Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Mathematik

Diplomarbeit

Laufzeitoptimierung der Robusten Perron Cluster Analyse (PCCA+)

Vorgelegt von Mascha Berg

Geb. am: 21.03.1986 in: Berlin

Erstgutachter: PD Dr. Marcus Weber
Zweitgutachter: Dr. Susanna Röblitz

Berlin, den 10.07.2012

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Ausführungen, die anderen veröffentlichten oder nicht veröffentlichten Schriften wörtlich oder sinngemäß entnommen wurden, habe ich kenntlich gemacht.
Die Arbeit hat in gleicher oder ähnlicher Fassung noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum *Mascha Berg*

Zusammenfassung

Die Robuste Perron Cluster Analyse (PCCA+) ist ein Algorithmus für spektrale Clusteranalyse, also das Gruppieren von Objekten mithilfe von Eigeninformationen. Das Ziel dieser Arbeit ist, zwei Ansätze zur Laufzeitverbesserung der PCCA+ auf ihre praktische Umsetzbarkeit zu überprüfen. Der erste Ansatz ist die Verkürzung der Eigenwertberechnung. Hier zeigt sich nur für manche Beispiele ein Potential zur Laufzeitverkürzung, ohne dass diese Beispiele einer bestimmten Klasse zugeordnet werden können.

Der andere Ansatz ist der Austausch einer in der PCCA+ verwendeten numerischen Minimierungsmethode. Dieser Ansatz führt zu einer deutlich schnelleren, aber unter Umständen ungenaueren Variante der PCCA+.

Inhaltsverzeichnis

1 Einleitung

Algorithmen für *Spectral Clustering*, also Clusteranalyse mithilfe von Eigenwertinformationen, haben sich laut Luxburg (2007) in letzter Zeit zu den verbreitetesten modernen Clusteringmethoden entwickelt. Eine Methode, die zu dieser Klasse von Algorithmen gehört, ist die Robuste Perron Cluster Analyse, abgekürzt PCCA+. Wenn große Datenmengen geclustert werden sollen oder die Anzahl der Cluster sehr hoch ist, kann jedoch die Laufzeit der PCCA+ sehr groß werden. Daher beschäftigt sich diese Arbeit mit zwei verschiedenen Ansätzen, die Laufzeit der PCCA+ zu reduzieren:

1. Verkürzung der Eigenwertberechnung
2. Austausch der verwendeten Minimierungsmethode

Für die Berechnung der Cluster in der PCCA+ werden genau so viele Eigenwerte und -vektoren benötigt, wie Cluster gesucht sind. Die Berechnung von Eigenpaaren ist jedoch sehr teuer und wird daher meist soweit wie möglich eingeschränkt. Da die PCCA+ auf Eigeninformationen basiert, ist es nicht möglich, die Berechnung der Eigenpaare komplett zu umgehen.

Die benötigten Eigenpaare werden in der PCCA+ mit einem iterativen Algorithmus berechnet. Es kam jedoch die Vermutung auf, dass die Informationen, die die PCCA+ für die Berechnung der Cluster benötigt, eventuell nicht erst in den letzten Iterierten der Eigenwerte vorhanden sind, sondern bereits früher. Einer der beiden in dieser Arbeit vorgestellten Ansätze, die PCCA+ zu optimieren, ist daher, das Eigenwertproblem nicht vollständig zu lösen und zu evaluieren, ob die Clusterings schon mit früheren Iterierten das gleiche Ergebnis liefern.

Zusammenfassend gibt es im Rahmen des ersten Ansatzes zwei Hypothesen, die in dieser Arbeit überprüft werden sollen:

- Je besser die Cluster getrennt sind (also je größer die *Schärfe* ist), desto mehr Iterationen lassen sich einsparen
- Es existiert eine Klasse von Problemen, bei denen Iterationen gespart werden können.

Die erste Hypothese enthält dabei implizit die Annahme, dass bei gut getrennten Clustern weniger Iterationsschritte bei der Eigenwertberechnung aufgewendet werden.

