SEMINARARBEIT

Eigenwertberechnung mithilfe des Lanczos-Verfahrens

ausgeführt am

Institut für
Analysis und Scientific Computing
TU Wien

unter der Anleitung von

Lothar Nannen

durch

Göth Christian **Moik Matthias** **Sallinger Christian**

Wien, am 17. April 2021

Inhaltsverzeichnis

1 Einleitung

Das folgende Projekt beschäftigt sich mit der numerischen Berechnung von Eigenwerten mithilfe des Lanczos-Verfahrens. Die Arbeit orientiert sich dabei in großen Teilen an [2] und [3]. Zur Motivation betrachten wir das Eigenwertproblem des negativen Laplace-Operators mit homogenen Neumann-Randbedingungen auf einem zweidimensionalen Gebiet Ω.

1.1 Problemstellung

Wir wiederholen zunächst die Definition eines Eigenwertproblems.

Definition 1.1. *Sei Ω eine offene und beschränkte Menge. Das Paar $(\lambda, u) \in \mathbb{C} \times H^2(\Omega) \setminus \{0\}$[1] heißt Eigenpaar, wenn es eine Lösung des Eigenwertproblems*

$$\begin{cases} -\Delta u = \lambda u & \text{in } \Omega, \\ \frac{\partial u}{\partial \nu} = 0 & \text{auf } \partial\Omega, \end{cases} \tag{1}$$

ist, wobei ν der äußere Normalenvektor an $\partial\Omega$ sei.

Um die aus dem numerischen Verfahren gewonnen Eigenwerte mit den tatsächlichen Eigenwerten vergleichen zu können, wählen wir als Gebiet das Rechteck $\Omega := (0, a) \times (0, b)$ mit $a, b \in \mathbb{R}^+$, auf dem sich diese analytisch berechnen lassen.

1.2 Analytische Lösung für ein Rechteck

Die Eigenwerte und Eigenfunktionen des oben genannten Problems werden zunächst mit einem Separationsansatz hergeleitet. Sei dazu $u(x, y) = v(x)w(y)$, die Differentialgleichung ergibt dann

$$-\Delta u(x, y) = -v''(x)w(y) - v(x)w''(y) = \lambda v(x)w(y)$$

Dividiert man formal durch $v(x)w(y)$, erhält man

$$\begin{aligned} & -\frac{v''(x)}{v(x)} - \frac{w''(y)}{w(y)} = \lambda \\ \Leftrightarrow \quad & -\frac{v''(x)}{v(x)} = \lambda + \frac{w''(y)}{w(y)}. \end{aligned}$$

Da die linke Seite nur von x abhängt und die rechte nur von y, müssen beide konstant sein. Es gibt also ein $\kappa_v^2 \in \mathbb{R} \setminus \{0\}$, sodass

$$-\frac{v''(x)}{v(x)} = \kappa_v^2, \quad \forall\, x \in (0, a).$$

Dies liefert uns die gewöhnliche Differentialgleichung für v

$$v''(x) = -\kappa_v^2 v(x)$$

mit der Lösung

$$v(x) = c_1 \sin(\kappa_v x) + c_2 \cos(\kappa_v x).$$

Analog erhält man $w(y) = c_3 \sin(\kappa_w y) + c_4 \cos(\kappa_w y)$, wobei $\lambda = \kappa_v^2 + \kappa_w^2$ gelten muss.

[1] $H^k(\Omega)$ bezeichnet hier den Sobolevraum $W^{k,2}(\Omega)$

Nun lassen wir noch die Neumann-Randbedingungen einfließen. Mit $\nabla u(x,y) = \Big(v'(x)w(y), v(x)w'(y)\Big)^\top$ erhalten wir für den linken Rand

$$\frac{\partial u}{\partial \nu}(0,y) = -v'(0)w(y) = \kappa_v(-c_1\cos(\kappa_v\cdot 0) + c_2\underbrace{\sin(\kappa_v\cdot 0)}_{=0})w(y) \stackrel{!}{=} 0$$

Es muss also $c_1 = 0$ gelten. Analog schließt man bei Betrachtung des unteren Randes auch $c_3 = 0$.

Für den rechten Rand gilt nun

$$\begin{aligned} &\frac{\partial u}{\partial \nu}(a,y) = v'(a)w(y) = \kappa_v c_2 \sin(\kappa_v\cdot a)w(y) \stackrel{!}{=} 0 \\ \Rightarrow\quad &\kappa_v = \frac{n\pi}{a}, \quad n\in\mathbb{N}. \end{aligned}$$

Wiederum analog schließt man bei Betrachtung des oberen Randes $\kappa_w = \frac{m\pi}{b}, m\in\mathbb{N}$. Wir wählen $c_2 = c_4 := 1$ und halten fest:

Satz 1.2. *Eigenfunktionen für das Eigenwertproblem* (1) *auf dem Gebiet* $\Omega := (0,a)\times(0,b)$ *sind gegeben durch*

$$u_{n,m}(x,y) = \cos(\frac{n\pi}{a}x)\cos(\frac{m\pi}{b}y), \quad n,m\in\mathbb{N}$$

und die zugehörigen Eigenwerte durch

$$\lambda_{n,m} = \pi^2(\frac{n^2}{a^2} + \frac{m^2}{b^2}), \quad n,m\in\mathbb{N}.$$

Beweis. Nachrechnen. □

Bemerkung 1.3. *Man rechnet leicht nach, dass die Eigenfunktionen aus Satz 1.2 paarweise orthogonal sind. Wählt man einen geeigneten Funktionenraum, lässt sich sogar zeigen, dass es sich bei den Eigenfunktionen um ein vollständiges Orthogonalsystem handelt.*

1.3 Numerischer Lösungsansatz

Das Eigenwertproblem (1) kann durch Multiplizieren mit $v\in H^1(\Omega)$ und Integrieren auf schwache Form gebracht werden (beim Anwenden des Satzes von Gauß verschwindet der Randterm aufgrund der homogenen Randbedingungen). Gesucht sind also Lösungen $(\lambda,u)\in\mathbb{C}\times H^1(\Omega)\setminus\{0\}$, sodass

$$\int_\Omega \nabla u\nabla v \,\mathrm{d}x = \lambda\int_\Omega uv\,\mathrm{d}x, \quad \forall\, v\in H^1(\Omega).$$

Dieses Problem kann mittels der Finite-Elemente-Methode auf ein endlichdimensionales verallgemeinertes Eigenwertproblem der Form

$$Ax_h = \lambda_h B x_h \tag{2}$$

gebracht werden, mit $A,B\in\mathbb{R}^{N\times N}$ und symmetrisch.

Lemma 1.4. *Mit* $\rho_h\in\mathbb{R}$, *welches kein Eigenwert ist, und* $\lambda_h = \frac{1}{\mu_h} + \rho_h$ *ist das verallgemeinerte Problem* (2) *äquivalent zu*

$$(A-\rho_h B)^{-1}Bx_h = \mu_h x_h \tag{3}$$

Beweis. Der Einfachheit halber lassen wir den Index h weg. Die Umformungen

$$\begin{aligned} & Ax = \lambda Bx = (\frac{1}{\mu} + \rho)Bx \\ \Leftrightarrow \quad & (A - \rho B)x = \frac{1}{\mu}Bx \\ \Leftrightarrow \quad & \mu x = (A - \rho B)^{-1}Bx \end{aligned}$$

zeigen die Äquivalenz □

Bemerkung 1.5. *Die Matrix $(A-\rho_h B)^{-1}B$ im Eigenwertproblem* (3) *ist zwar Produkt zweier symmetrischer Matrizen, aber i. A. nicht mehr symmetrisch. Dies wird im Laufe des Projektes noch wichtig sein, da das Lanczos-Verfahren nur für hermitesche Matrizen funktioniert.*

Die Diskretisierung durch die Finite-Elemente-Methode liefert uns Matrizen A und B mit sehr großer Dimension N. Daher ist es sinnvoll, ein Verfahren zu entwickeln, das uns für sehr große Matrizen eine gute Approximation zumindest an extremale Eigenwerte liefert. Das Lanczos-Verfahren wird dies unter Zuhilfenahme des QR-Verfahrens bewerkstelligen.

2 Das QR-Verfahren

2.1 Idee und Algorithmus

Das QR-Verfahren ist eine Methode zur Eigenwertbestimmung einer Matrix $A \in \mathbb{C}^{n\times n}$ mit Eigenwerten $\lambda_1, ..., \lambda_n$. Dabei wird mithilfe der QR-Zerlegung iterativ eine Folge $(A^{(t)})_{t\in\mathbb{N}}$ definiert. Diese ist durch

$$A^{(0)} \coloneqq A, \quad A^{(t+1)} \coloneqq R^{(t)}Q^{(t)} \tag{4}$$

gegeben, wobei $A^{(t)} = Q^{(t)}R^{(t)}$ die QR-Zerlegung von $A^{(t)}$ ist. Für alle $t \in \mathbb{N}$ hat $A^{(t)}$ die gleichen Eigenwerte wie A und die Folge $(A^{(t)})_{t\in\mathbb{N}}$ konvergiert gegen eine obere Dreiecksmatrix. Das heißt, die Diagonaleinträge von $A^{(t)}$ konvergieren gegen die Eigenwerte von A.

Algorithmus 1 Basisalgorithmus QR-Verfahren

Input: $A \in \mathbb{C}^{n\times n}$

1: **while** Abbruchbedingung nicht erfüllt **do**
2: Berechne die QR-Zerlegung $A = QR$
3: $A \coloneqq RQ$
4: **end while**

Output: A wird überschrieben mit einer Matrix, die eine Approximation einer oberen Dreiecksmatrix ist und die gleichen Eigenwerte wie die ursprüngliche Matrix A besitzt

Die Konvergenz dieses Algoritmus kann unter gewissen Voraussetzungen gezeigt werden. Wir zitieren hier nur den Satz und verweisen für den Beweis auf [3], Satz 9.15.

Satz 2.1 (Konvergenz des QR-Verfahrens)**.** *Sei $A \in \mathbb{C}^{n\times n}$ eine diagonalisierbare Matrix mit Eigenwerten $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$ und $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| > 0$. Weiters sei $V \in \mathbb{C}^{n\times n}$ die Matrix, die die Eigenvektoren $v_1, \ldots v_n$ zu den jeweiligen Eigenwerten als Spalten enthält. Zudem existiere eine untere normierte Dreiecksmatrix L und eine obere Dreiecksmatrix U, sodass $V^{-1} = LU$.*
Dann konvergieren die Hauptdiagonaleinträge der Matrizen $A^{(t)}$, die in Algorithmus 1 definiert sind, gegen die Eigenwerte von A.

2.2 Beschleunigung der Konvergenz

Wir betrachten die Matrizen $M^{(t)} := \Lambda^t L \Lambda^{-t}$, $t \in \mathbb{N}$, die im Beweis von Satz 2.1 vorkommen. Diese haben die Form

$$M^{(t)} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ m_{21}\left(\frac{\lambda_2}{\lambda_1}\right)^t & 1 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ m_{n1}\left(\frac{\lambda_n}{\lambda_1}\right)^t & \dots & m_{n(n-1)}\left(\frac{\lambda_n}{\lambda_{n-1}}\right)^t & 1 \end{pmatrix}.$$

Da die Eigenwerte der Größe nach sortiert sind, erkennt man, dass diese Matrixfolge gegen die Einheitsmatrix konvergiert. Dies zieht schlussendlich die Konvergenz von $(A^{(t)})_{t\in\mathbb{N}}$ gegen eine obere Dreiecksmatrix nach sich. Die Einträge konvergieren schneller gegen 0, wenn die Brüche kleiner sind. Das versuchen wir zu erreichen, indem wir einen shift durchführen.
Für einen Eigenwert λ einer Matrix $A \in \mathbb{C}^{n\times n}$ gilt

$$\ker(A - \lambda\,\mathrm{id}) \neq \{0\} \Leftrightarrow \ker((A - \rho\,\mathrm{id}) - (\lambda - \rho)\,\mathrm{id}) \neq \{0\} \Leftrightarrow (\lambda - \rho) \text{ ist Eigenwert von } (A - \rho\,\mathrm{id}).$$

Die Wahl des shifts als $a_{nn}^{(t)}$ macht also Sinn, da dieser Eintrag für $t \to \infty$ gegen λ_n konvergiert. Dieser shift wird auch Rayleigh-Quotienten-Shift genannt. Nach der QR-Zerlegung von $A^{(t)}$ setzt man $A^{(t+1)} := R^{(t)}Q^{(t)} + \rho\,\mathrm{id}$, um wieder eine Matrix zu erhalten, die die gleichen Eigenwerte wie $A^{(t)}$ und somit auch wie A besitzt. Je nachdem, wie genau λ_n approximiert werden soll, wählt man eine Toleranz, die von den Einträgen $a_{ni}^{(t)}$, $i = 1, \dots, n-1$ unterschritten werden muss, bevor analoge Schritte für den zweitkleinsten Eigenwert durchgeführt werden. Dessen Näherung ist der Eintrag $a_{(n-1)(n-1)}^{(t)}$.

