„Weniger ist mehr.“

Eigenwertfilterung am Beispiel hermitescher Eigenwertprobleme

Thorsten Matthias Lucke

Betreuer: Prof. Jörg Liesen, Prof. Christian Mehl

Arbeit zur Erlangung des akademischen Grades
Bachelor of Science

Department Name
Technische Universität Berlin

20. Mai 2017

Dedicatio

For anyone.

Declaratio

Hiermit erkläre ich, dass ich alles von Wikipedia abgeschrieben habe, so wie sich das gehört für eine großartige arbeit.

Recognitio

Thanks to anyone.

Inhaltsverzeichnis

Kapitel 1

Introductio

Das Lösen von Eigenwertproblemen ist eine Standarddisziplin in der numerischen linearen Algebra. Gleichungen der Gestalt

$$Ax = \lambda Bx \tag{1.1}$$

begegnet man in ganz unterschiedlichen Kontexten. So sind sie beispielsweise bei der Bestimmung von Eigenfrequenzen oder dem Ermitteln von Fixpunkten beim Rotieren eines Fußballs[1] ebenso wie beim Untersuchen des PageRanks einer Website von Bedeutung. Entsprechend strotz der Kanon von angebotenen numerischen Lösungsmethoden von Vielfalt und Virtuosität.

Nun mag der Fall eintreten, da es notwendig wird, lediglich eine Teilmenge aus der Menge aller Eigenpaare zu untersuchen.
Geeignetes Beispiel finden.

Wie soll nun aber der eifrige Numeriker die gewünschte Menge an Eigenpaaren finden?

Man könnte zunächst versuchen sämtliche Eigenpaare zu berechnen, die das Problem hergibt und händisch die gewünschte Teilmenge auswählen. Bei einem Problem dieser Größenordnung dürfte der Rechenknecht allerdings eine ordentliche Weile beschäftigt sein und das Leben ist fürwahr zu knapp, um auf das Terminieren von Algorithmen zu warten.

Viel eleganter lässt sich dieses Problem mit einem Verfahren lösen, welches in dem von *Eric Polizzi* geschriebenen Paper „Density-matrix-based algorithm for solving eigenvalue problems“ [Pol09] vorgestellt wird. Speziell ist dieser mit *FEAST* abgekürzte Algorithmus darauf ausgelegt, Probleme der Art (1.1) für eine hermitesche Matrix A und eine hermite-

[1] Hier wird auf den bekannten *Satz vom Fußball* angespielt. Dieser besagt, dass auf einem Fußball zwei Punkte existieren, die zu Spielbeginn und zur Halbzeit an der gleichen Stelle liegen – informell formuliert.

sche, positiv definite Matrix B zu lösen.[2]

Die vorliegende Arbeit wird die grundlegenden mathematischen Ideen dieses Algorithmus' vorstellen. Es werden Möglichkeiten der Implementation präsentiert und neben einer naiven Implementation in *MATLAB* auch effizientere Umsetzungen besprochen – etwa vermöge der von *Mario Berljafa* und *Stefan Güttel* entwickelten „*Rational Krylov Toolbox for MATLAB*“ [Ber16].

1.1 Grundlagen

Um das Lesen dieser Arbeit mehr zu einer Freude denn zu einer Schikane zu machen, soll dieser Abschnitt einige Grundlagen der linearen Algebra und der Funktionentheorie bereitstellen. Obschon sich der Autor bemüht hat, in der Literatur gängige Notation zu benutzen, bittet er den verständnissvollen Leser bei Unklarheiten im Anhang „Notationen“ nachzuschlagen.

Den Anfang machen Definitionen und Resultaten aus der Matrizentheorie. Eine Matrix $A \in \mathbb{C}^{n,n}$ wird als *hermitesch* bezeichnet, falls sie die Identität $A = A^H$ erfüllt. Sie ist *positiv definit*, sofern für alle Vektoren $x \in \mathbb{C}^n \setminus \{0\}$ die Abschätzung

$$x^H A x > 0$$

gilt. Folglich werden wir eine Matrix *hermitesch positiv definit (HPD)* nennen, wenn sie sowohl hermitesch als auch positiv definit ist.