Der zweite in dieser Arbeit vorgestellte Ansatz zur Laufzeitoptimierung ist der Austausch der verwendeten Minimierungsmethode. Im Ablauf der PCCA+ muss eine Zielfunktion minimiert werden. Es hat sich in Versuchen mit Testdatensätzen gezeigt, dass diese Minimierung mehr als 99 % der Laufzeit der PCCA+ ausmacht, daher verspricht die Wahl einer geeigneteren Methode eine Verkürzung der Laufzeit. Nach dem Austausch der Methode soll die Laufzeitverbesserung anhand von numerischen Beispielen nachgewiesen werden.

Die vorliegende Arbeit ist wie folgt gegliedert:
In Kapitel 2 wird das große Thema *Spectral Clustering* eingeführt, und der für diese Arbeit wesentliche Algorithmus PCCA+ vorgestellt. Das Thema des 3. Kapitels ist der iterative Eigenwertlöser *Implicitly Restarted Arnoldi Method* (kurz IRAM), der für die Untersuchung des ersten Verbesserungsansatzes verwendet

wird. In Kapitel 4 werden die verwendeten Minimierungsmethoden *fminsearch* und NLSCON erklärt. Im 5. Kapitel werden schließlich die Methoden und Ergebnisse dieser Arbeit präsentiert. Zunächst wird ein Tool zur Erzeugung von Testdaten vorgestellt. Danach werden Vorgehensweise und numerische Beispiele für die beiden Ansätze zur Verbesserung der Laufzeit dargestellt.

2 Spectral Clustering und Robuste Perron Cluster Analyse

In diesem Kapitel soll zunächst eine Einführung in das Thema Clustering gegeben werden, bevor es spezieller um das Teilgebiet des *Spectral Clusterings* geht. Danach soll die PCCA+ vorgestellt werden.

2.1 Clustering

Clusteranalyse ist das Gruppieren von Objekten nach bestimmten Eigenschaften, d.h. nach vom Benutzer festgelegten Ähnlichkeits-Kriterien. Die gefundenen Gruppen werden Cluster genannt, und eine Einteilung in Cluster heißt Clustering.

Definition 1 (Clustering) *Sei* $O = \{o_1, \ldots, o_n\}$ *eine Menge von Objekten, wobei üblicherweise* $o_i = (m_1, \ldots, m_d)$ *ein Vektor ist, der Messwerte oder Daten des Objekts enthält. (Im Folgenden ist* n *die Anzahl der Objekte,* d *die Anzahl der Messwerte/Daten pro Objekt, und* k *die Anzahl der Cluster.) Es ist zu unterscheiden zwischen hartem Clustering und fuzzy Clustering.*
Eine Abbildung $c : O \to \{1, \ldots, k\}$ *ist ein* ***hartes Clustering*** *von* O *in* k *Cluster. Alternativ lässt sich ein hartes Clustering durch eine Zugehörigkeitsmatrix darstellen: Sei* $Z \in \mathbb{R}^{n \times k}$ *und* $Z_{i,j} = 1$ *genau dann, wenn* o_i *in Cluster* j *liegt, und sonst* $Z_{i,j} = 0$.
Ein ***fuzzy Clustering*** *ist durch eine Zugehörigkeitsmatrix* $Z \in \mathbb{R}^{n \times k}$ *definiert, die Einträge* $Z_{i,j}$ *liegen hierbei im Intervall* $[0, 1]$*. So wird jedem Objekt ein Zugehörigkeitswert zu jedem Cluster zugeordnet, wobei die Summe der Zugehörigkeitswerte eines Objektes jeweils 1 beträgt.*

Ein anschauliches Beispiel ist die Einteilung von Pflanzen in Gruppen, sortiert nach physiologischen Merkmalen, beispielsweise der Größe der Kelchblätter. Eine Veranschaulichung ist in den Abbildungen 1, 2 und 3 dargestellt. Gezeigt werden beispielhafte Clusterings nach verschiedenen Eigenschaften, wobei gleichfarbige Blumen jeweils dem gleichen Cluster zugeordnet sind. Hierbei ist gut zu erkennen, dass es für gegebene Objekte (hier: 12 Blumen) und zugehörige Daten (hier: Höhe der Blumen und Abstand zueinander) kein eindeutiges Clustering gibt, sondern dass die Lösung von diversen Entscheidungen abhängt. In jedem der drei Bilder ist ein Clustering zu sehen, das in Bezug auf die Vorgaben (z.B. „Finde ein Clustering, das ähnliche große Blumen in einem Cluster zusammenfasst.“) eine gute Lösung ist.

