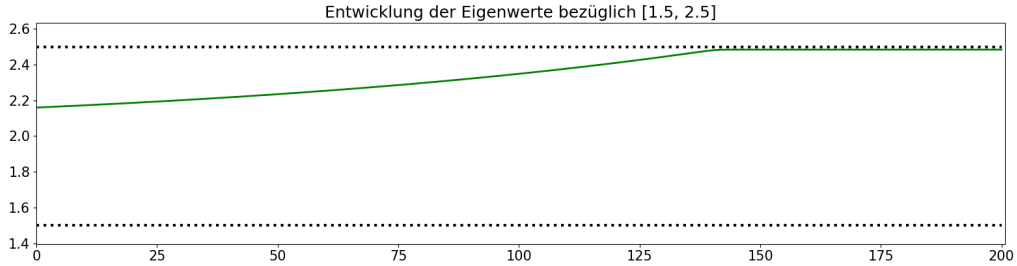

Abbildung 5.1: System 1, $m = 100$, $\lambda_* = 0.05$

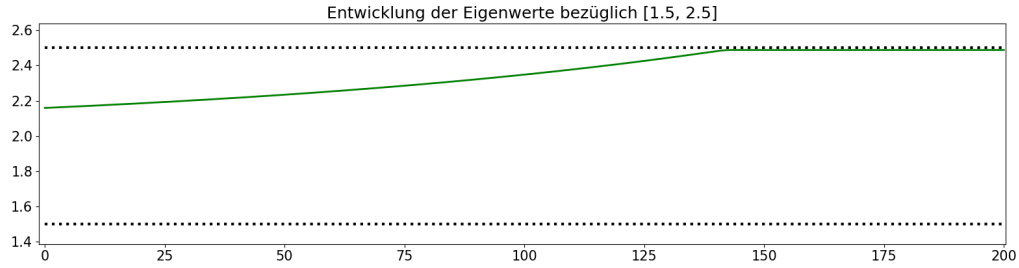

Abbildung 5.2: System 1, $m = 150$, $\lambda_* = 0.05$

einer Legende beschriftet, da dies keinen Mehrwert bringt. Mit „erster Eigenwert“ werde hier immer der größte Eigenwert bezeichnet, den man sehen kann, mit „zweiter Eigenwert“ der nächstkleinere. Man sehe zuerst in Abb. 5.1 und 5.2, dass sich der betrachtete Eigenwert dem oberen Rand des Intervalls (λ_b) nähert, aber ihn nicht überschreitet. Daran ändert sich auch nach weiteren 9800 Schritten nichts, wie man anhand von Tabelle 5.1 vermuten kann. Diese Durchläufe unterscheiden sich so minimal voneinander, dass die Unterschiede nur durch sehr starke Vergrößerung sichtbar werden. Dies kann man sich ebenfalls durch Ausführung des Programms klar machen.

Bei den nächsten beiden Durchläufen betrachte nur die Schritte 0 bis 17.