Ist $A \in \mathbb{C}^{n,n}$ eine solche HPD-Matrix und sind $x, y \in \mathbb{C}^n$, so werden wir anstelle von $x^H A y$ die Notation $\langle x, y \rangle_A$ verwenden. Im Falle $A = I_n$ schreiben wir kurz $\langle x, y \rangle$.

[2] Die Definitionen dieser Begriffe werden im anschließenden Abschnitt wiederholt.

Kapitel 2

Polynomfilter

Wie der Titel des Kapitels bereits ankündigt, beschäftigt sich dieser Teil der Arbeit mit der Filterung von Eigenwerten mit Hilfe von Polynomfunktionen.

Kapitel 3

Rationalfunktionelle Filter

Nachdem vorgestellt wurde, wie mit Polynomen Eigenwerte extrahiert werden können, verpflichtet sich diese Kapitel der Vorstellung einer anderen Klasse von Methoden, die das Filtern von Eigenwerten ermöglichen. Die Protagonisten dieser Methoden werden durch sogenannte *rationale Funktionen* verkörpert. Ausgehend von zwei Polynomfunktionen $p, q \colon \mathbb{R} \to \mathbb{R}$ mit

$$p(t) := \sum_{k=0}^{n} p_k t^k \text{ und } q(t) := \sum_{k=0}^{n} q_k t^k$$

für reelle/rationale? Koeffizienten $(p_i)_{i=1:n}$ und $(q_i)_{i=1:n}$ definieren wir eine *rationale Funktion* $r \colon \mathbb{R} \to \mathbb{R}$ vermöge

$$r(t) := \frac{p(t)}{q(t)}.$$

Kapitel 4

Konturintegration am Beispiel des FEAST-Algorithmus'

Abschließend soll eine letzte Klasse von Methoden vorgestellt werden, die das Filtern von Eigenwerten ermöglicht. Konturintegration.

4.1 Mathematische Idee

Zun"chst wollen wir uns mit den mathematischen Ideen, welche die Grundlage für den FEAST-Algorithmus bilden, vetraut machen. Dafür werden wir die von E. Polizzi in seinem Paper [Pol09] verwendeten Notationen in großen Teilen – aber nicht vollständig – übernehmen.

Es seien nun zwei Zahlen $\lambda_1, \lambda_2 \in \mathbb{R}$ vorgegeben, die das reelle Intervall $I := [\lambda_1, \lambda_2] \subseteq \mathbb{R}$ definieren. Für zwei hermitesche Matrizen $A, B \in \mathbb{C}^{n,n}$ mit der zusätzlichen Forderung, dass B positiv definit ist, sollen Paare der Gestalt $(\lambda, x) \in I \times \mathbb{C}^n$ ermittelt werden, welche der verallgemeinerten Eigenwertgleichung

$$Ax = \lambda Bx \tag{4.1}$$

genügen. Um diese Eigenpaare[1] bestimmen zu können, wird das Problem mit Hilfe einer gewissen Matrix $Q \in \mathbb{C}^{n,n}$ in das äquivalente Problem

$$A_Q \phi = \mu B_Q \phi \tag{4.2}$$

überführt. Hierbei sind $A_Q = Q^T A Q$, $B_Q = Q^T B Q$ und $(\mu, \phi) \in I \times \mathbb{C}^n$. Wird die Matrix Q richtig gewählt, so ist dann jeder zulässige Eigenwert von (4.2) auch ein zulässi-

[1] Um der besseren Lesbarkeit Willen werden im Folgenden die verallgemeinerten Eigenvektoren und deren verallgemeinerte Eigenwerte kurz als Eigenvektor und Eigenwert bezeichnet.

ger Eigenwert von (4.1) und umgekehrt.[2] Für die Ermittlung der gesuchten Eigenvektoren bedarf es hingegen zusätzlicher Arbeit.