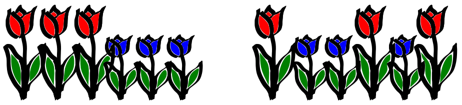

Abbildung 1: Beispiel für Clustering nach Höhe

Clusteranalyse hat ein breites Anwendungsfeld, und die Wahl der Methode hängt stark von der Herkunft und Qualität der Daten ab. Einige Anwendungsfelder für Clustering sind die folgenden:

- Marketing: Beispielsweise Gruppierung von Personendaten in Zielgruppen

Abbildung 2: Beispiel für Clustering nach Abstand

Abbildung 3: Beispiel für Clustering nach Höhe und Abstand

- Cluster-based Routing: Sorgt für eine effiziente Energieverwaltung in Sensornetzwerken
- Handschrifterkennung: Zuordnung in vordefinierte Buchstaben-/Zeichen-Cluster
- Suchmaschinen: Gruppierung der durchsuchten Datenbank für differenziertere Suchergebnisse

Laut Volkmann (2000) finden sich Anwendungsfelder 'überall dort, wo heterogene Gruppen von Objekten in homogenen Teilgesamtheiten zerlegt werden sollen.'

Wie bereits am obigen Beispiel zu sehen war, ist eine wesentliche Eigenschaft der Clusteranalyse die Abhängigkeit der Ergebnisse von Entscheidungen des Benutzers. Hierbei stellen sich nach Jain u. a. (1999) mehrere Fragen, deren Beantwortung zu verschiedenen Algorithmen und verschiedenen Ergebnissen führt, so zum Beispiel:

- Wie werden die Daten normiert?
- Welches Ähnlichkeits-Maß ist geeignet?
- Wie können Kenntnisse über die Daten effektiv benutzt werden?
- Wie können sehr große Datenmengen (beispielsweise eine Million Objekte) effizient geclustert werden?

Eine Normierung der Daten ist vor allem dann nötig, wenn Werte in verschiedenen Größenordnungen verwendet werden. Wenn beispielsweise eine gegebene Objekteigenschaft in der Größenordnung 10^5 ist, und eine andere in der Größenordnung 10^1, würde ohne eine Normierung die erste Eigenschaft einen viel größeren Einfluss auf die Ergebnisse haben als die zweite.

Es stehen diverse sehr verschiedene Clusteranalyse-Algorithmen zur Verfügung, die sich nach Jain u. a. (1999) durch folgende jeweils entgegengesetzte Eigenschaften charakterisieren lassen:

- hierarchisch/partitional:
  Hierarchische Algorithmen entwickeln eine hierarchische Cluster-Struktur, die in jedem Schritt verfeinert wird. Hierbei können einmal einem Cluster

zugeordnete Objekte nicht mehr getauscht werden, es können nur bereits bestehende Cluster zusammengefasst oder aufgeteilt werden. Bekannte Beispiele für hierarchische Clustering-Algorithmen sind der *single-link*- und der *complete-link*-Algorithmus.
Partitionelle Verfahren hingegen berechnen ein einziges Clustering, meist mittels der Minimierung einer Zielfunktion.

- agglomerativ/divisiv:
  Ein agglomerativer Clustering-Algorithmus beginnt mit n Clustern (also ein Cluster pro Objekt) und fasst dann Cluster zusammen, bis ein Stop-Kriterium erfüllt ist. Ein divisiver Algorithmus hingegen beginnt mit einem einzigen Cluster, welches geteilt wird, bis ein Abbruchkriterium erfüllt ist. Beides sind Eigenschaften von hierarchischen Cluster-Algorithmen.

- monothetisch/polythetisch:
  Diese Unterscheidung bezieht sich auf die Verarbeitung der d verschiedenen Eigenschaften der Objekte. Monothetische Algorithmen verarbeiten die Eigenschaften nacheinander, während bei polythetischen Algorithmen alle Eigenschaften gleichzeitig benutzt werden. Dies kann beispielsweise durch die Definition eines Abstandes zwischen je zwei Objekten realisiert werden. Die meisten Clustering-Algorithmen sind polythetisch.