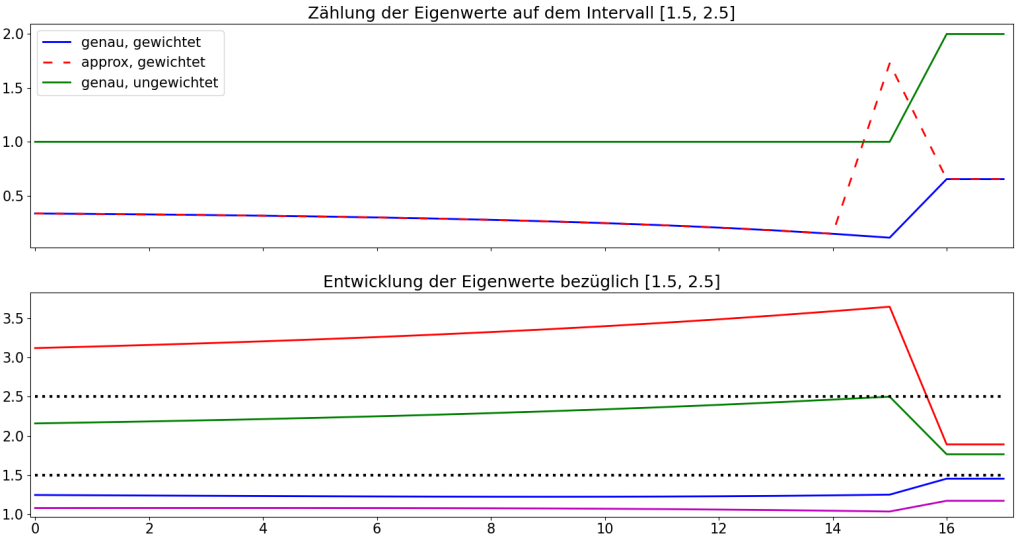

Abbildung 5.3: System 1, $m = 100$, $\lambda_* = 0.5$

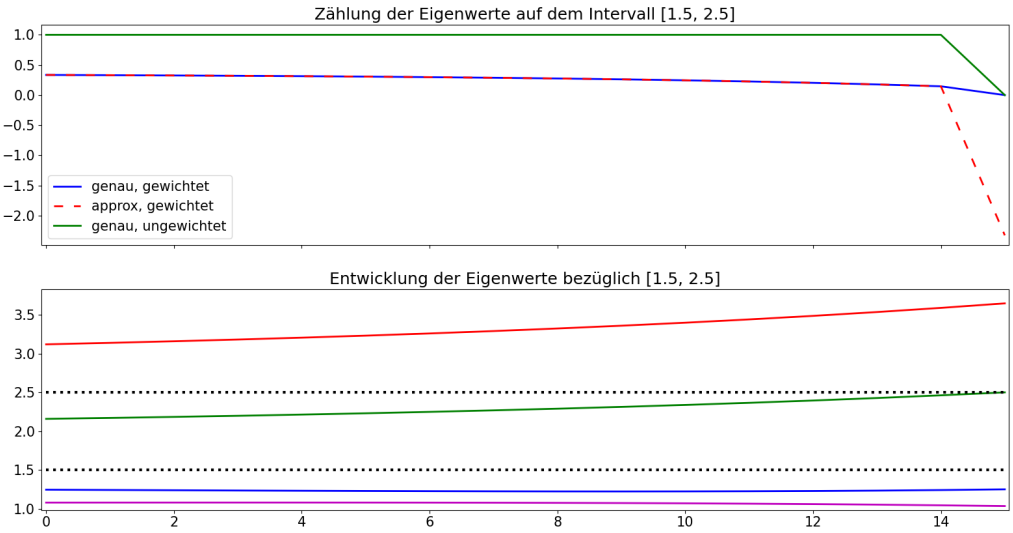

Abbildung 5.4: System 1, $m = 100$, $\lambda_* = 0.5$

In Abb. 5.3 und 5.4 kann man die Unterschiede zwischen den Genauigkeiten der Quadraturformeln leicht erkennen: während in Abb. 5.3 nach Schritt 15 nicht alle Eigenwerte aus dem Intervall entfernt wurden, so befinden sich in Abb. 5.4 nach dem gleichen Schritt alle Eigenwerte außerhalb des Intervalls. Da die Abbruchbedingung des Gradientenverfahrens hier die genaue gewichtete Eigenwertzählung ist, wird das Gradientenverfahren nach diesem Schritt beendet.

Man sehe aber auch, dass die approximierte gewichtete Eigenwertzählung bei beiden Durchläufen in Schritt 15 sehr stark von der genauen abweicht. Dies kann sehr wahrscheinlich auf die verwendete Quadraturformel zurückgeführt werden: Bei der Approximation $J^*_{\mathrm{Tr}}(s)$ wird $L(z, s)$ an den Stellen $z = \lambda_a$ und $z = \lambda_b$ ausgewertet. Falls aber ein Eigenwert sich nahe des Intervallrands befindet, so befindet sich eine Stützstelle nahe einer Polstelle, siehe Gleichung (4.2). Das führt zu unerwünschten Erscheinungen:

In Abb. 5.3 wird die gewichtete Eigenwertzählung bei Schritt 15 viel zu hoch approximiert, in Abb. 5.4 wird sie bei Schritt 15 negativ.

Bei Durchlauf 5.3 wird infolge dessen nicht mehr versucht, den zweiten Eigenwert zu vergrößern, damit er das obere Ende des Intervalls überschreitet, sondern der Design-Parameter wird derart verändert, sodass nun sogar der erste Eigenwert im Intervall liegt. Obwohl die negative Approximation bei Abb. 5.4 einen Wert annimmt, der für die genauen Eigenwertzählung nicht möglich ist, so ändert es nichts an dem Verlauf, da in diesem Schritt alle Eigenwert außerhalb des Intervalls liegen und laut Algorithmus 1 die genaue gewichtete Zählung $J(s)$ verwendet wird, um zu entscheiden, ob ein Minimum erreicht wurde. Wie man sehen kann verschwinden die genauen Eigenwertzählung bei Schritt 15. Sollte man nur Informationen über die approximierte Zählung besitzen, so wäre das Verfahren, je nach Programmierung, nicht beendet worden, sondern fortgeführt worden, obwohl es ein Minimum erreicht hatte.

Dieses Verhalten der approximierten gewichteten Eigenwertzählung wird in Kapitel 6 erneut angesprochen.

Betrachte nun die Durchläufe von System 2. Es wurde sich auf die Schritte 0 bis 150 beschränkt, um diese im Detail sehen zu können.