Algorithmus 2 QR-Verfahren mit Rayleigh-Quotienten-Shift

Input: $A \in \mathbb{C}^{n\times n}$

```
for i = n, ..., 2 do
    while |a_{i,i-1}| > tol do
        ρ = a_{i,i}
        Berechne die QR-Zerlegung A − ρ id = QR
        A = RQ + ρ id
    end while
end for
```

Output: A wird überschrieben mit einer Matrix, die eine Approximation einer oberen Dreiecksmatrix ist und die gleichen Eigenwerte wie die ursprüngliche Matrix A besitzt

Eine weitere Möglichkeit, den shift zu wählen, ist, sich die Eigenwerte ρ_1, ρ_2 der 2×2 Matrix rechts unten von $A^{(t)}$ zu berechnen und anschließend den Eigenwert, der betragsmäßig näher an $a_{nn}^{(t)}$ liegt, zu wählen. Die Konvergenzgeschwindigkeit wird dadurch nochmals verbessert (vgl. [3], S. 114). Dieser shift wird auch Wilkinson-Shift genannt.

Algorithmus 3 QR-Verfahren mit Wilkinson-Shift

Input: $A \in \mathbb{C}^{n\times n}$

for $i = n, \dots, 2$ **do**
  while $a_{i(i-1)} > tol(a_{(i-1)(i-1)} + a_{ii})$ **do**
    Berechne die Eigenwerte ρ_1, ρ_2 von $\begin{pmatrix} a_{(i-1)(i-1)} & a_{(i-1)i} \\ a_{i(i-1)} & a_{ii} \end{pmatrix}$
    if $|\rho_1 - a_{ii}| < |\rho_2 - a_{ii}|$ **then**
      $\rho = \rho_1$
    else
      $\rho = \rho_2$
    end if
    Berechne die QR-Zerlegung $A - \rho\,\mathrm{id} = QR$
    $A = RQ + \rho\,\mathrm{id}$
  end while
end for

Output: A wird überschrieben mit einer Matrix, die eine Approximation einer oberen Dreiecksmatrix ist und die gleichen Eigenwerte wie die ursprüngliche Matrix A besitzt

Eine weitere Beschleunigung erreichen wir bei der QR-Zerlegung, wenn die Matrix A Hessenbergform hat. Auf diesen Fall werden wir im nächsten Abschnitt eingehen.

2.3 Die QR-Zerlegung für Hessenberg-Matrizen

Den Hauptaufwand beim QR-Verfahren bildet die QR-Zerlegung die in jedem Iterationsschritt durchgeführt werden muss. Der Aufwand einer QR-Zerlegung mithilfe von Householdermatrizen wächst kubisch mit der Dimension der Matrix (vgl. [3], S.78). Nun gibt es für sogenannte Hessenberg-Matrizen einen wesentlich effizienteren Algorithmus, bei dem statt Householdermatrizen andere unitäre Matrizen verwendet werden.

Definition 2.2 (Hessenberg-Matrix)**.** *Eine Matrix $A = (a_{ij})_{i,j=1}^n \in \mathbb{C}^{n\times n}$ hat (obere) Hessenbergform, falls $a_{ij} = 0$ für alle $j = 1, \dots, n$ und $i = j + 2, \dots, n$. Eine Hessenberg-Matrix hat also fast obere Dreiecksform, nur in der unteren Nebendiagonale sind noch zusätzliche nicht-null Einträge.*

Definition 2.3 (Givens-Rotation)**.** *Eine Givens-Rotation $G \in \mathbb{C}^{n\times n}$ ist eine Matrix der Form*

$$G(j,k,c,s) = \left(\begin{array}{c|c|c} \begin{matrix} 1 & & \\ & \ddots & \\ & & 1 \end{matrix} & & \\ \hline & \begin{matrix} \overline{c} & & \overline{s} \\ & 1 & \\ & & \ddots \end{matrix} & \\ & \begin{matrix} 1 & \\ -s & c \end{matrix} & \\ \hline & & \begin{matrix} 1 & & \\ & \ddots & \\ & & 1 \end{matrix} \end{array}\right)$$

mit $|c|^2 + |s|^2 = 1$. Der innere Block reicht dabei von der j-ten bis zur k-ten Zeile sowie Spalte.

Wir wollen nun für die QR-Zerlegung geeignete Givens-Rotationen verwenden. Zunächst sehen wir, dass Givens-Rotationen unitär sind, da

$$\begin{pmatrix} \overline{c} & \overline{s} \\ -s & c \end{pmatrix} \begin{pmatrix} c & -\overline{s} \\ s & \overline{c} \end{pmatrix} = \begin{pmatrix} |c|^2+|s|^2 & 0 \\ 0 & |c|^2+|s|^2 \end{pmatrix}.$$

Wir werden es mit Givens-Rotationen der Form

$$G(j,j+1,c,s) = \left(\begin{array}{ccc|cc|ccc} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ \hline & & & \overline{c} & \overline{s} & & & \\ & & & -s & c & & & \\ \hline & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{array}\right)$$

zu tun haben.

Wie man unmittelbar sieht, hat die Multiplikation einer Givens-Rotation der obigen Form von links an eine Matrix $A \in \mathbb{C}^{n\times n}$ nur in der j-ten sowie $j+1$-ten Zeile von A eine Auswirkung. Dort werden für $k = 1,\ldots,n$ die Einträge $a_{j,k}$ sowie $a_{j+1,k}$ ersetzt durch $a_{j,k}\overline{c} + a_{j+1,k}\overline{s}$ sowie $-a_{j,k}s + a_{j+1,k}c$. Dies können wir nun dazu verwenden, um bei einer Hessenberg-Matrix die Einträge unter der Diagonale zu eliminieren und dadurch eine obere Dreiecksmatrix erhalten.

Dazu müssen wir die Gleichung $-a_{j,j}s + a_{j+1,j}c = 0$ unter der Nebenbedingung $|c|^2 + |s|^2 = 1$ lösen. Dieses Problem hat zwei Lösungen, die sich, wie man sich leicht überzeugen kann, nur durch das Vorzeichen unterscheiden. Eine Lösung ist gegeben durch

$$\begin{pmatrix} c \\ s \end{pmatrix} = \mathbf{rot}(a_{j,j}, a_{j+1,j}) := \frac{1}{\sqrt{|a_{j,j}|^2 + |a_{j+1,j}|^2}} \begin{pmatrix} a_{j,j} \\ a_{j+1,j} \end{pmatrix}.$$

Zur Vermeidung von over- bzw. underflow verwendet man in der Praxis jedoch die äquivalenten Formeln

$$c = \frac{a_{j,j}/|a_{j,j}|}{\sqrt{1+|\tau|^2}},\ s = \frac{\tau}{\sqrt{1+|\tau|^2}},\ \tau = \frac{a_{j+1,j}}{|a_{j,j}|}, \quad \text{für } |a_{j,j}| \geq |a_{j+1,j}|,$$
$$c = \frac{\tau}{\sqrt{1+|\tau|^2}},\ s = \frac{a_{j+1,j}/|a_{j+1,j}|}{\sqrt{1+|\tau|^2}},\ \tau = \frac{a_{j,j}}{|a_{j+1,j}|}, \quad \text{für } |a_{j,j}| < |a_{j+1,j}|.$$

Eine Hessenberg-Matrix $A \in \mathbb{C}^{n\times n}$ können wir dann durch Anwendung von $n-1$ Givens-Rotationen auf die Form einer oberen Dreiecksmatrix bringen. Dies können wir iterativ durchführen, dazu definieren wir $A^{(0)} := A$ und für $j = 1,\ldots,n-1$

$$A^{(j)} := G(j,j+1,c_j,s_j)A^{(j-1)} \quad \text{mit } (c_j,s_j)^\top = \mathbf{rot}(a_{j,j}, a_{j+1,j}).$$

Die spezielle Struktur der Hessenberg-Matrix können wir nun verwenden, um Rechenschritte zu sparen: in der j-ten Zeile sind wegen unserer Wahl von c und s die ersten $j-1$ Spalteneinträge bereits 0 (erst recht in der $j+1$-ten Zeile). Daher müssen wir die Anwendung der Givens-Rotation erst ab dem j-ten Spalteneintrag realisieren. Wir skizzieren die Vorgehensweise für $n = 4$, wobei $+$ für beliebige Einträge der Matrix und $*$ für die im jeweiligen Schritt durch Anwendung der Givens-Rotation veränderten Einträge steht.

$$A = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ & + & + & + \\ & & + & + \end{pmatrix} \xrightarrow{j=1} \begin{pmatrix} * & * & * & * \\ & * & * & * \\ & + & + & + \\ & & + & + \end{pmatrix} \xrightarrow{j=2} \begin{pmatrix} + & + & + & + \\ & * & * & * \\ & & * & * \\ & & + & + \end{pmatrix} \xrightarrow{j=3} \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & * & * \\ & & & * \end{pmatrix} = R$$

Um schließlich noch die Matrix Q aus der QR-Zerlegung zu erhalten, bemerken wir zunächst, dass die Multiplikation zweier unitärer Matrizen wieder eine unitäre Matrix liefert. Durch unseren Algorithmus erhalten wir

$$R := A^{(n-1)} = G(n-1, n, c_{n-1}, s_{n-1}) \cdots G(1, 2, c_1, s_1) A.$$

Die Multiplikation der Givens-Rotationen gibt wieder eine unitäre Matrix, definieren wir also

$$Q^* := G(n-1, n, c_{n-1}, s_{n-1}) \cdots G(1, 2, c_1, s_1), \tag{5}$$

erhalten wir Q durch Adjungieren des Produktes der Givens-Rotationen, da dann

$$QR = A.$$

Bei Multiplikation der $l+1$-ten Givens-Rotation mit dem Produkt der l vorhergegangenen können wir wieder verwenden, dass nur in zwei Zeilen eine Änderung passiert. Da das Produkt der Givens-Rotationen jedoch keine Hessenberg-Matrix ist, müssen wir die Rechnung in allen Spalten durchführen.

Algorithmus 4 QR-Zerlegung für Hessenberg-Matrizen

Input: $A \in \mathbb{C}^{n \times n}$

$B = \mathrm{id} \in \mathbb{C}^{n \times n}$
for $i = 1, \ldots, n-1$ **do**
  $\begin{pmatrix} c_i \\ s_i \end{pmatrix} = \frac{1}{\sqrt{|a_{i,i}|^2 + |a_{i+1,i}|^2}} \begin{pmatrix} a_{i,i} \\ a_{i+1,i} \end{pmatrix}$
  for $j = i, \ldots, n$ **do**
    $\begin{pmatrix} a_{i,j} \\ a_{i+1,j} \end{pmatrix} = \begin{pmatrix} \overline{c_i} & \overline{s_i} \\ -s_i & c_i \end{pmatrix} \begin{pmatrix} a_{i,j} \\ a_{i+1,j} \end{pmatrix}$
  end for
  for $j = 1, \ldots, n$ **do**
    $\begin{pmatrix} b_{i,j} \\ b_{i+1,j} \end{pmatrix} = \begin{pmatrix} \overline{c_i} & \overline{s_i} \\ -s_i & c_i \end{pmatrix} \begin{pmatrix} b_{i,j} \\ b_{i+1,j} \end{pmatrix}$
  end for
end for

Output: A wird mit oberer Dreiecksmatrix ($= R$) überschrieben, B ist das Produkt der Givens-Rotationen und $Q = B^*$ ist die gewünschte unitäre Matrix

Wie man hier schnell erkennt, hat eine QR-Zerlegung einer Hessenberg-Matrix mithilfe von Givens-Rotationen nun Aufwand $\mathcal{O}(n^2)$, ist also um eine ganze $n-$Potenz weniger aufwändig als die Zerlegung mit Householdermatrizen.