Zur Bestimmung der Transformationsmatrix Q wird eine Konturintegration bemüht. Ausgangspunkt dieser Integration ist die durch

$$\begin{aligned} G\colon \Omega &\to \mathbb{C}^{n,n} \\ \omega &\mapsto (\omega B - A)^{-1} \end{aligned}$$

definierte GREEN-Funktion, wobei $\Omega \subseteq \mathbb{C}$ eine passende Teilmenge der komplexen Zahlen ist. Insbesondere müssen Ω und das Spektrum von $B^{-1}A$ disjunkt sein, da G andernfalls nicht wohldefiniert ist.

Die Funktion G wird nun über eine geschlossene komplexe Kurve γ, die um das vorgegebene Intervall I herumläuft, in der Gestalt

$$-\frac{1}{2\pi\iota}\int_\gamma G(\omega)\,\mathrm{d}\omega$$

integriert.[3]

Wir wollen nun annehmen, dass es genau $k \in \mathbb{N}$ Eigenwerte gibt, die im Inneren des Intervalls I liegen.[4] Für zu diesen Eigenwerten passende Eigenvektoren $\{x_i\}_{i=1:k} \subseteq \mathbb{C}^n$ sei außerdem die Matrix $X_k := [x_i]_{i=1:k}$ gegeben. Dann lässt die Konturintegration Rückschlüsse auf die Eigenvektoren zu, denn man kann zeigen, dass die Identität

$$X_k X_k^T = \sum_{i=1}^{k} x_i x_i^T = -\frac{1}{2\pi\iota}\int_\gamma G(\omega)\,\mathrm{d}\omega \tag{4.3}$$

gilt.

Wir wählen nun k linear unabhängige Vektoren $\{y_i\}_{i=1:k} \subseteq \mathbb{C}^n$ aus, die wir in der Matrix $Y_k := [y_i]_{i=1:k}$ zusammenfassen. Unsere gewünschte Matrix Q definieren wir nun durch Matrixmultiplikation vermöge

$$Q := X_k X_k^T Y_k. \tag{4.4}$$

Die Spalten von Q sind also gerade Linearkombinationen aus den Eigenvektoren. Sind

[2] Daher ist die Angabe $\mu \in I$ gerechtfertigt.

[3] Mit ι ist im Folgenden stets die imaginäre Einheit bezeichnet, also $\iota = \sqrt{-1}$.

[4] Diese Forderung ist wohldefiniert, da der Eigenschaften von A und B wegen alle Eigenwerte von $B^{-1}A$ reell sind.

dann $\{\phi_i\}_{i=1:k} \subseteq \mathbb{C}^n$ Vektoren die das transformierte Eigenwertproblem (4.2) lösen und ist $\Phi_k := [\phi_i]_{i=1:k}$ gegeben, so gilt $X_k = Q\Phi_k$.

In der Praxis sind natürlich weder die Anzahl der Eigenwerte noch die dazugehörigen Eigenvektoren bekannt. Während die Anzahl mit Hilfe einer passenden Filterfunktion bestimmt werden kann, verhilft die wegen (4.3) und (4.4) gültige Gleichheit

$$Q = -\frac{1}{2\pi\iota}\left(\int_\gamma G(\omega)\,\mathrm{d}\omega\right)\cdot Y \tag{4.5}$$

zur Berechnung der Eigenvektoren.