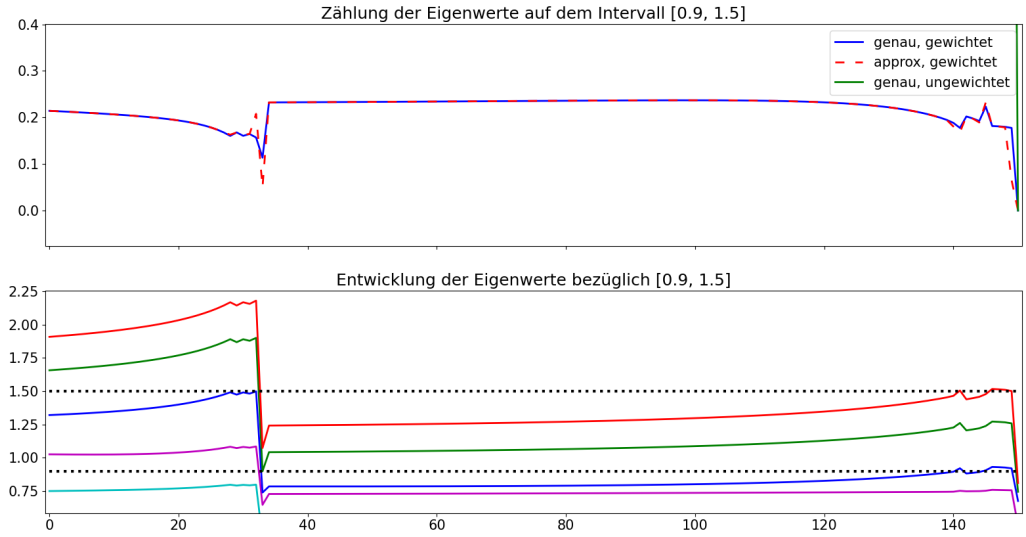

Abbildung 5.5: System 2, $m = 100$, $\lambda_* = 0.05$

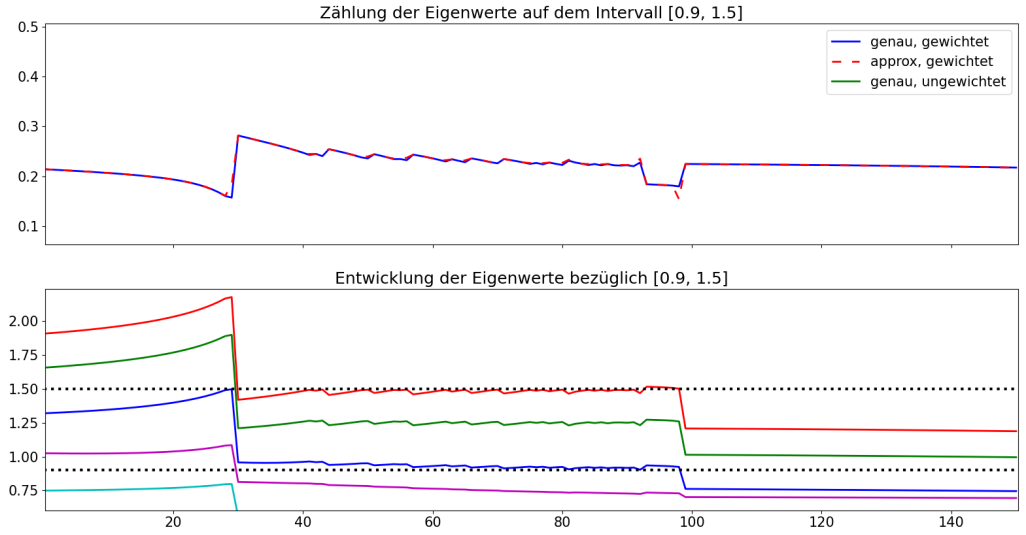

Abbildung 5.6: System 2, $m = 150$, $\lambda_* = 0.05$

Wie man leicht in Abb. 5.5 sehen kann, war das Erreichen eines Minimums an zwei Stellen keine kontrollierte, kleine Veränderung: An beiden Stellen kam ein Eigenwert sehr nahe an eine der beiden Intervallgrenzen. Wie schon in den vorherigen Abbildungen führt das zu einer ungenauen Approximation der gewichteten Eigenwertzählung. Zufällig waren diese Ungenauigkeit und somit auch die Sprünge in Abb. 5.5 derart, dass nach dem zweiten Sprung alle Eigenwerte außerhalb des Intervalls lagen, wohingegen bei Abb. 5.6 die genauere Quadraturformel zur Folge hatte, dass nicht alle Eigenwerte nach diesen Sprüngen außerhalb des Intervalls lagen.

Da schon Veränderungen der Quadraturformel ausreichten, um den Verlauf der

Eigenwerte komplett zu verändern[1], werden auch andere wichtige Faktoren den Algorithmus und somit die Minimalstelle beeinflussen können.

Laut [3, S. 7] hänge der Algorithmus besonders von dem Startwert s_0 und seinen Parametern, wie α, ab.

[1] es reicht sogar ein Wechsel des Gerätes, welches das Programm ausführt, um unterschiedliche Ergebnisse hervorzubringen

Kapitel 6

Verbesserungen

In Kapitel 5 sind folgende Probleme deutlich zum Vorschein getreten: Einerseits ist die approximierte Zählung der Eigenwerte unzureichend, wenn sich ein Eigenwert in der Nähe der Intervallgrenzen befindet. Am einfachsten ist dies an den Abbildungen 5.3, 5.4 und 5.5 zu sehen: Immer wenn ein Eigenwert in die Nähe der Intervallgrenze und damit der Integrationskurve kam, wurde die gewichtete Eigenwertzählung ungenau approximiert, was dazu führte, dass die Systeme in einem Schritt sehr große Sprünge ausführten. Auch das Verhalten des Eigenwerts in den Abbildungen 5.1 und 5.2 kann damit erklärt werden. Hier kam es zwar nicht zu Sprüngen, aber die Quadraturformel verhinderte, dass der Parameter s genug verändert wird, sodass der Eigenwert das Intervall verlassen kann[1]. Ferner sieht man anhand der Abbildungen 5.1, 5.2 und 5.4, dass eine größere Schrittweite des Gradientenverfahrens deutlich bessere Ergebnisse hervorbringen kann. Man sehe zudem, dass sich viele der Durchläufe mit kurzen Schrittweiten λ_* über lange Strecken kaum verändern. Um dem vorzubeugen berechne die Schrittweite des Gradientenverfahrens dynamisch, wie es auch im klassischen Gradientenverfahren [17, S. 285 f.] üblich ist. Zuletzt sind diese langen Zeiten für Probleme mit einstelligen Freiheitsgraden nicht wünschenswert, da laut [1, S. 359] einige tatsächlich verwendete Berechnungsmodelle $n > 1.000.000$ Freiheitsgrade besitzen.

6.1 Verbesserte Quadraturformeln

Man wende sich zuerst der Verbesserung der Quadraturformel zu. Man erhofft sich hier, Ungenauigkeiten, die auftreten, wenn sich ein Eigenwert dem Intervallrand nähert, zu verringern, um große Sprünge in der Verteilung der Eigenwerte zu vermeiden. Durch verbesserte Quadraturformeln sollen die Eigenwerte das Intervall geordnet verlassen und nicht während eines großen Schrittes wie in Abb. 5.5.

Man nutze daher eine Quadraturformel, welche das Integral zwar approximiert, aber nicht an den Stellen λ_a und λ_b ausgewertet wird.

Die einfachste Quadraturformel, die dies bewerkstelligt ist die „Mittelpunkts-Regel“ [19, S. 526].

[1] die Tatsache, dass dieses Verhalten in der Quadraturformel begründet ist, kann in [21, *./Verbesserung_Trapez.py*] gesehen werden, dort wurden die Stützstellen der Trapezregel aus Kapitel 5 so verschoben, dass keine Stützstelle sich in der Nähe von λ_a und λ_b befindet

Sei dazu $\gamma_k, k = 0, \ldots, m-1$ wie in (5.1), nach (5.2) gilt:

$$\begin{aligned}\int_{\gamma_k} L(z,s)\ dz =& \int_{2\pi\frac{k}{m}}^{2\pi\frac{k+1}{m}} L(\tilde{z}_0 + \tilde{r}\exp(it), s) i\tilde{r}\exp(it)\ dt \\ \approx& L(\tilde{z}_0 + \tilde{r}\exp\left(\frac{2\pi i}{m}(k+\frac{1}{2})\right), s) i\tilde{r}\left(\frac{2\pi i}{m}(k+\frac{1}{2})\right)\frac{2\pi}{m} \\ =& L(z_{k+\frac{1}{2}}, s)(\tilde{z}_0 - z_{k+\frac{1}{2}})\frac{2\pi i}{m}\,,\end{aligned}$$

wobei die Definition $z_{k+\frac{1}{2}} := \tilde{z}_0 + \tilde{r}\exp(\frac{2\pi i}{m}(k+\frac{1}{2})), k = 0, \ldots, m$ und die Formel der Mittelpunkts-Regel aus [19, S. 526] verwendet wurde.