Das QR-Verfahren für Hessenberg-Matrizen kann nun noch weiter vereinfacht werden, wenn wir die Matrix Q nicht explizit aufstellen. Man kann die obige QR-Zerlegung noch dahingehend abändern, dass nicht die unitäre Matrix Q und die Dreiecksmatrix R zurückgegeben werden, sondern direkt das Produkt RQ. Dazu verwenden die Gleichheit in (5), wonach wir

$$Q = G^*(1,2,c_1,s_1)\cdots G^*(n-1,n,c_{n-1},s_{n-1})$$

erhalten. Nun hat die Multiplikation der Givens-Rotation $G^*(j,j+1,c_j,s_j)$ von rechts an eine Matrix $A \in \mathbb{C}^{n\times n}$ nur in der $j-$ten sowie $j+1-$ten Spalte eine Auswirkung. Hierbei werden nun für $k = 1,\ldots,n$ die Einträge $a_{k,j}$ sowie $a_{k,j+1}$ ersetzt durch $a_{k,j}c + a_{k,j+1}s$ sowie $-a_{k,j}\overline{s} + a_{k,j+1}\overline{c}$. So können wir mit nochmaliger Anwendung von $n-1$ Inversen der Givens-Rotationen schließlich das Produkt RQ berechnen, dabei wird die Matrix R mit der Matrix RQ überschrieben. Da, um die Matrix R zu erhalten, ja bereits unsere ursprüngliche Matrix A überschrieben wurde, muss nur für die $2\cdot(n-1)$ Werte von c und s neuer Speicher allokiert werden. Wir skizzieren die Vorgehensweise wiederum für den Fall $n = 4$, wobei wieder genau die Einträge in denen Änderungen geschehen mit $*$ gekennzeichnet sind.

$$R = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & + & + \\ & & & + \end{pmatrix} \xrightarrow{j=1} \begin{pmatrix} * & * & + & + \\ * & * & + & + \\ & & + & + \\ & & & + \end{pmatrix} \xrightarrow{j=2} \begin{pmatrix} + & * & * & + \\ + & * & * & + \\ & * & * & + \\ & & & + \end{pmatrix} \xrightarrow{j=3} \begin{pmatrix} + & + & * & * \\ + & + & * & * \\ & + & * & * \\ & & * & * \end{pmatrix}$$

Auch hier können wir die Struktur der Matrix verwenden. Wir müssen die Anwendung der Inversen der Givens-Rotationen nur bis zum $j+1$-ten Zeileneintrag realisieren. Wie man schnell einsieht, ist nun auch RQ wieder von oberer Hessenbergform. Im Folgenden ist der Algorithmus in Pseudocode formuliert.

Algorithmus 5 Berechnung von RQ für Hessenberg-Matrizen

Input: $A \in \mathbb{C}^{n\times n}$

for $i = 1,\ldots,n-1$ **do**
  $\begin{pmatrix} c_i \\ s_i \end{pmatrix} = \frac{1}{\sqrt{|a_{i,i}|^2+|a_{i+1,i}|^2}} \begin{pmatrix} a_{i,i} \\ a_{i+1,i} \end{pmatrix}$
  for $j = i,\ldots,n$ **do**
    $\begin{pmatrix} a_{i,j} \\ a_{i+1,j} \end{pmatrix} = \begin{pmatrix} \overline{c_i} & \overline{s_i} \\ -s_i & c_i \end{pmatrix} \begin{pmatrix} a_{i,j} \\ a_{i+1,j} \end{pmatrix}$
  end for
end for
for $j = 1,\ldots,n-1$ **do**
  for $i = 1,\ldots,j+1$ **do**
    $(a_{i,j}, a_{i,j+1}) = (a_{i,j}, a_{i,j+1}) \begin{pmatrix} c_j & -\overline{s_j} \\ s_j & \overline{c_j} \end{pmatrix}$
  end for
end for

Output: A wird mit dem Produkt RQ überschrieben, wobei Q und R die Matrizen aus einer QR-Zerlegung von A sind

2.4 Implementierung und Vergleich der verschiedenen Methoden

Die Implementierung der Verfahren wurde in Python durchgeführt. Wir werden bei den Codeausschnitten nur auf etwaige Unterschiede oder Ergänzungen zum Pseudocode eingehen, da die Algoritmen grundsätzlich schon dort erklärt wurden.

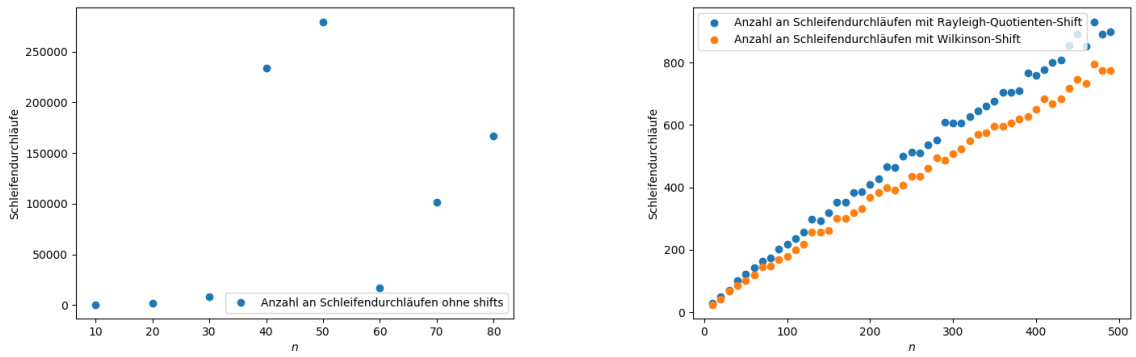
```
1  def QR_simple(A, tol = 1e-7):
2    n = A.shape[1]
3    count = 0
4    for i in range(n-1,0,-1):
5      while abs(A[i,i-1]) > tol:
6          Q,R = np.linalg.qr(A)
7          A = R@Q
8          count +=1
9      A[i,:i-n] = 0
10 return A, sorted(np.diag(A)), count
```

Listing 1: Implementation des QR-Verfahrens ohne shifts

Hier werden in Codezeile 5 die Zeilen der Matrix von unten nach oben durchgegangen. Sobald das (modulo Konstanten) größte Element, das laut Theorie in der unteren Nebendiagonale zu finden ist, nahe bei 0 ist, wird (in Codezeile 9) die gesamte Matrixzeile bis zum Diagonalelement auf 0 gesetzt. Dies ist nötig, um sich aufschaukelnde Fehler zu vermeiden. Es wird zudem für den späteren Vergleich eine Programmvariable eingeführt, die die Anzahl an Schleifendurchläufen im Verfahren zählt. Die QR-Zerlegung wird zunächst mit dem numpy-linalg Paket durchgeführt.

Die Implementation der anderen Verfahren unterscheidet sich vom Pseudocode nur in den schon dargelegten Punkten. Die anderen Codeausschnitte sind daher im Anhang zu finden.

Kommen wir nun zum Vergleich der QR-Verfahren. Zur Testung werden zufällige Matrizen, deren Eigenwerte bekannt sind, erstellt. Dazu wird zuerst ein zufälliges Array mit n Elementen generiert (unsere Eigenwerte), sowie eine Diagonalmatrix mit ebendiesen Werten auf der Diagonale. Schließlich führen wir eine Ähnlichkeitstransformation mit einer Zufallsmatrix durch.



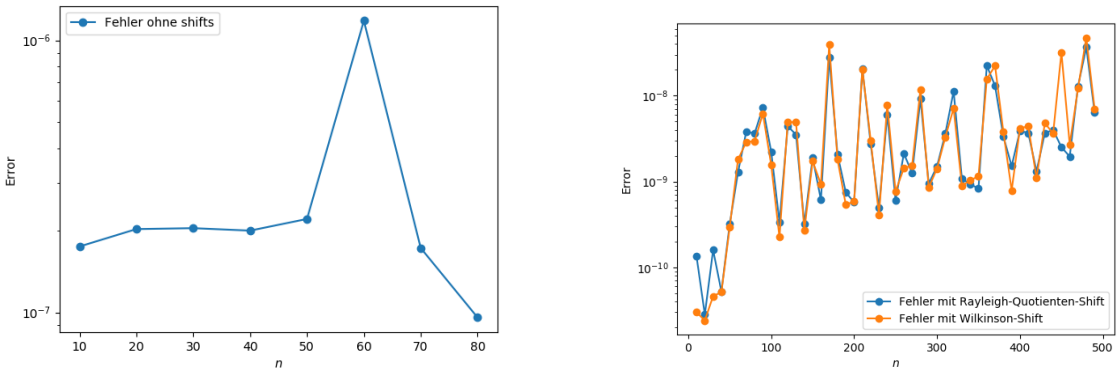
(a) Ohne shifts (b) Mit shifts

Abbildung 1: Anzahl der Schleifendurchläufe bei Matrixdimension n

Wir sehen in Abbildung 1 die Anzahl an Schritten bei den Verfahren für verschiedene Matrixdimensionen n. Hier kann man beobachten, dass das Verfahren ohne shifts selbst für kleine Matrizen schon sehr viele Schritte benötigt. Man sieht, dass die Anzahl an Schleifendurchläufen mit dem Wilkinson-Shift doch erkennbar niedriger ist als die Anzahl beim Verfahren mit Rayleigh-Quotienten-Shift.

In Abbildung 2 wird der Fehler in der Approximation semilogarithmisch dargestellt. Dabei wird das Eigenwertarray sowie das Array mit den Approximationen an die Eigenwerte sortiert, die Differenz der Arrays

gebildet und davon die Euklidnorm gemessen. Im Hinblick auf die Laufzeit wird bei dem Verfahren ohne shifts die Toleranz etwas höher gewählt, was sich auch im Fehler widerspiegelt. Die Fehler in den beiden Verfahren mit den shifts unterscheiden sich nur minimal. Daher ist wohl das QR-Verfahren mit Wilkinson-Shift vorzuziehen.



(a) Ohne shifts

(b) Mit shifts

Abbildung 2: Fehler in den Eigenwerten, gemessen in $||\cdot||_2$ bei Matrixdimension n

Um noch die Laufzeit zwischen dem QR-Verfahren mit Wilkinson-Shift, welches die vorimplementierte QR-Zerlegung aus dem *numpy.linalg* Paket verwendet, und dem selbstimplementierten Verfahren speziell für Hessenberg-Matrizen zu vergleichen, verwenden wir das *time* Paket und messen so die Zeit, die das Verfahren bei zufällig generierten Hessenberg-Matrizen mit immer höher werdender Dimension benötigt.

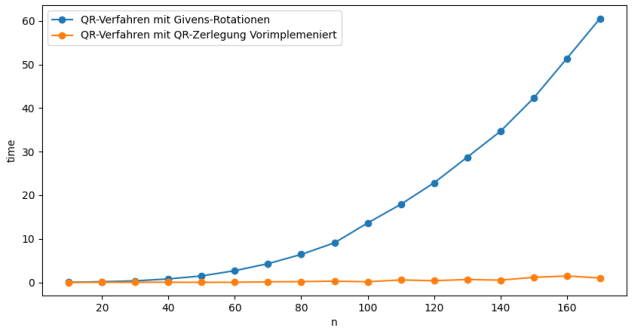


Abbildung 3: Vergleich der Laufzeit des vorimplementierten QR-Verfahrens mit QR-Verfahren speziell für Hessenberg-Matrizen

In Abbildung 3 werden dabei beide Verfahren auf zufällige Matrizen der Dimension n angewendet und die Laufzeit gemessen. Dem QR-Verfahren mit der vorimplementierten QR-Zerlegung werden dabei volle Matrizen übergeben, während das QR-Verfahren für Hessenberg-Matrizen natürlich auf eine Hessenberg-Matrix angewendet wird. Dabei zeigt sich, dass das Verfahren mit der vorimplementierten QR-Zerlegung selbst bei vollbesetzten Matrizen noch um ein Vielfaches schneller ist als das selbstimplementierte Verfahren. In der *numpy* Dokumentation liest man, dass die verwendete vorimplementierte QR-Zerlegung Interface für eine `LAPACK`-Routine ist. Das `LAPACK` (Linear Algebra Package) ist Fortran basiert und Fortran ist für Berechnungen aus der numerischen Linearen Algebra um ein Vielfaches schneller als Python-Code. Das erklärt wohl auch, warum das Verfahren mit der vorimplementierten QR-Zerlegung so viel besser abschneidet. Eine Dokumentation von `LAPACK` ist in [1] zu finden.

3 Das Lanczos-Verfahren

Wie in der Einleitung erwähnt, liefert uns das dort beschriebene Eigenwertproblem Matrizen mit sehr großer Dimension. Außerdem ist man in der Praxis häufig nur an wenigen extremalen Eigenwerten interessiert. Das QR-Verfahren ist für solche Probleme also nicht geeignet. Im Folgenden werden wir deshalb das Lanczos-Verfahren kennenlernen.

3.1 Idee

Wir zeigen zunächst ein theoretisches Resultat, welches eine Darstellung der Eigenwerte für hermitesche Matrizen liefert. Man sieht leicht, dass es sich hierbei um eine Verallgemeinerung des Satzes von Rayleigh handelt.