- hart/*fuzzy*:
  Wie bereits in Definition 1 beschrieben, ordnet ein hartes Clustering jedes Objekt genau einem Cluster zu, während ein *fuzzy* Clustering Zugehörigkeitszahlen für jedes Objekt und Cluster ausgibt.

- deterministisch/stochastisch:
  Dieser Aspekt betrifft vorwiegend partitionale Algorithmen, die eine Fehler-Funktion minimieren. Die Minimierung kann hier entweder mit den üblichen numerischen Methoden (deterministisch) durchgeführt werden, oder mittels einer zufälligen (stochastischen) Suche im Zustandsraum, der aus allen möglichen Clusterings besteht.

Ein sehr bekanntes Beispiel für partitionelle Algorithmen sind die k-means Algorithmen. Sie basieren alle auf der gleichen Abfolge von Aktionen, es gibt allerdings verschiedene Varianten und Implementierungen. Nach Jain u. a. (1999) ist das Prinzip von k-means:

1. Wähle für jedes Cluster einen Startwert für das Clusterzentrum.

2. Ordne jeden Punkt demjenigen Cluster zu, für das der Abstand (in einem vorher gewählten Abstandsmaß)von diesem Punkt zum Clusterzentrum am kleinsten ist.

3. Berechne die Clusterzentren neu als Mittelwert der Punkte im jeweiligen Cluster.

4. Prüfe ein vorher gewähltes Abbruchkriterium. Wenn das Kriterium erfüllt ist, dann stoppe, sonst gehe zurück zu 2. Hier kann beispielsweise als Abbruchkriterium geprüft werden, ob sich bei Wiederholung von 2. die Zuordnung ändern würde, oder ob eine quadratische Fehlerfunktion ihren

Wert ändert. Sobald die Zuordnung gleich oder die Fehlerfunktion konstant bleibt, wird abgebrochen.

Ein Schwachpunkt von k-means ist die notwendige Wahl der Startwerte für die Clusterzentren.

2.2 Spectral Clustering

Der Charakterisierung von Clustering-Algorithmen im vorigen Abschnitt folgend lässt sich *Spectral Clustering* als eine Gruppe von partitionalen, deterministischen Verfahren beschreiben. Die Bezeichnung 'spectral' beruht darauf, dass das Clustering aus dem Spektrum der Ähnlichkeitsmatrix berechnet wird. Erstmals vorgeschlagen wurde die Berechnung von Partitionen mithilfe von Eigenwerten und -vektoren von Fiedler (1973) und Donath u. Hoffman (1973).

Vorab einige wichtige Definitionen, die, sofern nicht anders angemerkt, der Darstellung in Luxburg (2007) folgen.

2.2.1 Graphen und Ähnlichkeiten

Für *Spectral Clustering* ist es notwendig, zu den zu clusternden Objekten nicht nur jeweils einen Vektor mit Messwerten/Daten zur Verfügung zu haben (wie im vorigen Abschnitt beschrieben), sondern es muss bereits ein weiterer Schritt getan sein: Aus diesen Daten müssen paarweise Ähnlichkeiten (zwischen 0 und 1) oder Abstände berechnet worden sein. Dies kann auf verschiedene Arten geschehen. Wichtig ist vor allem, dass sichergestellt wird, dass relativ hohe Ähnlichkeitswerte auch den gewünschten Clusterkriterien entsprechen. Ungenauigkeiten bei niedrigen Ähnlichkeiten sind weniger schwerwiegend, da in den meisten Fällen nur hohe Werte einen Einfluss auf das Ergebnis haben.

Wie Abstände berechnet werden, hängt sehr davon ab, in welcher Form die Daten vorliegen, und ob die Herkunft der Daten eventuell eine bestimmte Variante der Abstandsberechnung impliziert. Die gebräuchlichste Variante ist sicherlich, als Abstand $dist_{ij}$ von zwei Datenpunkten den Euklidischen Abstand $||o_i - o_j||_2$ zu wählen.

Eine Möglichkeit, aus Datenpunkten in $\mathbb{R}^d$ Ähnlichkeiten zu berechnen, ist die Gaußsche Ähnlichkeitsfunktion.