Es gilt für $J(s)$

$$\begin{aligned}J(s) =& \frac{1}{2\pi i}\sum_{k=0}^{m-1}\int_{\gamma_k} L(z,s)\ dz \approx \frac{1}{2\pi i}\sum_{k=0}^{m-1} L(z_{k+\frac{1}{2}}, s)(\tilde{z}_0 - z_{k+\frac{1}{2}})\frac{2\pi i}{m} \\ =& \frac{1}{2\pi i}\left[\frac{2\pi i}{m}\sum_{k=0}^{m-1} L(z_{k+\frac{1}{2}}, s)(\tilde{z}_0 - z_{k+\frac{1}{2}})\right] =: J^*_{\text{Mittel}}(s)\,. \qquad (6.1)\end{aligned}$$

Die Faktoren vor der Summe wurden wieder nicht gekürzt, da der erste Faktor von der Anwendung des Residuensatzes stammt und der zweite Faktor ein Teil der Quadraturformel ist[2].

Durch die Verwendung dieser Quadraturformel erhält man folgende Ergebnisse:

System	m	λ_*	Iterationen	Zeit in s
1	100	0.05	142	3.7
1	150	0.05	143	5.3
1	100	0.5	15	0.5
1	150	0.5	15	0.7
2	100	0.05	300	13.9
2	150	0.05	289	21.7

Tabelle 6.1: Kennzahlen bei Verwendung der Mittelpunkts-Regel

Die Ergebnisse und die zugehörigen Plots können auch in der Datei [21, *./Verbesserung_Mittelpunkt.py*] betrachtet werden.

Es fällt auf, dass die Minimierung in den ersten zwei Durchläufe im Vergleich zu den Durchläufen aus Tabelle 5.1 nun viel schneller erfolgen, da nur 142-143 Schritte benötigt werden. Natürlich wären diese Durchläufe auch schneller, wenn sie alle zu keinem Ergebnis kommen würden, da hier die Funktion pro Teilintervall nur an einer Stelle auszuwerten ist.

Wie man ferner sehen kann, besteht in System 1 eine indirekte Proportionalität zwischen Schrittweite λ_* und der Anzahl an Schritten, was auf die Eindimensionalität von Design-Parameter s zurückzuführen ist. In den Durchläufen zu System 2

[2]falls man die Faktoren kürzen würde, erhält man die Stützstellen z_k und Gewichte ω_k aus [7, S. 128], das Kürzen würde die Berechnung zwar beschleunigen, aber die Berechnung der J^* wurde in [21, *./Funktionen.py*] so implementiert, dass jede Quadraturformel genutzt werden kann

sieht man zudem, dass sich die Anzahl an Iterationen für $m = 100$ im Vergleich zu Tabelle 5.1 verdoppelt, allerdings ist dies auf die Tatsache zurückzuführen, dass der Verlauf der Eigenwerte keine plötzlichen Sprünge mehr ausführt, wie sie in Abb. 5.5 zu sehen waren.

Zuletzt sei angemerkt, dass sich die Zeiten zwischen 2 Durchläufen mit unterschiedlicher Anzahl an Teilintervallen m ungefähr im Verhältnis $1 : 1.5$ befinden, was offenbar auf das Verhältnis der Anzahl an Teilintervallen m zurückzuführen ist.

Die folgende Abbildung zeigt, wie sich die Eigenwerte des ersten Durchlaufs verändern. Da ab nun alle Plots von System 1 sehr ähnlich zueinander sind, sei Abb. 6.1 in diesem Abschnitt der einzige Plot zu System 1.

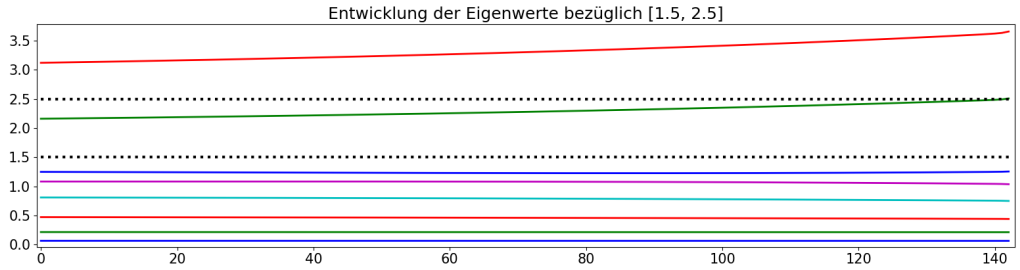

Abbildung 6.1: System 1, $m = 100$, $\lambda_* = 0.05$, Mittelpunkts-Regel verwendet

Der zweite Eigenwert verlässt in Abb. 6.1 langsam das Intervall und hat keine Probleme, die Intervallgrenze zu überqueren. Betrachte nun einen Durchlauf zu System 2:

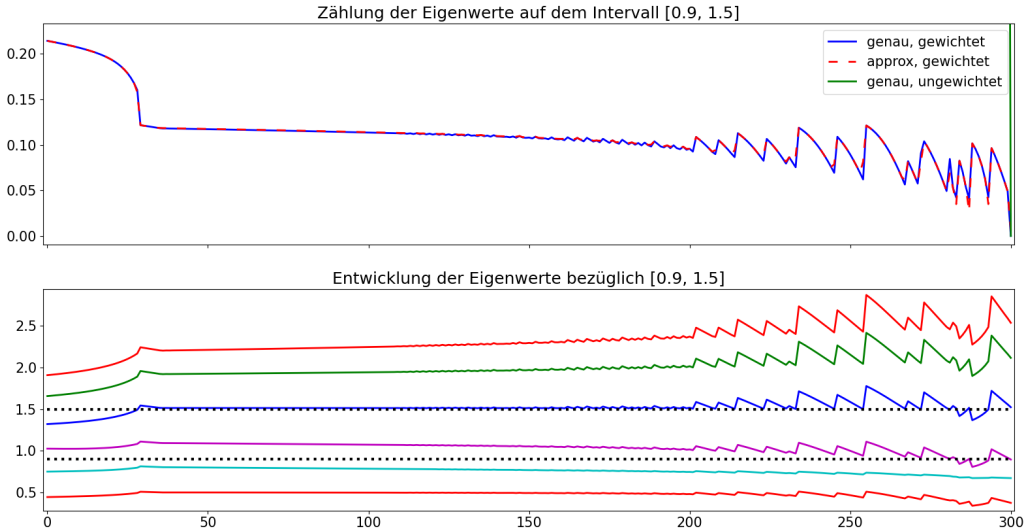

Abbildung 6.2: System 2, $m = 100$, $\lambda_* = 0.05$, Mittelpunkts-Regel verwendet