Lemma 3.1. *Sei $A \in \mathbb{C}^{n\times n}$ hermitesch bezüglich des Skalarproduktes $(\cdot,\cdot)$, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ die Eigenwerte von A (gemäß Vielfachheit gezählt) und $u_1, \ldots, u_n$ die zugehörigen normierten Eigenvektoren. Dann gilt*

$$\lambda_1 = \max_{v\in\mathbb{C}^n\setminus\{0\}} \frac{(Av,v)}{(v,v)}, \quad \lambda_k = \max_{\substack{v\in\mathbb{C}^n\setminus\{0\}\\(u_j,v)=0, j=1,\ldots,k-1}} \frac{(Av,v)}{(v,v)}, \quad k = 2,\ldots,n \tag{6}$$

und

$$\lambda_k = \max_{\substack{S\subseteq\mathbb{C}^n\\ \dim S = k}} \min_{v\in S\setminus\{0\}} \frac{(Av,v)}{(v,v)}, \quad k = 1,\ldots n. \tag{7}$$

Für $v \neq 0$ nennt man $\frac{(Av,v)}{(v,v)}$ den Rayleigh-Quotienten von v.

Beweis. Sei $v \in \mathbb{C}^n \setminus \{0\}$ beliebig. Da $u_1, \ldots, u_n$ eine Orthonormalbasis des $\mathbb{C}^n$ bilden, gilt $v = \sum_{j=1}^n (v,u_j)u_j$ und $\|v\|^2 = \sum_{j=1}^n |(u_j,v)|^2$. Damit folgt

$$(Av,v) = \sum_{j=1}^n (v,u_j)(\lambda_j u_j, v) = \sum_{j=1}^n \lambda_j |(u_j,v)|^2 \leq \lambda_1 \sum_{j=1}^n |(u_j,v)|^2 = \lambda_1 (v,v).$$

Dividieren durch (v,v) liefert $\lambda_1 \geq \frac{(Av,v)}{(v,v)}$ für alle $v \in \mathbb{C}^n \setminus \{0\}$ und für $v = u_1$ gilt sogar Gleichheit.

Für ein anderes $k = 2,\ldots,n$ können wir analog ein beliebiges $v \in \mathbb{C}^n \setminus \{0\}$ wählen, nun aber mit der Bedingung $(u_j,v) = 0, j = 1,\ldots,k$ betrachten. In der Summe $\sum_{j=1}^n \lambda_j |(u_j,v)|^2$ fallen somit die ersten $k-1$ Summanden weg und wir können die verbleibenden Eigenwerte mit λ_k nach oben abschätzen. Gleichheit gilt für den jeweiligen Eigenvektor u_k.

Für die Gleichheit in (7) gilt ähnlich zu oben für $v \in \tilde{S} := \mathbf{span}\{u_1,\ldots,u_k\}$, dass

$$(Av,v) = \sum_{j=1}^k \lambda_j |(v,u_j)|^2 \geq \lambda_k (v,v)$$

und daraus

$$\lambda_k \leq \min_{v\in\tilde{S}\setminus\{0\}} \frac{(Av,v)}{(v,v)}.$$

Für $v = u_k$ gilt wiederum Gleichheit.

Betrachten wir nun einen anderen Unterraum $S \neq \tilde{S}$ mit $\dim S = k$. Dann gibt es ein $v_0 \in S \setminus \{0\}$ mit $(v_0,u_j) = 0$ für $j = 1,\ldots,k$. Für dieses v_0 gilt $(Av_0,v_0) \leq \lambda_k (v_0,v_0)$. Das heißt, für beliebige solche S ist

$$\min_{v\in S\setminus\{0\}} \frac{(Av,v)}{(v,v)} \leq \lambda_k,$$

womit die Gleichheit gezeigt ist. □

Wir wollen das Verfahren der Vektoriteration zur Bestimmung des größten Eigenwertes einer hermiteschen Matrix wiederholen. Diese erzeugt ausgehend von einem Startvektor $x^{(0)} \in \mathbb{C}^n$ eine Folge von Iterierten $x^{(m)} := \frac{Ax^{(m-1)}}{\|Ax^{(m-1)}\|}$ für $m \in \mathbb{N}$. Definiert man die Approximationen $\lambda^{(m)}$ an den größten Eigenwert λ_1 durch den Rayleigh-Quotienten von $x^{(m-1)}$, so konvergiert $\lambda^{(m)}$ quadratisch gegen λ_1 (vgl. [3], Satz 9.9).

Definition 3.2 (Krylov-Raum)**.** *Sei $v_0 \in \mathbb{C}^n$ und $A \in \mathbb{C}^{n\times n}$. Dann bezeichnet*

$$\mathcal{K}_m(A, v_0) := \textbf{\textit{span}}\{v_0, Av_0, \ldots, A^{m-1}v_0\}, \quad m \in \mathbb{N}$$

den Krylov-Raum von A und v_0. Es bezeichne $\mathcal{P}_m \in \mathbb{C}^{n\times n}$ die orthogonale Projektion auf $\mathcal{K}_m$, also die eindeutige hermitesche Matrix mit $\mathcal{P}_m^2 = \mathcal{P}_m$ und $\mathcal{R}(\mathcal{P}_m) = \mathcal{K}_m$.

Beim Lanczos-Verfahren wird nun als Näherung an den größten Eigenwert das Maximum des Rayleigh-Quotienten über dem Krylov-Raum

$$\mathcal{K}_m(A, x^{(0)}) = \textbf{span}\{x^{(0)}, x^{(1)}, \ldots, x^{(m-1)}\}$$

verwendet, also

$$\lambda_1^{(m)} := \max_{x\in\mathcal{K}_m\setminus\{0\}} \frac{(Ax, x)}{(x, x)}.$$

Da $x^{(m-1)} \in \mathcal{K}_m$, gilt

$$\lambda^{(m)} \leq \lambda_1^{(m)} \leq \lambda_1$$

und $\lambda_1^{(m)}$ ist somit eine mindestens so gute Approximation von λ_1 wie $\lambda^{(m)}$.

Betrachten wir die Abbildung

$$\begin{aligned} \mathcal{A}_m : \mathcal{K}_m &\to \mathcal{K}_m \\ x &\mapsto \mathcal{P}_m A x, \end{aligned}$$

so sehen wir, dass diese selbstadjungiert ist. Es gilt nämlich für $x, y \in \mathcal{K}_m$

$$(\mathcal{A}_m x, y) = (\mathcal{P}_m A \mathcal{P}_m x, y) = (x, \mathcal{P}_m A \mathcal{P}_m y) = (x, \mathcal{A}_m y),$$

da $\mathcal{P}_m$ und A hermitesch sind. Also besitzt $\mathcal{A}_m$ nur reelle Eigenwerte $\lambda_1(\mathcal{A}_m) \geq \lambda_2(\mathcal{A}_m) \geq \cdots \geq \lambda_m(\mathcal{A}_m)$. Für den größten Eigenwert gilt

$$\lambda_1(\mathcal{A}_m) = \max_{x\in\mathcal{K}_m\setminus\{0\}} \frac{(\mathcal{A}_m x, x)}{(x, x)} = \max_{x\in\mathcal{K}_m\setminus\{0\}} \frac{(Ax, \mathcal{P}_m x)}{(x, x)} = \max_{x\in\mathcal{K}_m\setminus\{0\}} \frac{(Ax, x)}{(x, x)} = \lambda_1^{(m)}.$$

Die erste Gleichheit kann man sich mit gleichen Rechnungen wie im Beweis von Lemma 3.1 überlegen. Der größte Eigenwert $\lambda_1(\mathcal{A}_m)$ von $\mathcal{A}_m$ ist also nach unseren bisherigen Überlegungen eine gute Approximation an den größten Eigenwert λ_1 von A. Es liegt nahe, für alle $j \leq m$ auch $\lambda_j(\mathcal{A}_m)$ als Approximation von λ_j zu betrachten. Dass dies eine gute Wahl ist, werden wir im Abschnitt 3.3 zeigen.

3.2 Herleitung des Verfahrens

Es ist also nun Ziel, die Eigenwerte der Abbildung $\mathcal{A}_m$ zu berechnen. Hierfür benötigen wir eine Repräsentation von $\mathcal{A}_m$ durch eine Matrix B_m bezüglich einer Orthonormalbasis $\{v_0, \dots, v_{m-1}\}$ des Krylov-Raums $\mathcal{K}_m(A, v_0)$. Für die Einträge von B_m gilt

$$(B_m)_{ij} = (v_i, \mathcal{A}_m v_j) = v_i^* A v_j,$$

woraus wir die Matrixgleichung

$$B_m = V_m^* A V_m \in \mathbb{C}^{m \times m} \tag{8}$$

erhalten.

Der Vorteil dieser Darstellung ist, dass die Eigenwerte von B_m jenen von $\mathcal{A}_m$ entsprechen. Wenn wir m nicht zu groß gewählt haben, können wir diese mit dem aus Kapitel 2 bekannten QR-Verfahren mit wenig Aufwand berechnen.

Eine Möglichkeit zur Bestimmung einer Orthonormalbasis liefert das CG-Verfahren, auf welches an dieser Stelle nicht genauer eingegangen werden soll. Aus diesem ergibt sich der Ansatz, B_m als eine Tridiagonalmatrix (also auch Hessenberg-Matrix) der Form

$$\begin{pmatrix} \gamma_0 & \delta_0 & 0 & \dots & 0 \\ \delta_0 & \gamma_1 & \delta_1 & & \vdots \\ 0 & \delta_1 & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \delta_{m-2} \\ 0 & \dots & 0 & \delta_{m-2} & \gamma_{m-1} \end{pmatrix} \tag{9}$$

aufzustellen. Die Gleichung (8) schreiben wir um als $AV_m = V_m B_m$, was spaltenweise zu folgenden Gleichungen führt:

$$\begin{aligned} Av_0 &= \gamma_0 v_0 + \delta_0 v_1 \\ Av_j &= \delta_{j-1} v_{j-1} + \gamma_j v_j + \delta_j v_{j+1}, \quad j = 1, \dots, m-2. \end{aligned}$$

Da es sich bei $\{v_0, \dots, v_{m-1}\}$ um ein Orthogonalsystem handelt, liefert Multiplizieren der j-ten Gleichung von links mit v_j^* die Bedingung $\gamma_j = v_j^* A v_j$ für $j = 0, \dots, m-2$. Somit können wir die j-te Gleichung rekursiv nach v_{j+1} auflösen:

$$v_{j+1} = \begin{cases} \frac{\overbrace{(A - \gamma_0) v_0}^{=: w_0}}{\delta_0} & , j = 0 \\ \frac{\overbrace{(A - \gamma_j) v_j - \delta_{j-1} v_{j-1}}^{=: w_j}}{\delta_j} & , j \geq 1 \end{cases}$$

Hierbei setzen wir $\delta_j := \|w_j\|$ falls $\|w_j\| \neq 0$. Andernfalls bricht die Iteration ab. Daraus ergibt sich nun der Algorithmus für das Lanczos-Verfahren.

3.3 Konvergenz der Eigenwerte von hermiteschen Matrizen

In diesem Abschnitt beweisen wir, dass die durch das Lanczos-Verfahren erhaltenen Eigenwerte $\lambda_1^{(m)}, \dots, \lambda_m^{(m)}$ gegen Eigenwerte $\lambda_1, \dots, \lambda_m$ der hermiteschen Matrix $A \in \mathbb{C}^{n \times n}$ konvergieren. Die Argumentation ist dabei in großen Teilen angelehnt an [2], Abschnitt 4.1.

Algorithmus 6 Lanczos-Verfahren

Input: $A \in \mathbb{C}^{n\times n}$ hermitesche Matrix, zufälliger normierter Startvektor $v_0 \in \mathbb{C}^n$, maximale Krylov-Raumdimension m

1: $\gamma_0 = v_0^* A v_0$
2: $w_0 = (A - \gamma_0 I)v_0$
3: $\delta_0 = \|w_0\|_2$
4: **while** $\delta_j \neq 0$ und $j < m$ **do**
5: $\quad v_{j+1} = w_j/\delta_j$
6: $\quad j = j + 1$
7: $\quad \gamma_j = v_j^* A v_j$
8: $\quad w_j = (A - \gamma_j I)v_j - \delta_{j-1} v_{j-1}$
9: $\quad \delta_j = \|w_j\|_2$
10: **end while**
11: Berechne mittels QR-Verfahren Eigenwerte von $B_j \in \mathbb{C}^{j\times j}$ aus (9) (Hessenberg-Matrix)

Output: Approximation an die Eigenwerte von $\mathcal{A}_j$

Wir werden ein Lemma benötigen, welches Elemente des Krylov-Raums in Zusammenhang mit Polynomen charakterisiert.