Definition 2 (Gaußsche Ähnlichkeitsfunktion) *Für zwei Objekte o_i und o_j ist die* ***Gaußsche Ähnlichkeitsfunktion*** *wie folgt definiert:*

$$s(o_i, o_j) = e^{-\frac{dist_{ij}^2}{2\sigma^2}} \tag{1}$$

Hierbei spielt die Wahl eines geeigneten σ eine wesentliche Rolle für das Ergebnis; es gibt jedoch keine generell funktionierende Regel für die Wahl von σ, Luxburg (2007) nennt nur einige Faustregeln.

Es seien also s_{ij} als Ähnlichkeiten der Objekte o_i und o_j gegeben. Es gilt außerdem $s_{i,j} = s_{j,i}$.

Definition 3 (Graph und Eigenschaften von Graphen) *Ein* ***Graph*** *ist ein Tupel $G = (V, E)$, wobei V die Eckenmenge ist und E die Kantenmenge. Es sei $V = \{v_1, v_2, \ldots\}$, und da eine Kante jeweils zwei Ecken verbindet, gilt $E \subseteq V \times V$.*

Ein Graph kann entweder ***gerichtet*** *oder* ***ungerichtet*** *sein. Der Unterschied besteht darin, dass bei einem gerichteten Graphen die Kanten* (v_i, v_j) *und* (v_j, v_i) *als verschiedene Objekte betrachtet werden, wohingegen in einem ungerichteten Graphen stets* $(v_i, v_j) = (v_j, v_i)$ *gilt.*
Eine weitere Eigenschaft von Graphen sind ***Gewichte****. Wenn ein Graph gewichtet ist, wird jeder Kante* $(v_i, v_j) \in E$ *eine positive reelle Zahl als Gewicht zugeordnet. Im Folgenden sei das Gewicht der Kante* (v_i, v_j) *stets mit* w_{ij} *bezeichnet. Wenn* $(v_i, v_j) \notin E$*, dann sei* $w_{ij} = 0$*.*
Die Matrix $W = (w_{ij})_{i,j=1,\ldots,n}$ *wird als* ***Adjazenzmatrix*** *bezeichnet.*
Zur Vereinfachung der Schreibweise definiere außerdem die Bezeichnung $i \in A$ *für alle Elemente* i *aus* $\{i|v_i \in A\}$*. Die* ***Indikatorfunktion*** $\mathbb{1}_A$ *für eine Menge* $A \in V$ *sei definiert als* $\mathbb{1}_A = (f_1, \ldots, f_n)^T$ *mit* $f_i = 1$ *wenn* $v_i \in A$*, und sonst* $f_i = 0$*.*

Definition 4 (Grad einer Ecke) *Der* ***Grad*** d_i *einer Ecke* v_i *ist hier (entgegen der üblicherweise in der Graphentheorie verwendeten Definition) definiert als die Summe der Gewichte seiner Kanten:*

$$d_i = \sum_{j=1}^{n} w_{ij} \tag{2}$$

Die Matrix D *sei die Diagonalmatrix der Eckengrade:*

$$D = diag(d_i, \ldots, d_n) \tag{3}$$

Um Graphen für Clusteringalgorithmen benutzen zu können, ist es nun nötig, mithilfe der gegebenen Ähnlichkeiten (s_{ij}) oder Abstände $(dist_{ij})$ einen sogenannten Ähnlichkeitsgraph zu definieren. Dies kann auf verschiedene Arten geschehen. Eine Möglichkeit ist der **ε-neighborhood-Graph**. Dieser entsteht durch die folgende Definition der Kanten:

$$(v_i, v_j) \in E \Leftrightarrow dist_{ij} < \varepsilon \tag{4}$$

für ein festes $\varepsilon > 0$. Der ε-neighborhood-Graph ist meist ungewichtet.

Eine zweite Möglichkeit ist der **fully connected-Graph**, bei dem einfach alle Ecken verbunden sind, und als Gewicht einer Kante jeweils die Ähnlichkeit der beiden Ecken definiert ist. Die Verwendung dieses Graphen ist nicht immer sinnvoll, da vorausgesetzt sein muss, dass die Ähnlichkeiten bereits die Informationen über lokale Nachbarschaften enthalten. Dies wird beispielsweise durch die oben erwähnte Gaußsche Ähnlichkeitsfunktion gewährleistet.