In Abb. 6.2 kann man sehen, dass immer Sprünge auftreten, wenn ein Eigenwert sich einer Intervallgrenze nähert und die Approximation der gewichteten Eigenwertzählung ungenau wird[3]. Wir werden nun versuchen, die beobachteten Sprünge

[3]siehe auch [21, *./Verbesserung_Mittelpunkt.py*]

zu unterbinden. Nutze dazu eine genauere Quadraturformel: die „Zweipunkt-Formel“ [19, S. 526]. Analog zu (5.3) und (6.1) folgt:

$$\begin{aligned} J(s) \approx & \frac{1}{2\pi i}\left[\frac{\pi i\tilde{r}}{m}\sum_{k=0}^{m-1}\left(L(z_{k,-},s)\exp\left(-\frac{\pi i}{\sqrt{3}m}\right)+L(z_{k,+},s)\exp\left(\frac{\pi i}{\sqrt{3}m}\right)\right)\right] \\ & =: J^*_{\text{Gauss}}(s) \end{aligned} \tag{6.2}$$

mit $z_{k,\pm} := \tilde{z}_0 + \tilde{r}\exp\left(\frac{\pi I}{\sqrt{3}m}(2\sqrt{3}k+\sqrt{3}\pm 1)\right)$.

Diese Formel erhält man durch Anwenden der Zweipunkt-Formel aus [19, S. 526] und Einsetzen von

$$u = \frac{a+b}{2} = \frac{2\pi}{m}(k+\frac{1}{2}) \text{ und } v = \frac{b-a}{2} = \frac{\pi}{m}$$

Es folgen die Ergebnisse des Programms *./Verbesserung_Gauss2.py*.

System	m	λ_*	Iterationen	Zeit in s
1	100	0.05	145	7.5
1	150	0.05	145	10.3
1	100	0.5	15	0.9
1	150	0.5	15	1.1
2	100	0.05	112	11.9
2	150	0.05	63	9.3

Tabelle 6.2: Kennzahlen bei Verwendung der Zweipunkt-Formel

Obwohl die Zweipunkt-Formel eine „Gaußsche Quadraturformel“[19, S. 523] ist und damit einen optimalen Genauigkeitsgrad von 5 besitzt (vgl. [19, S. 522f.]), so ist der Unterschied zur Mittelpunkts-Regel in den Durchläufen zu System 1 gering. Im Vergleich zu Tabelle 6.1 bemerke man aber, dass sich die benötigten Zeiten bei System 1 verdoppelten. Dies war zu erwarten, wenn die doppelte Anzahl an Funktionswerten pro Teilintervall berechnet wird. In den Durchläufen zu System 2 kann aber festgestellt werden, dass sich die Anzahl an benötigten Iteration stark verringert. Betrachte dazu auch Abb. 6.3.

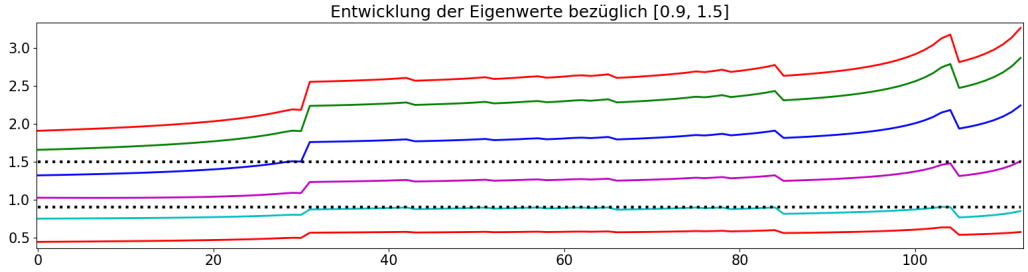

Abbildung 6.3: Plot zu System 2, $m = 100$, $\lambda_* = 0.05$, Zweipunkt-Formel verwendet

Man sieht in Abb. 6.3, dass sich die Eigenwerte hier von Anfang an schnell auf den Intervallrand zubewegen, im Gegensatz zu Abb. 6.2, wo die Eigenwerte sich in den

Schritten 40 bis 200 wenig bis kaum veränderten. Durch die bessere Approximation kann trotz des erhöhten Rechenaufwands, der mit der Zweipunkt-Formel entsteht, die benötigte Zeit stark reduziert werden. Ferner kommt dazu, dass für System 2 nun auch die Wahl von m eine Rolle spielt: In den Tabellen 6.1 und 6.2 verringert sich die Anzahl an Iterationen bei erhöhter Anzahl an Stützstellen. Man nehme daher für die Minimierung der Eigenwerte in dem restlichen Kapitel immer die Zweipunkt-Formel.

Die Tatsache, dass diese zwei Quadraturformeln nicht bei λ_a und λ_b ausgewertet werden und sie deshalb in Nähe einer Polstelle genauer approximieren, kann auch in Programm *./Vergleich Quadraturformeln.py* gesehen werden. Dort wurde ebenfalls die verschobene Trapezregel betrachtet, welche die Stützstellen $z_{k+\frac{1}{2}}$ statt z_k verwendet. Allein dies macht einen gewaltigen Unterschied:

Für eine immer näher an das Kurvenintegral rückende Polstelle wächst die Approximation durch die Trapezregel aus (5.3) sehr stark an. Alle anderen Quadraturformeln nähern sich stattdessen 0.5. Wie auch in den Durchläufen zu System 1 liegen die Ergebnisse der Mittelpunkts-Regel und der Zweipunkt-Formel sehr dicht beieinander.

6.2 dynamische Schrittweite

Als nächstes wende man sich dem Gradientenverfahren zu: Die Schrittweite λ_* soll nun nicht mehr fest sein, sondern möglichst optimal berechnet werden.