Lemma 3.3. *Sei Π_m der Raum der Polynome in einer Veränderlichen mit maximalem Grad m. Dann ist $v \in \mathcal{K}_m(A, v_0) \subseteq \mathbb{C}^n$ genau dann, wenn ein Polynom $p \in \Pi_{m-1}$ existiert mit $v = p(A)v_0$.*
Ist A diagonalisierbar mit Eigenwerten $\lambda_1, \ldots, \lambda_n$ und zugehörigen Eigenvektoren $u_1, \ldots, u_n$, dann existiert eine eindeutige Darstellung $v_0 = \sum_{j=1}^n \alpha_j u_j$ und es gilt

$$v \in \mathcal{K}_m \Leftrightarrow \exists\, p \in \Pi_{m-1} : v = \sum_{j=1}^{n} p(\lambda_j)\alpha_j u_j.$$

Beweis. Wir zeigen zunächst die erste Aussage. Nach Definition des Krylov-Raums gilt

$$v \in \mathcal{K}_m \Leftrightarrow \exists\, a_0, \ldots a_{m-1} \in \mathbb{C} : v = \sum_{i=0}^{m-1} a_i A^i v_0 = (\sum_{i=0}^{m-1} a_i A^i)v_0 \tag{10}$$

$$\Leftrightarrow \exists\, p : x \mapsto \sum_{i=0}^{m-1} a_i x^i \in \Pi_{m-1} : v = p(A)v_0. \tag{11}$$

Da die Eigenvektoren eine Basis des $\mathbb{C}^n$ bilden, folgt die eindeutige Darstellung von $v_0 = \sum_{j=1}^n \alpha_j u_j$. Setzen wir diese in (10) ein und verwenden die Tatsache, dass $A^i u_j = \lambda_j^i u_j$ für $j = 1, \ldots, n$ und $i = 0, \ldots, m-1$, so erhalten wir auch die zweite Aussage. □

Definition 3.4. *Für $m \in \mathbb{N}$ sind die Chebyshev-Polynome $T_m \in \Pi_m$ definiert durch*

$$T_m(x) := \frac{1}{2}\Big((x + \sqrt{x^2-1})^m + (x - \sqrt{x^2-1})^m\Big), \quad x \in \mathbb{R}. \tag{12}$$

Bemerkung 3.5. *Es gibt weitere Definitionen der Chebyshev-Polynome. Die bekannteste ist*

$$T_m(x) := \cos(m \arccos x), \quad x \in [-1, 1], \tag{13}$$

welche aus der Identität $T_m(\cos\varphi) = \cos(m\varphi)$ für $\varphi \in \mathbb{R}$ folgt.

Aus dieser Darstellung folgt auch mittels Additionstheorem die 2-Term-Rekursion

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_m(x) = 2xT_{m-1}(x) - T_{m-2}(x), \quad m \geq 2,$$

welche zeigt, dass es sich bei T_m tatsächlich um ein Polynom vom Grad m handelt.

Lemma 3.6. *Sei $[a,b]$ ein nicht-leeres Intervall in $\mathbb{R}$ und sei $c \geq b$. Dann gilt mit $\gamma := 1 + 2\frac{c-b}{b-a} \geq 1$*

$$\min_{\substack{p\in\Pi_m \\ p(c)=1}} \max_{x\in[a,b]} |p(x)| \leq \frac{1}{|T_m(\gamma)|} \leq 2(\gamma + \sqrt{\gamma^2-1})^{-m}. \tag{14}$$

Beweis. Wir verwenden die affin lineare Abbildung $\Phi : [a,b] \to [-1,1]$ mit

$$\Phi(x) = 1 + 2\frac{x-b}{b-a}, \quad x \in [a,b].$$

Wir definieren

$$\hat{p} := \frac{T_m \circ \Phi}{|T_m(\Phi(c))|}$$

und können die Darstellung von T_m aus (13) verwenden (da Φ nach $[-1,1]$ abbildet), aus der

$$\max_{x\in[a,b]} |\hat{p}(x)| \leq \frac{1}{|T_m(\gamma)|}$$

folgt und daraus auch die erste Ungleichung.

Die zweite Ungleichung folgt aus der Definition der Chebyshev-Polynome (12) und der Tatsache, dass $\gamma \geq 1$ und deshalb $|T_m(\gamma)| = T_m(\gamma) \geq \frac{1}{2}(\gamma + \sqrt{\gamma^2-1})^m$ gilt. □

Mit den bis hier getroffenen Vorbereitungen können wir nun die Konvergenz des Projektionsverfahrens nachweisen.

Satz 3.7 (Konvergenz der Eigenwerte hermitescher Matrizen)**.** *Sei $A \in \mathbb{C}^{n\times n}$ eine hermitesche Matrix mit paarweise verschiedenen Eigenwerten $\lambda_1 > \lambda_2 > \cdots > \lambda_n$. Bezeichne mit $u_1, \ldots, u_n$ eine Orthonormalbasis aus Eigenvektoren zu den jeweiligen Eigenwerten. Für $1 \leq m < n$ werden die Eigenwerte der linearen Abbildung $\mathcal{A}_m : \mathcal{K}_m(A, v_0) \to \mathcal{K}_m(A, v_0)$, die durch $v \mapsto \mathcal{P}_m A v$ gegeben ist, mit $\lambda_1^{(m)} \geq \lambda_2^{(m)} \geq \cdots \geq \lambda_m^{(m)}$ bezeichnet. Dabei ist v_0 ein beliebiger Startvektor, der nicht orthogonal zu den ersten $m-1$ Eigenvektoren von A ist. Dann gilt*

$$0 \leq \lambda_i - \lambda_i^{(m)} \leq (\lambda_i - \lambda_n)(\tan\theta_i)^2 \kappa_i^{(m)} \left(\frac{1}{T_{m-i}(\gamma_i)}\right)^2, \quad i = 1, \ldots, m-1, \tag{15}$$

wobei

$$\tan\theta_i := \frac{\|(\mathrm{id} - \mathcal{P}_{u_i})v_0\|}{\|\mathcal{P}_{u_i} v_0\|}, \quad \gamma_i := 1 + 2\frac{\lambda_i - \lambda_{i+1}}{\lambda_{i+1} - \lambda_n}$$

und

$$\kappa_1^{(m)} := 1, \quad \kappa_i^{(m)} := \left(\prod_{k=1}^{i-1} \frac{\max_{j=i+1,\ldots,n}(\lambda_k^{(m)} - \lambda_j)}{\lambda_k^{(m)} - \lambda_i}\right)^2, \quad i = 2, \ldots, m.$$

Dabei wird mit $\mathcal{P}_{u_i}$ die Projektion auf den eindimensionalen Unterraum, der von u_i aufgespannt wird, bezeichnet. Das θ_i kann somit als Winkel zwischen v_0 und u_i verstanden werden.

Beweis. Aufgrund der Übersichtlichkeit bezeichnen wir im folgenden Beweis mit $\mathcal{K}_m$ den Krylov-Raum $\mathcal{K}_m(A, v_0)$.
Seien $v_1, \ldots, v_m$ Orthonormalbasisvektoren von $\mathcal{K}_m$ und $V \in \mathbb{C}^{n\times m}$ die Matrix, die diese Vektoren als Spalten besitzt. Dann ist

$$B_m := V^* A V \in \mathbb{C}^{m\times m}$$

die Matrixdarstellung der Abbildung $\mathcal{A}_m$ bezüglich der Orthonormalbasis $\{v_1, \dots, v_m\}$.

Diese Matrix ist offensichtlich hermitesch. Somit können die Eigenwerte $\lambda_1^{(m)}, \dots, \lambda_m^{(m)}$ mithilfe von (7) berechnet werden.

Es gilt also

$$\begin{aligned}
\lambda_k^{(m)} &= \max_{\substack{S \subseteq \mathbb{C}^m \\ \dim S = k}} \min_{v \in S \setminus \{0\}} \frac{(B_m v, v)}{(v, v)} \\
&= \max_{\substack{S \subseteq \mathbb{C}^m \\ \dim S = k}} \min_{v \in S \setminus \{0\}} \frac{(AVv, Vv)}{(Vv, Vv)} \\
&= \max_{\substack{T \subseteq \mathcal{K}_m \\ \dim T = k}} \min_{w \in T \setminus \{0\}} \frac{(Aw, w)}{(w, w)} \\
&\leq \max_{\substack{T \subseteq \mathbb{C}^n \\ \dim T = k}} \min_{w \in T \setminus \{0\}} \frac{(Aw, w)}{(w, w)} = \lambda_k.
\end{aligned} \tag{16}$$

Dabei gilt die dritte Gleichheit, da V als Abbildung von $\mathbb{C}^m$ nach $\mathcal{K}_m$ bijektiv ist. Also kann jeder k-dimensionale Unterraum $T \subseteq \mathcal{K}_m$ als Bild von einem k-dimensionalen Unterraum $S \subseteq \mathbb{C}^m$ unter der Abbildung V gesehen werden. Somit ist die erste Ungleichung in (15) erfüllt.

Wir beweisen nun die zweite Ungleichung zuerst für den Fall $i = 1$. Sei $v_0 = \sum_{j=1}^n \alpha_j u_j$. Wir bemerken, dass $\alpha_1, \dots, \alpha_{m-1}$ nicht verschwinden, da v_0 nicht orthogonal zu diesen Eigenvektoren gewählt wurde. Dies rechtfertigt im Folgenden Abschätzungen, bei denen ohne diese Bedingung durch 0 geteilt werden würde.

Mit analogen Umformungen wie in (16), dem Satz von Pythagoras und Lemma 3.3 gilt somit

$$\begin{aligned}
\lambda_1 - \lambda_1^{(m)} &= \lambda_1 - \max_{v \in \mathcal{K}_m \setminus \{0\}} \frac{(Av, v)}{(v, v)} = \min_{v \in \mathcal{K}_m \setminus \{0\}} \lambda_1 - \frac{(Av, v)}{(v, v)} = \min_{v \in \mathcal{K}_m \setminus \{0\}} \frac{\lambda_1 (v, v) - (Av, v)}{(v, v)} \\
&= \min_{p \in \Pi_{m-1} \setminus \{0\}} \frac{\lambda_1 \left\| \sum_{j=1}^n p(\lambda_j) \alpha_j u_j \right\|^2 - (\sum_{j=1}^n p(\lambda_j) \alpha_j A u_j, \sum_{j=1}^n p(\lambda_j) \alpha_j u_j)}{\left\| \sum_{j=1}^n p(\lambda_j) \alpha_j u_j \right\|^2} \\
&= \min_{p \in \Pi_{m-1} \setminus \{0\}} \frac{\lambda_1 \sum_{j=1}^n |p(\lambda_j) \alpha_j|^2 - \sum_{j=1}^n \lambda_j |p(\lambda_j) \alpha_j|^2}{\sum_{j=1}^n |p(\lambda_j) \alpha_j|^2} \\
&\leq \min_{\substack{p \in \Pi_{m-1} \\ p(\lambda_1) = 1}} \frac{\sum_{j=2}^n (\lambda_1 - \lambda_j) |p(\lambda_j) \alpha_j|^2}{\sum_{j=1}^n |p(\lambda_j) \alpha_j|^2} \\
&\leq (\lambda_1 - \lambda_n) \frac{\sum_{j=2}^n |\alpha_j|^2}{|\alpha_1|^2} \min_{\substack{p \in \Pi_{m-1} \\ p(\lambda_1) = 1}} \max_{j=2,\dots,n} \left(\frac{|p(\lambda_j)|}{|p(\lambda_1)|} \right)^2.
\end{aligned} \tag{17}$$

Wir betrachten nun den Ausdruck $\tan \theta_i$. Für diesen gilt

$$(\tan \theta_1)^2 = \frac{\|(\mathrm{id} - \mathcal{P}_{u_1}) v_0\|^2}{\|\mathcal{P}_{u_1} v_0\|^2} = \frac{\|v_0 - \alpha_1 u_1\|^2}{\|\alpha_1 u_1\|^2} = \frac{\sum_{j=2}^n |\alpha_j|^2}{|\alpha_1|^2}. \tag{18}$$

Die rechte Seite von (18) finden wir auch in obiger Abschätzung und können sie also durch $(\tan \theta_1)^2$ ersetzen. Mithilfe der Ungleichung (14) mit $a = \lambda_n, b = \lambda_2$ und $c = \lambda_1$ erhalten wir außerdem

$$\min_{\substack{p \in \Pi_{m-1} \\ p(\lambda_1) = 1}} \max_{j=2,\dots,n} \left(\frac{|p(\lambda_j)|}{|p(\lambda_1)|} \right)^2 \leq \min_{\substack{p \in \Pi_{m-1} \\ p(\lambda_1) = 1}} \max_{\lambda \in [\lambda_n, \lambda_2]} |p(\lambda)|^2 \leq \left(\frac{1}{T_{m-1}(\gamma_1)} \right)^2.$$

Daraus folgt die gewünschte Ungleichung und (15) ist für $i = 1$ erfüllt.