4.2 Beispiel

In diesem Abschnitt soll mit Hilfe eines einfachen Beispiels die eben erarbeitete Theorie verifiziert werden. Dafür betrachten wir die beiden Matrizen $A, B \in \mathbb{C}^{n,n}$, welche durch

$$A := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \text{ und } B := \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/4 \end{bmatrix}$$

gegeben sind. Durch einfaches Nachrechnen überprüft man, dass das verallgemeinerte Eigenwertproblem (4.1) durch Vektoren $x_1 \in \text{Span}_{\mathbb{C}}\{e_1\}$ mit zugehörigem Eigenwert $\lambda_1 = 0$, Vektoren $x_2 \in \text{Span}_{\mathbb{C}}\{e_2\}$ mit zugehörigem Eigenwert $\lambda_2 = 2$, sowie Vektoren $x_3 \in \text{Span}_{\mathbb{C}}\{e_3\}$ mit zugehörigem Eigenwert $\lambda_3 = -4$ gelöst wird.[5]

Wir wollen nun auf dem reellen Intervall $[-3, 3]$ die Eigenpaare bestimmen. Als Integrationskontur wählen wir daher den Kreis mit einem Radius von drei Längeneinheiten und dem Mittelpunkt im Ursprung. Das heißt, wir werden die Green-Funkion G über die Kurve

$$\gamma\colon [0, 2\pi] \to \mathbb{C}^n, \varphi \mapsto 3e^{\iota\varphi} \tag{4.6}$$

integrieren. Da nach Konstruktion keiner der Eigenwerte auf dem Graphen von γ liegt, ist G auf der gesamten Kontur wohldefiniert.

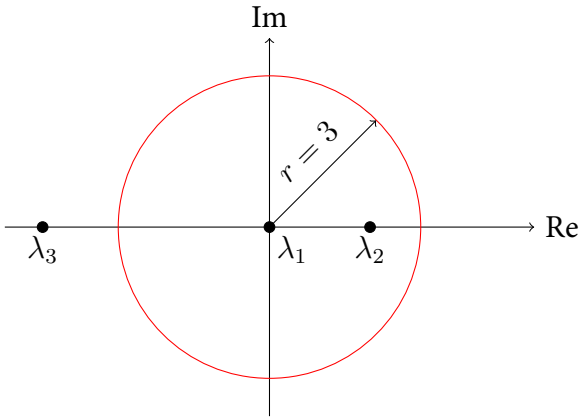

Abbildung 4.1: Skizze der Kurve γ in der komplexen Ebene.

Zunächst überprüfen wir die Identität (4.3). Mit $x_1 := \begin{bmatrix} \sqrt{2}\iota & 0 & 0 \end{bmatrix}^T$ und $x_2 := \begin{bmatrix} 0 & \sqrt{2}\iota & 0 \end{bmatrix}^T$

[5] Hier bezeichnen $e_1, e_2, e_3 \in \mathbb{C}^n$ die kanonischen Einheitsvektoren.

sind zwei passende Eigenvektoren für unser Eigenwertproblem gegeben, und in der Tat gilt

$$\begin{aligned}
-\frac{1}{2\pi\iota}\int_\gamma G(\omega)\,\mathrm{d}\omega &= -\frac{1}{2\pi\iota}\int_0^{2\pi} G(\gamma(\omega))\cdot\gamma'(\omega)\,\mathrm{d}\omega \\
&= -\frac{1}{2\pi\iota}\int_0^{2\pi} \begin{bmatrix} \frac{2}{3e^{\iota\omega}} & 0 & 0 \\ 0 & \frac{2}{3e^{\iota\omega}-2} & 0 \\ 0 & 0 & \frac{4}{3e^{\iota\omega}+4} \end{bmatrix} \cdot 3\iota e^{\iota\omega}\,\mathrm{d}\omega \\
&= \mathrm{diag}(-2,-2,0) \\
&= x_1x_1^T + x_2x_2^T.
\end{aligned}$$

Als Nächstes überprüfen wir, ob die Matrix $X_2 := \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ mit Hilfe der in Abschnitt 4.1 beschriebenen Methode rekonstruierbar ist.