Berechne daher[4] $\nabla J^*(s_{\text{neu}})$ für $s_{\text{neu}} = s_{\text{alt}} - d\lambda_*$ und entscheide aufgrund dessen, ob:

- dieser Wert (s_{neu}) zurückgegeben wird,
- der vorherige Wert (s_{alt}) zurückgegeben wird oder
- man noch einen Schritt geht.

Offenbar bestimmt λ_* nicht mehr direkt die Schrittweite, sie bestimmt aber die Entfernung zweier Punkte s_{neu}.

Der Algorithmus zur Bestimmung des neuen Punktes ist in [21, *./Funktionen.py*] in der Funktion *schrittGradientenverfahren()* definiert.

Diese Funktion gibt einen Wert s aus, der möglichst nahe an der optimalen Stelle bezüglich Richtung d ist. Die optimale Stelle würde laut [17, S. 286]

$$\nabla J(s) \cdot d^T = 0$$

erfüllen, der nächste Schritt würde dann garantiert senkrecht auf diesem Schritt stehen. Da man nur eine endliche Zahl an Stellen auswerten kann, findet man diese Stelle fast sicher nicht. Daher soll die Bedingung

$$|\nabla J(s) \cdot d^T| < 0.00001$$

ausreichen. Es besteht weiterhin die Möglichkeit, dass s die zulässige Menge Ω verlässt, oder man über einen optimalen Parameter springt. Auch diese Fälle werden in der Funktion *schrittGradientenverfahren()* behandelt.

[4] mit d wie in Algorithmus 1

Das Anwenden dieses Algorithmus zur Bestimmung des nächsten Punktes bringt folgende Ergebnisse:

System	m	λ_*	Iterationen	Zeit in s
1	100	0.05	1	10.4
1	150	0.05	1	17.26
1	100	0.5	1	1.8
1	150	0.5	1	3.1
2	100	0.05	24	16.7
2	150	0.05	178	62.0

Tabelle 6.3: Kennzahlen bei Verwendung eines dynamischen Schrittweite

Die Ergebnisse sind auch in [21, *./Verbesserung_dynamischeSchrittweite.py*] zu finden.

Wie man sehen kann, wird die Eigenwertzählung bei allen Durchläufen zu System 1 innerhalb eines Schrittes minimiert. Dies war aber zu erwarten, da s in System 1 ein Skalar war. Sofern die Ableitung hier immer das gleiche Vorzeichen besitzt, gehen auch alle anderen Durchläufe aus Kapitel 6 bei System 1 immer nur in eine Richtung. Somit erreicht hier das Verfahren in einem Schritt, wofür vorher p Schritte benötigt wurden.

Hier benötigt der zweite Durchlauf zu System 2 mehr Schritte als der der vergleichbare Durchlauf aus Tabelle 6.2 und die Minimierung dauert dabei noch länger, da jeder Schritt des Gradientenverfahrens hier zusätzlich aus Unterschritten (der Bestimmung der optimalen Schrittweite) besteht.

Insgesamt lohnt sich eine dynamische Schrittweite in diesem Problem nicht, da keine einfache Möglichkeit besteht, die optimale Schrittweite zu bestimmen. Für die Bestimmung der optimalen Schrittweite werden hier die Schritte in die Richtung d gegangen, aber schon nach dem ersten Schritt ist diese Richtung im Allgemeinen nicht mehr optimal. Ein Verfahren ohne dynamsiche Schrittweite würde nun in wieder in die Richtung des steilsten Abstiegs gehen, wohingegen dieses Verfahren solange in Richtung d geht, bis es einen (bezüglich dieser Richtung) optimalen Parameter s erreicht hat. Es kommt auch zu keiner Reduktion der Rechendauer, da nach jedem dieser Unterschritte wieder $\nabla J(s)$ bestimmt wird.

6.3 Approximation der Ableitung der gewichteten Eigenwertzählung

Zuletzt bemerke zuerst, dass die Approximation des Integrals durch die Quadraturformel die meiste Zeit benötigt, hingegen die genaue gewichtete Eigenwertzählung $J(s)$ sehr schnell berechnet werden kann. Da aber keine explizite Abhängigkeit vom Design-Parameter s besteht, ist diese Formel nur gut, um als Bedingung für das Ende des Gradientenverfahrens zu dienen.

Approximiere nun $\nabla J(s)$ durch die Anwendung eines Differenzenverfahrens auf $J(s)$ aus (4.1), wie es in Kapitel 5 angewendet wurde, um $\frac{dM}{ds}$ zu approximieren.

Nun kann diese Approximation genutzt werden, um die Berechnung von $\nabla J^*(s)$ und damit die Anwendung der Quadraturformel zu umgehen[5].

Da diese Berechnung sehr viel schneller ist, erhöhe die maximal mögliche Anzahl an Iterationen pro Durchlauf auf 100000. Die Anzahl an Teilintervallen wird zwar nicht für die Berechnung an sich benötigt, aber bei jeder Iteration des Gradientenverfahrens wird für den aktuellen Wert des Design-Parameters s ein entsprechendes $J^*(s)$ berechnet und gespeichert, um den Verlauf der approximierten gewichteten Eigenwertzählung in dem oberen Plot abzubilden. Um diesen Mehraufwand so gering wie möglich zu halten sei $m = 1$. Die approximierte gewichtete Eigenwertzählung wird dadurch zwar sehr ungenau, aber hier wird sie für keinen Schritt des Gradientenverfahrens verwendet. Nehme daher für System 1 nicht vier Durchläufe, sondern nur zwei, da die Unterscheidung bezüglich m entfällt. Den vierten Durchlauf findet man zwar nicht in dem Programm [21, *./Verbesserung_NablaJ.py*], man kann ihn aber durch Veränderung des Werts von Argument „maxIter“ in Zeile 15 berechnen lassen. Dieser Durchlauf wird nicht immer durchgeführt, da er zu viel Zeit in Anspruch nimmt. Die folgenden Ergebnisse können auch in *./Verbesserung_NablaJ.py* wiedergefunden werden.