Betrachten wir nun den Fall $i > 1$. Bezeichne mit $\tilde{u}_j^{(m)} \in \mathbb{C}^m$, $j = 1, \dots, m$ eine Orthonormalbasis aus Eigenvektoren zu den Eigenwerten $\lambda_j^{(m)}$, $j = 1, \dots, m$ der Abbildung B_m. Mit (6) erhalten wir dann

$$\begin{aligned} \lambda_i^{(m)} &= \max_{\substack{v \in \mathbb{C}^m \setminus \{0\} \\ (\tilde{u}_j^{(m)}, v) = 0, j = 1, \dots, i-1}} \frac{(B_m v, v)}{(v, v)} \\ &= \max_{\substack{v \in \mathbb{C}^m \setminus \{0\} \\ (\tilde{u}_j^{(m)}, v) = 0, j = 1, \dots, i-1}} \frac{(AVv, Vv)}{(Vv, Vv)} \\ &= \max_{\substack{w \in \mathcal{K}_m \setminus \{0\} \\ (V\tilde{u}_j^{(m)}, w) = 0 j = 1, \dots, i-1}} \frac{(Aw, w)}{(w, w)} \end{aligned} \tag{19}$$

Da die $\tilde{u}_j^{(m)}, j = 1, \dots, m$ eine Orthonormalbasis von $\mathbb{C}^m$ sind, sind somit die $u_j^{(m)} := {}^{V\tilde{u}_j^{(m)}} / \left\| V\tilde{u}_j^{(m)} \right\|$ eine Orthonormalbasis von $\mathcal{K}_m$. Weil es für jedes $w \in \mathcal{K}_m$ ein $\tilde{w} \in \mathbb{C}^m$ gibt, sodass $w = V\tilde{w}$ ist, folgt mit Lemma 3.3 die Darstellung

$$w = V \left(\sum_{k=1}^{m} p(\lambda_k^{(m)}) \tilde{\beta}_k \tilde{u}_j^{(m)} \right) = \sum_{k=1}^{m} p(\lambda_k^{(m)}) \beta_k u_k^{(m)},$$

wobei die $b_k \in \mathbb{C}$ sind und $p \in \Pi_{m-1}$ ist.

Sei nun also $w \in \mathcal{K}_m$ gegeben durch $w = \sum_{k=1}^{m} p(\lambda_k^{(m)}) \beta_k u_k^{(m)}$. Dann gilt für alle $j < i$

$$0 \overset{!}{=} (w, u_j^{(m)}) = (\sum_{k=1}^{m} p(\lambda_k^{(m)}) \beta_k u_k^{(m)}, u_j^{(m)}) = \sum_{k=1}^{m} p(\lambda_k^{(m)}) \beta_k (u_k^{(m)}, u_j^{(m)}) = p(\lambda_j^{(m)}) \beta_j.$$

Es gibt also zwei Möglichkeiten, diese Gleichheit zu erfüllen. Entweder ist $p(\lambda_j^{(m)}) = 0$ oder $\beta_j = 0$. Wir betrachten zuerst den Fall, dass $\beta_j \neq 0$ für alle $j = 1, \dots, i-1$ gilt.

Da dann die $\lambda_j^{(m)}$, $j = 1, \dots, i-1$ Nullstellen von p sind, können wir

$$p(\lambda) = \prod_{j=1}^{i-1} \frac{\lambda_j^{(m)} - \lambda}{\lambda_j^{(m)} - \lambda_i} q(x)$$

schreiben, wobei $q \in \Pi_{m-i}$.

Mit der Darstellung (19) von $\lambda_i^{(m)}$ gilt nach analogen Umformungen wie in (17)

$$\begin{aligned}
\lambda_i - \lambda_i^{(m)} &= \min_{\substack{p\in\Pi_{m-1}\setminus\{0\}\\ p(\lambda_j^{(m)})=0, j=1,\dots,i-1}} \frac{\sum_{j=1}^n (\lambda_i-\lambda_j)|p(\lambda_j)\alpha_j|^2}{\sum_{j=1}^n |p(\lambda_j)\alpha_j|^2} \\
&= \min_{\substack{p\in\Pi_{m-1}\setminus\{0\}\\ p(\lambda_j^{(m)})=0, j=1,\dots,i-1}} \frac{\sum_{j=1,j\neq i}^n (\lambda_i-\lambda_j)|p(\lambda_j)\alpha_j|^2}{\sum_{j=1}^n |p(\lambda_j)\alpha_j|^2} \\
&\le (\lambda_i-\lambda_n)\frac{\sum_{j=1,j\neq i}^n |\alpha_j|^2}{|\alpha_i|^2} \min_{\substack{p\in\Pi_{m-1}, p(\lambda_i)\neq 0\\ p(\lambda_j^{(m)})=0, j=1,\dots,i-1}} \max_{j=i+1,\dots,n} \left(\frac{|p(\lambda_j)|}{|p(\lambda_i)|}\right)^2 \\
&\le (\lambda_i-\lambda_n)\frac{\sum_{j=1,j\neq i}^n |\alpha_j|^2}{|\alpha_i|^2} \min_{\substack{q\in\Pi_{m-i}\\ q(\lambda_i)=1}} \max_{j=i+1,\dots,n} \left(\left|\prod_{k=1}^{i-1}\frac{\lambda_k^{(m)}-\lambda_j}{\lambda_k^{(m)}-\lambda_i}\right|\left|\frac{q(\lambda_j)}{q(\lambda_i)}\right|\right)^2 \\
&\le (\lambda_i-\lambda_n)\frac{\sum_{j=1,j\neq i}^n |\alpha_j|^2}{|\alpha_i|^2} \left|\prod_{k=1}^{i-1}\frac{\max_{j=i+1,\dots,n}(\lambda_k^{(m)}-\lambda_j)}{\lambda_k^{(m)}-\lambda_i}\right|^2 \min_{\substack{q\in\Pi_{m-i}\\ q(\lambda_i)=1}} \max_{j=i+1,\dots,n} \left(\left|\frac{q(\lambda_j)}{q(\lambda_i)}\right|\right)^2.
\end{aligned}$$

Wieder ist

$$(\tan\theta_i)^2 = \frac{\|(\mathrm{id}-\mathcal{P}_{u_i})v_0\|^2}{\|\mathcal{P}_{u_i}v_0\|^2} = \frac{\|v_0-\alpha_i u_i\|^2}{\|\alpha_i u_i\|^2} = \frac{\sum_{j=1,j\neq i}^n |\alpha_j|^2}{|\alpha_i|^2}$$

und mit $a=\lambda_n, b=\lambda_{i+1}$, $c=\lambda_i$ in Lemma 3.6, erhalten wir die Behauptung.

Wenn nun β_j existieren, die gleich 0 sind, kann man die Bedinungung $p(\lambda_j^{(m)})=0$ für die Polynome, über die minimiert wird, weglassen. Da somit mehr Polynome betrachtet werden, wird das Minimum sicherlich nicht größer. Das heißt, wir können für diesen Fall die gleiche Abschätzung verwenden. Also gilt (15) für alle $i=1,\dots,m-1$. □

Die mit m exponentielle Konvergenz der Eigenwerte ist nun ersichtlich, da mit (14)

$$\left(\frac{1}{T_{m-i}(\gamma_i)}\right)^2 \le 4\left(\gamma_i+\sqrt{\gamma_i^2-1}\right)^{-2(m-i)}$$

gilt. Die Konvergenzgeschwindigkeit nimmt in der Theorie in Richtung der kleineren Eigenwerte ab, da der Grad des Chebyshev-Polynoms mit wachsendem i kleiner wird. Wir bemerken noch, dass bei der Approximation von λ_i bei Eigenwerten $\lambda_i \approx \lambda_j$, $j<i$, das Produkt $\kappa_i^{(m)}$ groß werden kann, da der Nenner wegen $\lambda_j^{(m)}-\lambda_i$ klein wird. Dies hat eine schlechtere Approximation zur Folge. Die anderen vorkommenden Terme wie $\tan\theta_i$ oder auch der Zähler vom Produkt in $\kappa_i^{(m)}$ haben für die Größenordnung der Abschätzung eher wenig Bedeutung. Wie sich die Konvergenz in der Praxis verhält, wird im Abschnitt 3.5 beschrieben.

3.4 Implementierung und Arnoldi-Verfahren

Die Implementation des Lanczos-Verfahrens entspricht dem Pseudocode und ist im Anhang zu finden. Hier werden wir besprechen, wo das Lanczos-Verfahren an seine Grenzen stößt. Da wir das Lanczos-Verfahren nur auf einfache Eigenwertprobleme und nicht auf verallgemeinerte Eigenwertprobleme der Form (2) anwenden können, müssen wir so ein verallgemeinertes Eigenwertproblem zuerst nach Lemma 1.4 zu einem äquivalenten einfachen Eigenwertproblem umformulieren. Hier kommt Bemerkung 1.5 zu tragen, das Lanczos-Verfahren können wir nicht ohne weiteres anwenden.

Um diese Problematik zu umgehen, werden wir das sogenannte Arnoldi-Verfahren verwenden. Auch in diesem Verfahren bekommen wir eine Orthonormalbasis des Krylov-Raums und eine Matrix, welche die Abbildung $\mathcal{A}_m$ bezüglich dieser Basis beschreibt. Dabei ist $\mathcal{A}_m$ nicht mehr selbstadjungiert. In dem Arnoldi-Verfahren

wird nun jeder neue Basisvektor mit dem (modifizierten) Gram-Schmidtschen Orthogonalisierungsverfahren gewonnen. Dabei wird ein neuer Basisvektor zu allen bisherigen Basisvektoren orthogonalisiert. Dies ist ein wesentlicher Unterschied zum Lanczos-Verfahren, wo nur gegen den vorherigen Basisivektor explizit orthogonalisiert werden muss. Es macht Sinn, auch beim Lanczos-Verfahren eine volle Reorthogonalisierung durchzuführen. Diese ist zwar etwas aufwändiger, aber wenn nur gegen den vorherigen Basisvektor orthogonalisiert wird, kann es zu Auslöschung und somit zu Verlust der Orthogonalität kommen. Es gibt auch diverse Strategien, bei denen keine volle, sondern nur eine teilweise Reorthogonalisierung durchgeführt wird. Dazu gibt es eine Vielzahl an Literatur, wir verweisen hier auf [5].

Die Matrix, die man schließlich aus dem Arnoldi-Verfahren erhält, hat dann nicht Tridiagonalform, sondern ist eine obere Hessenberg-Matrix. Die Eigenwerte dieser können wir wieder mit dem QR-Verfahren für Hessenberg-Matrizen approximieren. Das Lanczos-Verfahren ist so gesehen eigenlich ein Spezialfall des Arnoldi-Verfahrens - es ist das Arnoldi-Verfahren angewandt auf hermitesche Matrizen.

Das Arnoldi-Verfahren funktioniert nun nicht nur für hermitesche Matrizen, sondern auch für allgemeine. Für einen Konvergenzbeweis des Arnoldi-Verfahrens verweisen wir auf [4]. Im Folgenden ist das Verfahren als Pseudocode formuliert.

Algorithmus 7 Arnoldi-Verfahren

Input: $A \in \mathbb{C}^{n\times n}$, zufälliger normierter Startvektor $v_0 \in \mathbb{C}^n$

for $j = 1, \dots, k$ **do**
  $w = Av_{j-1}$
  for $l = 1, \dots, j$ **do**
    $h_{l,j} = (w, v_{l-1})$
    $w = w - h_{l,j}v_{l-1}$
  end for
  $h_{j+1,j} = \|w\|_2$
  if $h_{j+1,j} = 0$ **then**
    Ein invarianter Unterraum wurde gefunden
  else
    $v_j = \frac{w}{h_{j+1,j}}$
  end if
end for

Output: Die Vektoren $v_0, \dots, v_{k-1}$ bilden eine Orthonormalbasis des Krylov-Raums, die Matrix H ist eine Hessenberg-Matrix, die die Projektion von A bezüglich dieser Basis darstellt

Eine weitere Möglichkeit um das Problem zu umgehen, wäre es anstatt des Euklidschen-Skalarproduktes das B-Skalarprodukt zu verwenden. Bezüglich diesem ist die verallgemeinerte Matirx aus (3) dann Symmetrisch:

$$((A-\rho_h B)^{-1}Bx, y)_B = ((A-\rho_h B)^{-1}Bx, By) = ((A-\rho_h B)^{-1}Bx)^\top By =$$
$$(Bx)^\top (A-\rho_h B)^{-1}By = x^\top B(A-\rho_h B)^{-1}By = (x, (A-\rho_h B)^{-1}By)_B$$

Damit kann das Lemma 3.1 angewendet werden, auf dem unsere Konvergenztheorie beruht. Im Lanczos-Verfahren müsste man dann die Skalarprodukte dementsprechend anpassen.