Dort wird von uns zunächst die Konstruktion der Matrix Q aus (4.4) verlangt. Da wir bereits wissen, dass wir zwei Eigenvektoren finden wollen, wählen wir also beliebig eine Matrix $Y_2 \in \mathbb{C}^{3,2}$ vollen Ranges – die wir in diesem Beispiel mit $Y_2 := \begin{bmatrix} e_1 & e_1+e_2 \end{bmatrix}$ festlegen wollen – und multiplizieren diese von rechts an das Integral der Green-Funktion über die durch (4.6) definierte Kontur.[6] Wir erhalten somit

$$Q = -\frac{1}{2\pi\iota}\left(\int_\gamma G(\omega)\,\mathrm{d}\omega\right)\cdot Y_2 = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} -2 & -2 \\ 0 & -2 \\ 0 & 0 \end{bmatrix}.$$

Nun lösen wir das in (4.2) beschriebene Eigenwertproblem. Wir suchen also Paare $(\mu,\phi) \in \mathbb{C}\times\mathbb{C}^n$ die der Gleichung

$$A_Q\phi = \mu B_Q\phi$$

genügen. Durch Nachrechnen überzeugt man sich davon, dass dieses von allen Vektoren $\phi_1 \in \mathrm{Span}_{\mathbb{C}}\{e_1\}$ mit dem Eigenwert $\mu_1 = 0$ sowie allen Vektoren $\phi_2 \in \mathrm{Span}_{\mathbb{C}}\{-e_1+e_2\}$ mit dem Eigenwert $\mu_2 = 2$ gelöst wird.

Die soeben ermittelten Eigenwerte stimmen also schon mit den Eigenwerten unseres betrachteten Problems überein. Wählen wir schließlich die Matrix

$$\Phi_2 := \begin{bmatrix} \phi_1 & \phi_2 \end{bmatrix} := \begin{bmatrix} -\iota/\sqrt{2} & \iota/\sqrt{2} \\ 0 & -\iota/\sqrt{2} \end{bmatrix},$$

[6] Es mag den Leser verwundern, wie bei Unkenntnis über die Anzahl $k \in \mathbb{N}$ der zu findenden Eigenvektoren die Spaltenzahl von Y_k korrekt zu ermitteln ist. Eine gängige Methode ist zum Beispiel das Konstruieren einer passenden Filterfunktion, welche die nicht erwünschten Eigenwerte aussortiert (Vgl. Abschnitt 4.3.2).

mit den Eigenvektoren ϕ_1 und ϕ_2, so erhalten wir wie gewünscht

$$Q\Phi_2 = \begin{bmatrix} -2 & -2 \\ 0 & -2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -\iota/\sqrt{2} & \iota/\sqrt{2} \\ 0 & -\iota/\sqrt{2} \end{bmatrix} = \begin{bmatrix} \sqrt{2}\iota & 0 \\ 0 & \sqrt{2}\iota \\ 0 & 0 \end{bmatrix} = X_2.$$

Bemerkung. An dieser Stelle sei darauf hingewiesen, dass die Matrix Φ_2 aus kosmetischen Gründen gerade so gewählt wurde, dass die Kalkulation am Ende X_2 ergibt und somit alles hübsch anzusehen ist. Der Wahl einer anderen Matrix Φ_2' mit skalaren Vielfachen der hier gewählten Eigenvektoren steht selbstverständlich nichts im Wege und führt freilich zu einer anderen Matrix X_2', deren Spalten ebenfalls zulässige Eigenvektoren enthält.

4.3 Realisierung des FEAST-Algorithmus'

Nun, da die mathematischen Ideen diskutiert und durch ein Bespiel verifiziert worden sind, widmen wir uns der Implementation des FEAST-Algorithmus' in MATLAB. Wir orientieren uns dabei an der Vorgehensweise aus dem Beispiel in Abschnitt 4.2. Auch die Notation wird beibehalten.