System	m	λ_*	Iterationen	Zeit in s
1	1	0.05	31	0.03
1	1	0.5	4	0.01
2	1	0.05	> 100000	> 198
2	1	0.05	> 150000	> 630

Tabelle 6.4: Kennzahlen bei Verwendung von $\nabla J(s)$

Man beachte, dass diese Approximation für System 1 genau genug war, um ein Minimum innerhalb von 0.03 s zu finden. Allerdings ist dieses Verfahren in dieser Form für System 2 unzureichend. Man könnte die Ableitung genauer approximieren, indem die Schrittweite h aus (5.4) verringert wird. Ferner würde vielleicht eine größere Schrittweite λ_* helfen, um (ähnlich wie bei System 1) die Anzahl an Iterationen zu verringern. Solche Überlegungen werden in dieser Arbeit aber nicht weiter ausgeführt.

Für die Durchläufe für System 2 erkenne man auch, dass die Zeit nun nicht mehr proportional zu den gegangenen Schritten ist. Dies liegt wahrscheinlich daran, dass nicht mehr alle Werte im Arbeitsspeicher gespeichert werden konnten, wodurch sich die Zugriffszeiten verlängern.

Diese Variante hat somit zwar sehr viel Potenzial (siehe Durchläufe zu System 1), ist aber in der jetzigen Form nicht für alle Probleme genau genug.

[5] man ersetzt also die Ableitung der Approximation von J durch die Approximation der Ableitung von J

Kapitel 7

Auswertung

In dieser Arbeit wurden in Kapitel 2 der Matrix-Pencil, die Eigenwerte eines Matrix-Pencils, die allgemeine Schur-Zerlegung und die Identität von Futamura vorgestellt. Während die Schur-Zerlegung für den Beweis der Identität von Futamura benötigt wird, liefert sie auch eine Aussage über die Eigenwerte des Matrix-Pencils. Diese Identität erlaubt eine direkte Aussage über die Anzahl an Eigenwerten in einem Intervall, dafür muss aber ein Kurvenintegral berechnet werden.

In Kapitel 3 wurden die Systeme vorgestellt, welche in Kapitel 5 und 6 behandelt werden. Um die Eigenwerte dieser Systeme zu berechnen, wurden die Massen- und Steifigkeitsmatrizen der Systeme berechnet und eine Formel für die Berechnung der Eigenkreisfrequenzen aufgestellt. Durch eine Substitution erhält man in (3.12) ein allgemeines Eigenwertproblem des Matrix-Pencils $(K - \lambda M)$. Die gesuchten Eigenwerte sind dabei genau die Eigenkreisfrequenzen des Systems.

In Kapitel 4 wurde die gewichtete Eigenwertzählung auf einem Intervall hergeleitet. Diese Funktion besitzt die Form aus (4.1) und ist einfach zu berechnen, hängt aber nicht direkt von dem Design-Parameter ab.

Durch die Identität von Futamura, dem Residuensatz und einer Quadraturformel erhält man in (5.3) eine Approximation dieser Eigenwertzählung, welche direkt von s abhängt.

Diese Approximation wird in Kapitel 5 genutzt, um die Eigenwerte der Systeme auf einem Intervall zu zählen und anschließend zu eliminieren.

Als Quadraturformel wurde in Kapitel 5 die normale Trapezregel verwendet, welche aber unzureichende Ergebnisse brachte (siehe Tabelle 5.1).

Unter anderem wurde in Kapitel 6 diese Problem behoben. Auch wurde in Kapitel 6 das Gradientenverfahren aus Algorithmus 1 weiterentwickelt, sodass die Schrittweite nun dynamisch berechnet wird. Durch mangelnde Möglichkeiten, diese Schrittweite effizient zu berechnen, war dieses Verfahren aber nicht so effektiv, wie das in Kapitel 5 vorgestellte Gradientenverfahren.

Zuletzt wurde in Kapitel 6 auch eine Approximation der Ableitung von $J(s)$ eingeführt. Diese kann zwar die Eigenwertzählung von System 1 unglaublich schnell minimieren, sie versagte aber bei einem mehrdimensionalen Design-Parameter, wie er in System 2 genutzt wird.

Der schnellste Weg zur Minimierung der Eigenwertzählung ist daher eine möglichst genaue Quadraturformel, welche möglichst wenig Stützstellen benötigt, um ein gutes

Ergebnis mit geringstem Zeitaufwand zu erhalten.

Für eindimensionale Design-Parameter kann das Verfahren durch die Approximation von $\nabla J(s)$ stark beschleunigt werden. Für einen Parameter $s \in \mathbb{R}^l, l > 1$ kann diese Approximation auch weiterhelfen, allerdings wurde dieser Ansatz für mehrdimensionale Parameter nicht weiter verfolgt.

Für alle Design-Parameter kann aber die Funktion ∇J^* mit einer entsprechend genauen Quadraturformel verwendet werden, um die Eigenwertzählung kontrolliert zu minimieren. Dieses Verfahren dauert zwar länger, bringt aber verlässlich gute Ergebnisse (siehe Tabelle 6.1 und 6.2).

Es gibt aber offenbar noch andere Möglichkeiten die Minimierung zu beschleunigen. Man könnte einerseits ein anderes Minimierungsverfahren nutzen, wie das Newton-Verfahren (siehe [17, S. 290]). Für dieses Verfahren bräuchte man zwar die zweite Ableitung von J oder J^*, es könnte sich aber positiv auf die Rechenzeit auswirken.

Zudem könnte man für Systeme mit einer hohen Anzahl an Freiheitsgraden die Spur(tr) in (4.11) approximieren. Man erhofft sich dadurch, die Komplexität der Funktion und ebenso die benötigte Zeit zu verringern (vgl. [7, S. 128]). Dazu wäre „Hutchinson's Schätzer“ [25, S. 142][1] oder dessen Erweiterung „Hutch++“ [25, S. 142] nützlich. Solche Überlegungen könnten in einer weiterführenden Arbeit behandelt und mit den hier erhaltenen Ergebnissen verglichen werden.