3.5 Ergebnisse

Zunächst wird das Lanczos-Verfahren mit zufällig generierten hermiteschen Matrizen mit bekannten Eigenwerten getestet. Dazu wird wieder ein zufälliges Array mit Eigenwerten generiert und auf die Diagonalmatrix mit diesen Eigenwerten eine Ähnlichkeitstransformation, hier mit zufälligen unitären Matrizen, angewendet. Dass dabei eine hermitesche Matrix entsteht, kann man schnell nachrechnen: sei dazu $A \in \mathbb{R}^{n\times n}$ eine Diagonalmatrix und $B \in \mathbb{C}^{n\times n}$ eine unitäre Matrix. Dann gilt also für die aus der Ähnlichkeitstransformation entstehende Matrix

$$(B^*AB)^* = B^*A^*B^{**} = B^*AB.$$

Erste Testungen mit zufällig generierten uniform verteilten Eigenwerten zeigen sehr mäßige Konvergenz. Lässt man dabei das gesamte Array mit den approximierten Eigenwerten ausgeben, zeigt sich eine gewisse Symmetrie, mit dem Fehler in den größten Eigenwerten sinkt auch der in den kleinsten Eigenwerten mit der selben Rate. In Abbildung 4 ist der absolute Fehler in den 2 größten Eigenwerten für steigende Krylov-Raumdimension dargestellt. Dabei wird das Lanczos-Verfahren auf eine Matrix der Dimension 1000 mit Eigenwerten uniform verteilt im Intervall $(-100, 400)$ angewendet. Selbst bei Krylov-Raumdimension 100 lässt sich noch keine besonders gute Approximation an nur zwei Eigenwerte erkennen. Für uniform verteilte Eigenwerte ist das Lanczos-Verfahren also nicht gut geeignet. Sobald jedoch Eigenwerte existieren, die sich weiter vom Spektrum entfernen, erhalten wir sehr schnelle Konvergenz gegen diese.

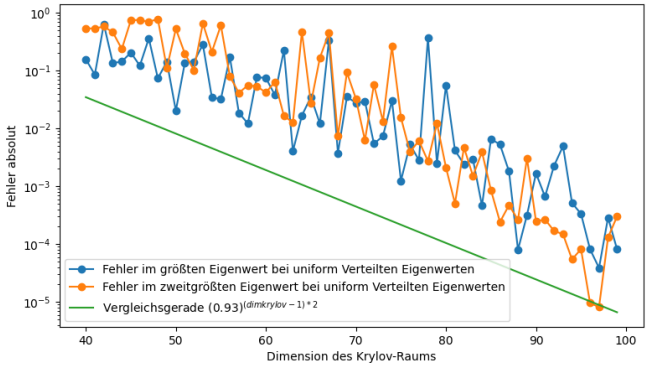


Abbildung 4: Fehler im größten sowie zweitgrößten Eigenwert bei Verwendung des Lanczos-Verfahrens

Diese Eigenschaft und andererseits die Symmetrie der Konvergenz kann man sehr gut in 5 erkennen. Dort wird das Lanczos-Verfahren auf eine 500×500 Matrix, mit Eigenwerten gleichverteilt im Intervall $(0, 2)$ und 4 extremalen Eigenwerten bei -1, -0.5, 2.5 und 3, angewendet. Bei $x = 12$ sieht man das gesamte Spektrum der Eigenwerte, wobei ein Eigenwert durch ein Kreuz mit dem entsprechenden Wert an der $y-$Achse dargestellt wird. Für $x = 2, ..., 11$ sind die Approximationen an die Eigenwerte bei entsprechender Krylov-Raumdimension dargestellt. Der größte sowie kleinste Eigenwert werden hier schon bei Krylov-Raumdimension 6 gut approximiert. Bei Krylov-Raumdimension 8 beginnt die rapide Konvergenz der Approximation an den zweitgrößten sowie zweitkleinsten Eigenwert.

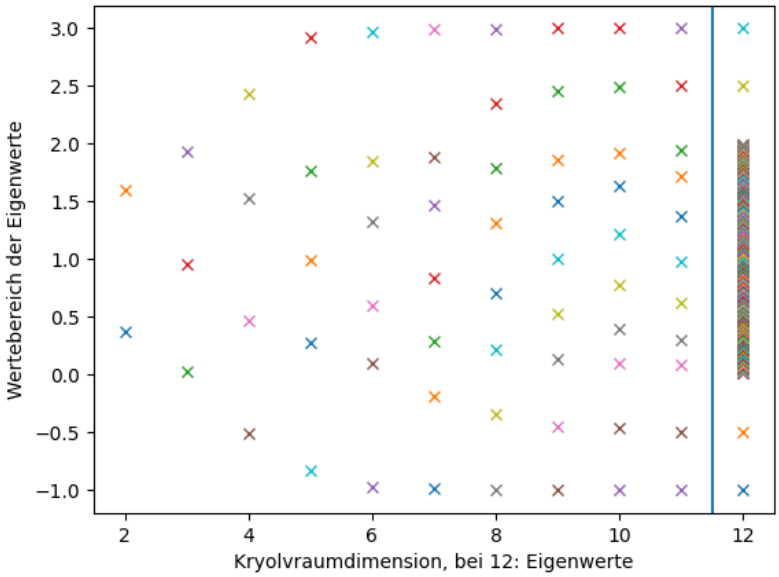


Abbildung 5: Die Approximation an die Eigenwerte beim Lanczos-Verfahren

Verwendet man nun also zur Testung keine uniform verteilten Eigenwerte, sondern lognormal verteilte, erhält man nun wesentlich bessere Konvergenzgeschwindigkeit. In Abbildung 6 wird das Lanczos-Verfahren wieder auf eine Matrix der Dimension 1000 angewendet, diesmal jedoch mit lognormal verteilen Eigenwerten ($\mu = 1, \sigma = 1$). Hier erkennt man die erwartete exponentielle Konvergenz, da der Fehler semilogarithmisch dargestellt wird.

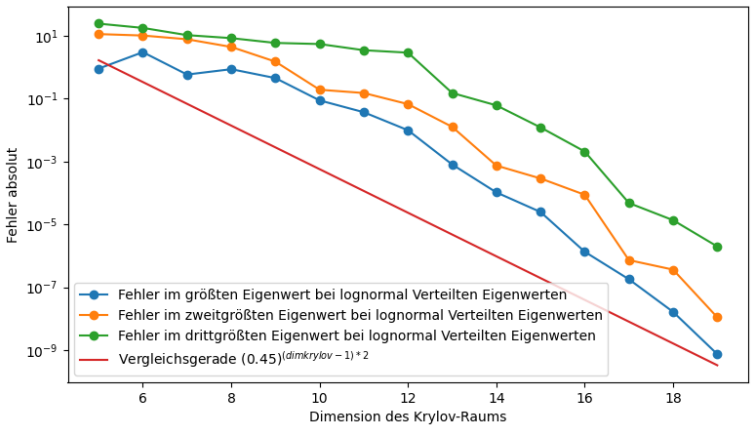


Abbildung 6: Fehler in den drei größten Eigenwerten bei Verwendung des Lanczos-Verfahrens mit lognormal Verteilen Eigenwerten

Die Testungen zeigen also, dass sich das Lanczos-Verfahren vorallem für Eigenwertprobleme eignet, bei denen sich einige Eigenwerte stark vom Spektrum abheben. Beim Arnoldi-Verfahren erkennt man bei Testung mit nun voll besetzten Matrizen ein ähnliches Verhalten. Auch hier stellt man wesentlich bessere Konvergenzgeschwindigkeit bei lognormal verteilen Eigenwerten fest, wie man in Abbildung 7 erkennen kann.

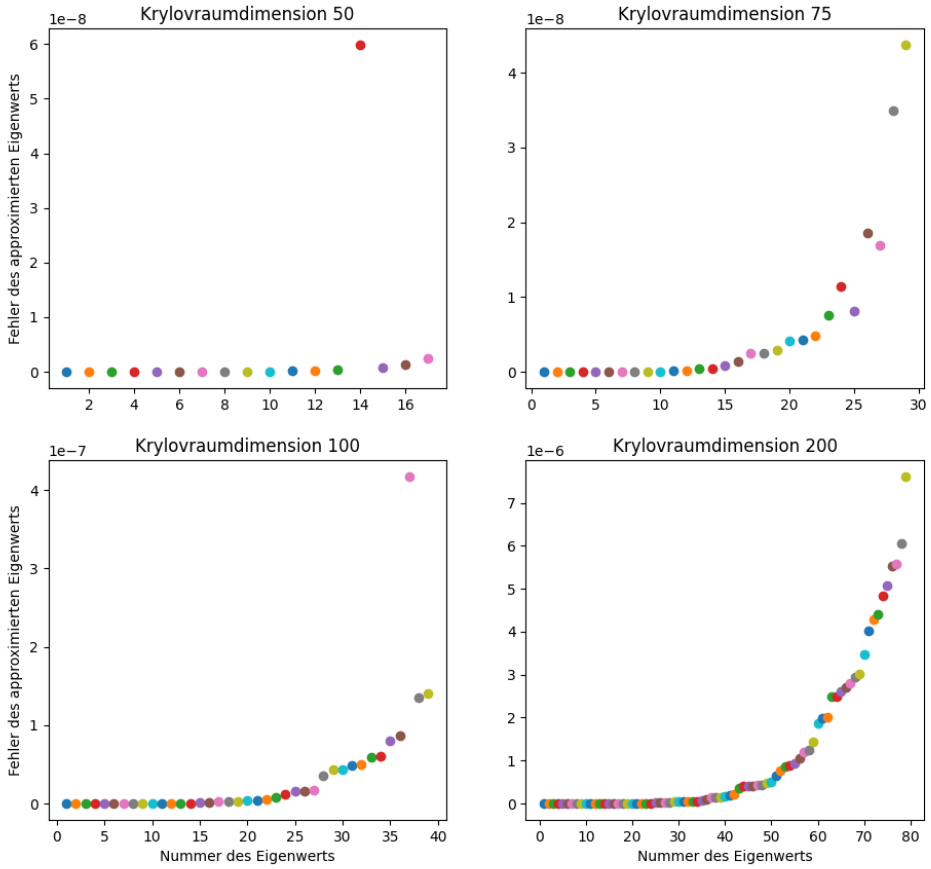
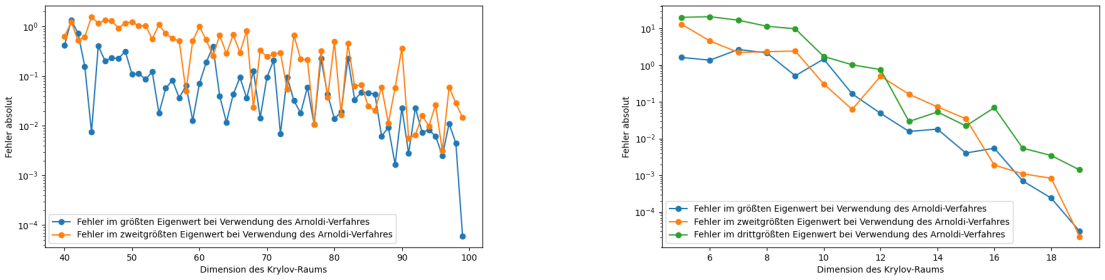


Abbildung 8: Fehler in den ersten 40% der Eigenwerte bei Anwendung des Arnoldi-Verfahrens auf das diskretisierte Problem



(a) uniform Verteilte Eigenwerte

(b) lognormal Verteilte Eigenwerte

Abbildung 7: Fehler in der Approximation bei Verwendung des Arnoldi-Verfahrens

Wir wollen nun das Arnoldi-Verfahren verwenden, um unser Problem aus der Einleitung, die Eigenwertberechnung für $-\Delta$ auf einem Rechteck Ω, zu lösen. Dabei verwenden wir zur Diskretisierung des Problems die

Finite-Elemente-Methode. Hier kommen das Netgen sowie NG-Solve Paket zum Einsatz. Mit diesen können wir das Rechteck $\Omega := (0,2) \times (0,1)$ triangulieren und einen Finite-Elemente-Raum darauf erstellen. Aus NG-Solve bekommen wir dann unsere Matrizen, die genaue Ausführung dazu befindet sich im Anhang. Zum Vergleich verwenden wir die in Satz 1.2 berechneten Eigenwerte. Als shift in (3) wird $\rho = 3$ gewählt. In diesem Schritt ist zusätzlicher Aufwand nötig, da in (3) eine Inverse berechnet werden muss.