4.3.1 Naive Implementation

Zunächst wählen wir also für $k \in \mathbb{N}$ eine vollrangige Zufallsmatrix Y_k, um die Transformationsmatrix Q zu ermitteln. Dazu führen wir die GREEN-Funktion

$$\text{funGreen} = @(\text{w})\ \text{inv}(\ \text{w.*B-A}\ )$$

ein und wählen eine passende Integrationskontur C. Um es einfach zu halten, wird diese in der Implementation die Menge

$$C = \left\{ z \in \mathbb{C} : \left\| z - \frac{\lambda_{\text{max}} - \lambda_{\text{min}}}{2} \right\|_1 = 1 \right\}$$

sein, wobei λ_{min} und λ_{max} die vorgegebenen Intervallgrenzen bezeichnen. Die entsprechenden Quadraturpunkte werden im Quellcode durch die Variablen

$$\text{n0},\ \text{n1},\ \text{n2},\ \text{n3}$$

repräsentiert.

Ist Q bestimmt, benutzen wir die MATLAB interne Eigenwertfunktion

$$\text{eig}(\text{Aq}, \text{Bq})$$

um die Ritz-Paare des transformierten Problems zu berechnen. Schließlich berechnen wir die Eigenwerte und Eigenvektoren des ursprünglichen Problems. Die Matrix Φ_k wird im Quelltext mit Vq bezeichnet.

Wir wollen annehmen, dass der Input des Users die Voraussetzungen des Problems (4.1) erfüllt und die Anzahl $k \in \mathbb{N}$ der Eigenwerte im Intervall $[\lambda_{\min}, \lambda_{\max}]$ bekannt ist. Eine naive Implementation könnte daher wie folgt aussehen:

```
%% Eingabe: Matrizen A, B des Eigenwertproblems,
%%          Intervallgrenzen lmin, lmax,
%%          Anzahl k der erwarteten Eigenwerte
%% Ausgabe: Matrix V mit Eigenvektoren,
%%          Matrix D mit Eigenwerten auf der Diagonalen

%% Funktionsaufruf: [V, D] = feast[A, B, lmin, lmax, k]

function[V, D] = feast(A, B, lmin, lmax, k);

n = length(A);

% erzeuge Zufallsmatrix
Y = randn(n, k);

% generiere Green-Funktion
funGreen = @(w) inv( w.*B - A );

% berechne die Integrationsknotenpunkte
n0 = lmax;
n1 = (lmin + lmax)/2 + (lmax - (lmin + lmax)/2)*i;
n2 = lmin;
n3 = (lmin + lmax)/2 - (lmax - (lmin + lmax)/2)*i;

% Integrationskontur
C = [n1 n2 n3];

% berechne Transformationsmatrix Q
Q = (-1/(2*pi*i))*integral(funGreen, n0, n0,...
    'Waypoints', C, 'ArrayValued', true) * Y;

% berechne Ritz-Paare des transformierten Problems
```

```
33 Aq = Q'*A*Q; Bq = Q'*B*Q;
34 [Vq, D] = eig(Aq, Bq);
35
36 % berechne die gesuchten Eigenvektoren
37 V = Q*Vq;
```

Algorithmus 4.1: Naive FEAST Implementation.

Überprüfen wir die Funktionalität mit den Matrizen A und B aus dem Abschnitt 4.2, erhalten wir folgende Ausgabe:

```
>> [V, D] = feast(A, B, -3, 3, 2)

V =

   -1.4142 - 0.0000i   0.0000 + 0.0000i
    0.0000 + 0.0000i   1.4142 - 0.0000i
   -0.0000 + 0.0000i  -0.0000 + 0.0000i


D =

    0.0000         0
         0    2.0000

>>
```

Die vorgestellte Umsetzung des FEAST-Algorithmus' lässt sich ohne Weiteres auf dynamische Matrixdimensionen übertragen. Da sich mit zunehmender Dimension unter Umständen auch die Kardinalität des Spektrums und die Anzahl der Eigenräume ändert, sollten gegebenenfalls Optimierungsmaßnahmen getroffen werden, wie etwa die Wahl der Knotenpunkte für die Konturintegration.

Man könnte sich nun die Frage stellen, warum man anstatt (4.1) nicht einfach das Problem

$$B^{-1}Ax = \lambda x$$

zu lösen versucht. Der positiven Definitheit von B wegen, wäre diese Umformulierung möglich und mit bekömmlicheren Verfahren lösbar.