[1] Übersetzung des Autors

Kapitel 8

Verzeichnisse

Literaturverzeichnis

1. DRESIG, Hans; HOLZWEISSIG, Franz; ROCKHAUSEN, Ludwig. *Maschinendynamik*. 10., neu bearb. Aufl. Springer, 2011. ISBN 9783642160097. Auch verfügbar unter: `https://katalog.slub-dresden.de/id/0-642805539`.
2. TILLY, Graham P. Dynamic behaviour and collapses of early suspension bridges. In: *Proceedings of the Institution of Civil Engineers-Bridge Engineering*. Thomas Telford Ltd, 2011, Bd. 164, S. 75–80. Nr. 2.
3. TKACHUK, Anton; TKACHUK, Mykola. Weighted Eigenvalue Counts on Intervals for Spectrum Optimization. In: *International Conference on Advanced Mechanical and Power Engineering*. Springer, 2021, S. 228–237.
4. GSELL, Daniel. *Nicht axialsymmetrische Wellenausbreitung in anisotropen zylindrischen Strukturen*. 2002. Diss. ETH Zurich.
5. GOLUB, Gene H.; VAN LOAN, Charles F. *Matrix computations*. 3. ed. Johns Hopkins Univ. Pr., 1996. ISBN 0801854148. Auch verfügbar unter: `https://katalog.slub-dresden.de/id/0-197761291`.
6. PARLETT, Beresford N. Symmetric matrix pencils. *Journal of Computational and Applied Mathematics*. 1991, Jg. 38, Nr. 1-3, S. 373–385.
7. FUTAMURA, Yasunori; TADANO, Hiroto; SAKURAI, Tetsuya. Parallel stochastic estimation method of eigenvalue distribution. *JSIAM Letters*. 2010, Jg. 2, S. 127–130.
8. WERNER, Dirk. Lineare Abbildungen. In: *Lineare Algebra*. Springer, 2022, S. 57–72.
9. HORN, Roger A.; JOHNSON, Charles R. *Matrix analysis*. 2nd ed. Cambridge University Press, 2013. ISBN 9780521548236. Auch verfügbar unter: `https://katalog.slub-dresden.de/id/0-1859628885`.
10. DÜCKER, Simon Julius. *Prinzip von d'Alembert - Physik* [online]. [besucht am 2024-09-04]. Abger. unter: `https://www.ingenieurkurse.de/physik/kinetik-ursache-von-bewegungen/prinzip-von-d-alembert.html`.
11. DÜCKER, Simon Julius. *Gewichtskraft, Federkraft - Physik* [online]. [besucht am 2024-09-04]. Abger. unter: `https://www.ingenieurkurse.de/physik/kinetik-ursache-von-bewegungen/beispiele-fuer-kraefte/gewichtskraft-federkraft.html`.

12. KUNST, Prof. Dr.-Ing. Dr. Sabine. *Kraft und Bewegungsänderung* [online]. [besucht am 2024-09-14]. Abger. unter: https://www.leifiphysik.de/mechanik/kraft-und-bewegungsaenderung/grundwissen/traegheitssatz-im-beschleunigten-system.

13. O.V. *Einfachste mechanische Schwingsysteme* [online]. [besucht am 2024-09-04]. Abger. unter: https://www.georgi-dd.de/files/a22.pdf.

14. PREUSSLER, Prof. Dr.-Ing. T. *5. Mehrmassenschwinger* [online]. [besucht am 2024-09-04]. Abger. unter: https://www.umwelt-campus.de/fileadmin/Umwelt-Campus/User/TPreussler/Download/Maschinendynamik_und_Betriebsfestigkeit/Foliensaetze/05_Mehrmassenschwinger_1.pdf.

15. MARSHALL, Donald E. *Complex Analysis*. Cambridge University Press, 2019. ISBN 9781107134829. Auch verfügbar unter: https://katalog.slub-dresden.de/id/0-1859627056.

16. O.V. *derivative of inverse matrix* [online]. [besucht am 2024-09-24]. Abger. unter: https://planetmath.org/derivativeofinversematrix.

17. BURKARD, Rainer E.; ZIMMERMANN, Uwe. *Einführung in die mathematische Optimierung*. Springer Spektrum, 2012. ISBN 9783642286728. Auch verfügbar unter: https://katalog.slub-dresden.de/id/0-688968007.

18. STEIN, Elias M; SHAKARCHI, Rami. *Complex analysis*. Bd. 2. Princeton University Press, 2010.

19. HERMANN, Martin. *Numerische Mathematik*. 3. überarb. und erw. Aufl. Oldenbourg, 2012. ISBN 9783486719703. Auch verfügbar unter: https://katalog.slub-dresden.de/id/0-1651937834.

20. GROSSMANN, Christian; ROOS, Hans-Görg. *Numerik partieller Differentialgleichungen*. Teubner, 1992. ISBN 3519020890. Auch verfügbar unter: https://katalog.slub-dresden.de/id/0-02750820X.

21. GÖPEL, Noah. *Bachelor_GewEwZahlung/Programmierung* [online]. [besucht am 2024-09-24]. Abger. unter: https://github.com/NextGen73/Bachelor_GewEwZaehlung/tree/main/Programmierung.

22. JOE; HALL, Matt; ABOAMMAR. *Append 2d array to 3d array* [online]. 2022. [besucht am 2024-09-25]. Abger. unter: https://stackoverflow.com/questions/72330041/append-2d-array-to-3d-array.

23. JARRE, Florian; STOER, Josef. *Optimierung: Einführung in mathematische Theorie und Methoden*. Projektionsverfahren. Springer Berlin Heidelberg, 2019. ISBN 978-3-662-58855-0. Abger. unter DOI: 10.1007/978-3-662-58855-0_10.

24. NUMPY DEVELOPERS. *numpy.transpose* [online]. [besucht am 2024-09-25]. Abger. unter: https://numpy.org/doc/2.0/reference/generated/numpy.transpose.html.

25. MEYER, Raphael A; MUSCO, Cameron; MUSCO, Christopher; WOODRUFF, David P. Hutch++: Optimal stochastic trace estimation. In: *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, 2021, S. 142–155.

Abkürzungsverzeichnis

Abb. Abbildung

approx approximiert

o.V. ohne Verfasser

vgl. vergleich

Erklärung

Hiermit erkläre ich, dass ich die am 26. 09. 2024 eingereichte Bachelorarbeit zum Thema *Gewichtete Zählung von Eigenwerten auf einem Intervall* unter Betreuung von Prof. Dr. Oliver Sander selbstständig erarbeitet, verfasst und Zitate kenntlich gemacht habe. Andere als die angegebenen Hilfsmittel wurden von mir nicht benutzt.

Dresden, 26. 09. 2024 Unterschrift