In Abbildung 8 sieht man den Fehler für verschiedene Krylov-Raumdimensionen. Dabei ist auf der $x-$Achse die Nummer des betrachteten Eigenwerts (aufsteigend sortiert) und auf der $y-$Achse je der Fehler der Approximation aus dem Arnoldi-Verfahren zu dem analytisch berechneten Eigenwert. Man erkennt eine gute Approximation in den ersten 40% der berechneten Eigenwerte, wobei die Anzahl an berechneten Eigenwerten genau mit der Krylov-Raumdimension übereinstimmt. Die Schranke der 40% ist dabei aus skalierungstechnischen Gründen gewählt, man erhält sogar eine ziemlich gute Approximation an die ersten (also kleinsten) 45% Eigenwerte, bei den größeren steigt der Fehler jedoch schnell und stark an. Im Bezug zu den vorhergegangenen Testungen scheinen die Eigenwerte der geshifteten Matrix aus dem verallgemeinerten Problem eine günstige Verteilung der Eigenwerte zu haben, bei dem sich einige extremale ausprägen.

4 Conclusio

Wir wollen am Ende die in der Arbeit verwendeten Verfahren noch einmal kurz vergleichen. Das QR-Verfahren liefert eine sehr gute Approximation an alle Eigenwerte. Der Nachteil ist der sehr große Aufwand, den die benötigte QR-Zerlegung hat. Um dem entgegenzuwirken, ist es sinnvoll, das Verfahren mit Wilkinson-Shifts durchzuführen. Bei diesem sind bei selber Approximations-Fehlergröße weniger QR-Zerlegungen nötig. Zudem kann auch für die spezielle Klasse der Hessenberg-Matrizen ein weiterer Speedup erreicht werden, indem man die QR-Zerlegung mit Givens-Rotationen durchführt.

Für die meisten Anwendungen in der Praxis ist die Laufzeit des QR-Verfahrens jedoch nicht tragbar, zumal ist man dort oft nicht an allen Eigenwerten interessiert, sondern nur an einigen wenigen. Hier kommt das Lanczos-Verfahren zum Einsatz. Für hermitesche Matrizen bekommt man mit diesem eine gute Approximation an die größten Eigenwerte. Der Aufwand beim Lanczos-Verfahren ist wesentlich geringer als der beim QR-Verfahren. Ein Nachteil des Verfahrens ist die eventuelle numerische Instabilität, welcher jedoch mit zusätzlichem Aufwand entgegengewirkt werden kann. Ein weiterer Nachteil ist, dass das Verfahren nur für hermitesche Matrizen funktionert. Will man nun Eigenwerte einer nicht hermiteschen Matrix berechnen, muss man auf das Arnoldi-Verfahren zurückgreifen.

Die Testungen zeigen zudem, dass die Konvergenz des Verfahrens sehr stark vom Spektrum der Eigenwerte abhängt. Wenn sich die extremalen Eigenwerte wirklich stark vom Spektrum abheben, erhalten wir eine gute Konvergenzrate. Es macht also auch dann Sinn, das Lanczos-Verfahren zu verwenden, wenn man schon eine ungefähre Vorstellung hat, wie die Eigenwerte verteilt sind. Wie wir gesehen haben, kann dies in der Praxis relevant sein.

Code

QR-Verfahren ohne shifts

```
def QR_simple(A, tol = 1e-7):
  n = A.shape[1]
  count = 0
  for i in range(n-1,0,-1):
    while abs(A[i,i-1]) > tol:
        Q,R = np.linalg.qr(A)
        A = R@Q
        count +=1
    A[i,:i-n] = 0
return A, sorted(np.diag(A)), count
```

Listing 2: Implementation des QR-Verfahren ohne shifts

QR-Verfahren mit Rayleigh-Quotienten-Shift

```
def QR_shift(A,tol=1e-10):
  n = A.shape[1]
  count = 0
  for i in range(n-1,0,-1):
      while abs(A[i,i-1]) > tol:
          rho = A[i,i]
          Q,R = np.linalg.qr(A-rho*np.identity(n))
          A = R@Q + rho*np.identity(n)
          count +=1
      A[i,:i-n] = 0
  return A, sorted(np.diag(A)), count
```

Listing 3: Implementation des QR-Verfahrens mit Rayleigh-Quotienten-Shift

QR-Verfahren mit Wilkinson-Shift

```
def QR_shift2(A,tol=1e-14):
    n = A.shape[1]
    count = 0
    for i in range(n-1,0,-1):
        while abs(A[i,i-1]) > tol*(abs(A[i-1,i-1])+abs(A[i,i])):
            w = np.linalg.eigvals(A[i-1:i+1,i-1:i+1])
            if abs(w[0] - A[i,i]) < abs(w[1] - A[i,i]):
                rho = w[0]
            else:
                rho = w[1]
            Q,R = np.linalg.qr(A-rho*np.identity(n))
            A = R@Q + rho*np.identity(n)
            count += 1
        A[i,:i-n] = 0
    return A, sorted(np.diag(A)), count
```

Listing 4: Implementation des QR-Verfahrens mit Wilkinson-Shift

QR-Zerlegung einer Hessenberg-Matrix mit Givens-Rotationen

```
1  def QR_decomp_hesse(A):
2      n = A.shape[0]
3      Q = np.eye(n,n, dtype = complex)
4
5      for i in range(0,n-1):
6          if abs(A[i,i]) >= abs(A[i+1,i]):
7              t = A[i+1,i]/abs(A[i,i])
8              root = (1+abs(t)**2)**(1/2)
9              c = A[i,i]/(abs(A[i,i])*root)
10             s = t/root
11         else:
12             t = A[i,i]/abs(A[i+1,i])
13             root = (1+abs(t)**2)**(1/2)
14             s = A[i+1,i]/(abs(A[i+1,i])*root)
15             c = t/root
16
17         for j in range(n):
18             if j < i:
19                 temp_2 = Q[i,j]
20                 Q[i,j] = c.conj()*temp_2 + s.conj()*Q[i+1,j]
21                 Q[i+1,j] = -s*temp_2 + c*Q[i+1,j]
22             else:
23                 temp_1 = A[i,j]
24                 A[i,j] = c.conj()*temp_1 + s.conj()*A[i+1,j]
25                 A[i+1,j] = -s*temp_1 + c*A[i+1,j]
26                 temp_2 = Q[i,j]
27                 Q[i,j] = c.conj()*temp_2 + s.conj()*Q[i+1,j]
28                 Q[i+1,j] = -s*temp_2 + c*Q[i+1,j]
29
30     return Q.T.conj(), A
```

Listing 5: Implementation der QR-Zerlegung für Hessenberg-Matrizen mit Givens-Rotationen

Berechnung von RQ für eine Hessenberg-Matrix mit Givens-Rotationen

```
1  def Hessenberg_QR_RQ(A):
2    n = A.shape[0]
3    c_arr = np.zeros(n, dtype = complex)
4    s_arr = np.zeros(n, dtype = complex)
5    for i in range(0,n-1):
6        if abs(A[i,i]) >= abs(A[i+1,i]):
7            t = A[i+1,i]/abs(A[i,i])
8            root = (1+abs(t)**2)**(1/2)
9            c_arr[i] = A[i,i]/(abs(A[i,i])*root)
10           s_arr[i] = t/root
11       else:
12           t = A[i,i]/abs(A[i+1,i])
13           root = (1+abs(t)**2)**(1/2)
14           s_arr[i] = A[i+1,i]/(abs(A[i+1,i])*root)
15           c_arr[i] = t/root
16
17       for j in range(i,n):
18               temp_1 = A[i,j]
19               A[i,j] = c_arr[i].conj()*temp_1 + s_arr[i].conj()*A[i+1,j]
20               A[i+1,j] = -s_arr[i]*temp_1 + c_arr[i]*A[i+1,j]
21
22   for i in range(0,n-1):
23       for j in range(0,i+2):
24           temp_1 = A[j,i]
25           A[j,i] = c_arr[i]*temp_1 + s_arr[i]*A[j,i+1]
26           A[j,i+1] = -s_arr[i].conj()*temp_1 + c_arr[i].conj()*A[j,i+1]
27
28   return A
```

Listing 6: Implementation eines Schrittes des QR-Verfahrens für Hessenberg-Matrizen

QR-Verfahren mit Wilkinson-Shift für Hessenberg-Matrix

```
1  def QR_hesse(A,tol=1e-14):
2    n = A.shape[1]
3    count = 0
4    for i in range(n-1,0,-1):
5        while abs(A[i,i-1]) > tol*(abs(A[i-1,i-1])+abs(A[i,i])):
6            w = np.linalg.eigvals(A[i-1:i+1,i-1:i+1])
7            if abs(w[0] - A[i,i]) < abs(w[1] - A[i,i]):
8                rho = w[0]
9            else:
10               rho = w[1]
11           A = Hessenberg_QR_RQ(A-rho*np.identity(n))
12           A += rho*np.identity(n)
13           count += 1
14       A[i,:i-n] = 0
15   return A, sorted(np.diag(A)), count
```

Listing 7: Implementation des QR-Verfahrens mit Wilkinson-Shift für Hessenberg-Matrizen

Lanczos-Verfahren

```
def lanczos(A,m = 0):
    n = A.shape[1]
    if m == 0:
        m = n

    v0 = np.random.rand(n)
    v = [v0/np.linalg.norm(v0)]
    gam = [v[0].T.conj()@A@v[0]]
    w = (A - gam[0]*np.identity(n))@v[0]
    delta = [np.linalg.norm(w)]
    j = 0
    while delta[j] > 1e-10 and j < m-1:
        v.append(w/delta[j])
        j +=1
        gam.append(v[j].T.conj()@A@v[j])
        w = (A - gam[j]*np.identity(n))@v[j] - delta[j-1]*v[j-1]
        delta.append(np.linalg.norm(w))
    T = np.diag(delta[:-1], -1) + np.diag(gam) + np.diag(delta[:-1], 1)
    return QR_hesse(T)
```

Listing 8: Implementation des Lanczos-Verfahrens

Arnoldi-Verfahren

```
def arnoldi(A, dim, k = 0):
  n = dim
  if k == 0:
      k = n

  v0 = np.random.rand(n)
  v = [v0/np.linalg.norm(v0)]
  h = np.zeros((k,k))

  for j in range(k):
      w = A(v[j])
      for l in range(j+1):
          h[l][j] = v[l].T.conj()@w
          w = w - h[l][j]*v[l]
      if j < k-1:
          h[j+1][j] = np.sqrt(w.T.conj()@w)
          if abs(h[j+1][j]) < 1e-14:
              return QR_hesse(h[:j+1,:j+1])
          else:
              v.append(w/h[j+1][j])
  return QR_hesse(h)
```

Listing 9: Implementation des Arnoldi-Verfahrens

Code zur Diskretisierung mit NG-Solve

```
a = 2
b = 1
maxh = 0.05
order = 4

geo = SplineGeometry()
geo.AddRectangle((0,0), (a,b), bcs=["b","r","t","l"])

mesh = Mesh(geo.GenerateMesh(maxh=maxh))
Draw(mesh)

fes = H1(mesh, complex=True, order=order)

eigenvec = GridFunction(fes,multidim=nr_eigs)

u = fes.TrialFunction()
v = fes.TestFunction()

a = BilinearForm (fes)
a += SymbolicBFI (grad(u)*grad(v))

b = BilinearForm (fes)
b += SymbolicBFI (u*v)

a.Assemble()
b.Assemble()

A=a.mat
B=b.mat

shift = 3

shifted = a.mat.CreateMatrix()
shifted.AsVector().data = a.mat.AsVector() - (shift**2)*b.mat.AsVector()
invshifted = shifted.Inverse(freedofs=fes.FreeDofs())

tmp1 = eigenvec.vec.CreateVector()
tmp2 = eigenvec.vec.CreateVector()

def matvec(v):
    tmp1.FV().NumPy()[:] = v
    tmp2.data = b.mat * tmp1
    tmp1.data = invshifted * tmp2
    return tmp1.FV().NumPy()

A = scipy.sparse.linalg.LinearOperator( (a.mat.height,a.mat.width), matvec)
```

Listing 10: Code mit Parametern die bei der Testung des Arnoldi-Verfahrens verwendet werden

Literatur

[1] URL: http://www.netlib.org/lapack95/lug95/lug95.html (besucht am 10. 02. 2021).

[2] Lothar Nannen. „Eigenwertprobleme“. 2020.

[3] Lothar Nannen. „Numerische Mathematik A, Wintersemester 2019/2020“. URL: https://tiss.tuwien.ac.at/education/course/documents.xhtml?dswid=6959&dsrid=486&courseNr=101313&semester=2019W#.

[4] Yousef Saad. *Numerical methods for large eigenvalue problems, second edition.* Society for Industrial und Applied Mathematics, 2011.

[5] Horst D. Simon. „The Lanczos Algorithm With Partial Reorthogonalization“. In: *Mathematics of Computation* 42.165 (1984), S. 115–142.