Nun ist aber das Invertieren von Matrizen bekanntermaßen numerisch eher ungünstig, da man unter Umständen mit hohem Rechenaufwand und dem Verlust von Stabilität bezahlen muss. Ein erneuter Blick auf (4.3) oder (4.4) stimmt demnach benklich, da sowohl mit

der Integration als auch mit dem Berechnen der GREEN-Funktion gewisse Risiken eingegangen werden.

Man muss sich also die Frage stellen, ob eine Implementation des FEAST-Algorithmus ohne Inversion und Integration möglich ist.

4.3.2 Rational Krylov Toolbox

Wie in der Einleitung versprochen, soll abschließend ein numerischer Werkzeugkoffer präsentiert werden, der es ermöglicht den FEAST-Algorithmus elegant und ohne die eben beschriebenen Bedenklichkeiten umzusetzen.

Mit der „*Rational Krylov Toolbox for MATLAB*“ [**RKT**] stellen deren Entwickler MARIO BERLJAFA und STEFAN GÜTTEL eine Bibliothek zur Verfügung, die es ermöglicht, gängige KRYLOV-Unterraum-Verfahren unkompliziert zu implementieren. Bevor wir dieses Tool genauer betrachten, wollen wir uns klarmachen, warum die Anwendung eines solchen Verfahrens möglich ist. Dazu begnügen wir uns an dieser Stelle mit einer Plausibilitätsbetrachtung.

Studieren wir erneut die in (4.3) formulierte Identität

$$X_k X_k^T := \sum_{i=1}^{k} x_i x_i^T = -\frac{1}{2\pi\iota} \int_\gamma G(\omega)\, \mathrm{d}\omega,$$

fällt sodann auf, dass die Abbildung

$$P\colon \mathbb{C}^n \to \mathbb{C}^n, v \mapsto Pv := \left(-\frac{1}{2\pi\iota} \int_\gamma G(\omega)\, \mathrm{d}\omega \right) v$$

Vektoren in den von den Eigenvektoren $(x_i)_{1:k}$ aufgespannten Unterraum abbildet. Man kann sogar zeigen, dass die Abbildung eine B-orthogonale Projektion ist. Die Idee, den FEAST-Algorithmus per Projektionsverfahren umzusetzen, liegt demnach gar nicht so fern.

Eine mögliche Implementation stellen die Autoren in ihren Beispielen auf der Seite

`http://www.guettel.com/rktoolbox/examples/html/example_feast.html`

vor:

```
1 % Search space basis V of dimension m.
2 m = 10;
```

```
3  V = randn(N, m);
4
5  for iter = 1:8
6    % Apply rational filter to V.
7    V = r(A, V);
8    % Compute and sort Ritz pairs.
9    Am = V'*A*V; Bm = V'*B*V;
10   [W, D]    = eig(Am, Bm);
11   [D, ind] = sort(diag(D)); W = W(:, ind);
12   % B-normalize W.
13   nrm = sqrt(diag(W'*Bm*W)); W = W/diag(nrm);
14   % Form approximate eigenvectors.
15   V = V*W;
16   % Check residuals and number of eigenpairs inside
17   % search iterval.
18   Di = diag(D(lmin < D & D < lmax));
19   Vi = V(:, lmin < D & D < lmax);
20   resid(iter) = norm(A*Vi - B*Vi*Di, 'fro');
21   nrvec(iter) = size(Vi, 2);
22 end
```

Algorithmus 4.2: FEAST-Implementation mit Unterraumprojektion.

Diese beginnt mit der Initialisierung und dem Festlegen eines Suchraums, der durch die Spalten der Matrix V aufgespannt wird. Hierbei ist N die Dimension der Matrizen A und B und m die Anzahl der erwarteten Eigenpaare.[7]

In der folgenden Iterationsschleife wird der Eigenraum approximiert. Besonderes Augenmerk muss auf die Filterfunktion r gelegt werden. Diese ist gerade so konzipiert, dass Argumente innerhalb des Suchintervalls in einen Bereich $(1-\varepsilon, 1+\varepsilon)$ abbgebildet werden. Der Wert $\varepsilon > 0$ ist hierbei vom Input abhängig und kann stark variieren. Argumente außerhalb des Intervalls werden in ähnlicher Manier auf einen Bereich $(-\delta, \delta)$ abgebildet – für ein entsprechendes $\delta > 0$.
Der Toolbox-interne Aufruf r(A,V) ermöglicht nun das Berechnen der Wirkung des Filters auf den Suchraum V.

Schlussendlich löst der Algorithmus auch hier das transformierte Eigenwertproblem und ermittelt daraus approximativ die gesuchten Eigenpaare.

[7] Auch in diesem Beispiel ist diese Anzahl der Eigenpaare im Voraus bekannt. Trotzdem wird ein Filter eingesetzt.

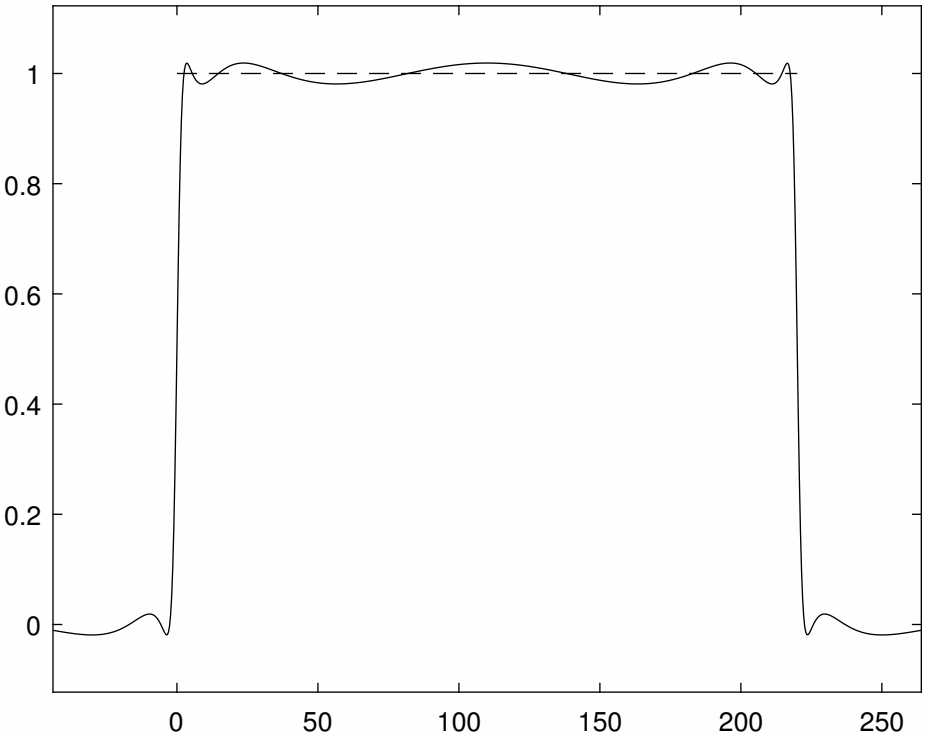

Abbildung 4.2: Filterfunktion r auf dem Intervall [0,220] wirkend.

Kapitel 5

Conclusio

Finale

Anhang A

yolo

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Literatur

[Bar15] Marc van Barel. “Designing rational filter functions for solving eigenvalue problems by contour integration.” In: (2015).

[Ber16] Mario Berljafa. “Rational Krylov Toolbox for MATLAB.” In: (2016).

[Lie13] Jörg Liesen. *Krylov Subspace Methods. Principles and Analysis.* 2013.

[Pol09] Eric Polizzi. “Density-matrix-based algorithm for solving eigenvalue problems.” In: (2009).