Die dritte und letzte hier vorgestellte Möglichkeit ist der **k-nearest-neighbor-Graph**: Hierbei ist $(v_i, v_j) \in E$ genau dann, wenn v_j eine der k nächsten Ecken von v_i ist. Der so entstehende Graph $G = (V, E)$ ist gerichtet und so für die üblichen Clusteringalgorithmen nicht brauchbar. Es gibt zwei gebräuchliche Möglichkeiten, einen ungerichteten Graphen $G' = (V, E')$ daraus zu erstellen. Die erste Variante ist, die Richtungen der Kanten schlicht zu ignorieren; das bedeutet:

$$(v_i, v_j) \in E' \Leftrightarrow (v_i, v_j) \in E \vee (v_j, v_i) \in E \tag{5}$$

Dieser Graph wird als *k-nearest-neighbor*-Graph bezeichnet. Die zweite Variante ist der *mutual k-nearest-neighbor*-Graph, bei dem die Kanten folgendermaßen definiert sind:

$$(v_i, v_j) \in E' \Leftrightarrow (v_i, v_j) \in E \wedge (v_j, v_i) \in E \tag{6}$$

Hier sind zwei Objekte nur dann verbunden, wenn beide jeweils zu den k nächsten Nachbarn des anderen gehören, dieser Graph hat also weniger Kanten. Bei beiden Varianten werden als Gewichte der Kanten stets die Ähnlichkeiten gewählt.

Nun stellt sich die Frage, welcher Ähnlichkeitsgraph in der Praxis gewählt werden sollte. Generell ist zu bedenken, dass in jedem Fall mindestens ein Parameter gewählt werden muss, zum Beispiel k beim *k-nearest-neighbor*-Graphen, ε beim *ε-neighborhood*-Graphen, und σ bei dem *fully-connected*-Graphen in Verbindung mit der Gaußschen Ähnlichkeitsfunktion. Luxburg (2007) stellt einen Vergleich dieser Möglichkeiten an und gibt Empfehlungen, wie die Parameter gewählt werden sollten. Mangels theoretischer Ergebnisse basieren diese Empfehlungen allerdings nur auf praktischen Erfahrungen. Zusammenfassend lässt sich sagen, dass diese Erfahrungen die Wahl des *k-nearest-neighbor*-Graph nahelegen, vor allem da dieser die positive Eigenschaft hat, nicht sehr anfällig für ungünstig gewählte Parameter zu sein.

2.2.2 Laplace-Matrizen

Ein weiteres wichtiges Instrument für *Spectral Clustering* sind Laplace-Matrizen. Hierbei ist zu beachten, dass es keine einheitliche Definition der Laplace-Matrix gibt, sondern dass mit diesem Begriff verschiedene Matrizen gemeint sein können. Im Folgenden werden einige davon vorgestellt. Vorausgesetzt ist immer, dass $G = (V, E)$ ein gewichteter ungerichteter Graph mit zugehöriger Adjazenzmatrix $W = (w_{ij})_{i,j=1,\dots,n}$ ist.

Definition 5 (Unnormierte Graph-Laplace-Matrix)

$$L = D - W \tag{7}$$

Wichtige Eigenschaften der unnormierten Laplace-Matrix werden in den beiden folgenden Sätzen zusammengefasst, deren Beweise jeweils in Luxburg (2007) zu finden sind.

Satz 1 *L hat folgende Eigenschaften:*

- *Für alle $x \in \mathbb{R}^n$ gilt:*

$$x^T L x = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (x_i - x_j)^2 \tag{8}$$

- *L ist symmetrisch und positiv semidefinit.*
- *L hat den kleinsten Eigenwert 0, und der zugehörige Eigenvektor ist der 1-Vektor.*
- *L hat n nicht-negative reelle Eigenwerte.*

Satz 2 *G sei ein ungerichteter gewichteter Graph. Dann ist die Anzahl c der zusammenhängenden Komponenten* $A_1, \dots A_c \in V$ *von G gleich der Vielfachheit des Eigenwerts 0 von L.*
Der Eigenraum des Eigenwerts 0 wird aufgespannt von $\mathbb{1}_{A_1}, \dots \mathbb{1}_{A_c}$.

Definition 6 (Normierte Graph-Laplace-Matrizen)

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{9}$$

$$L_{rw} = D^{-1} L = I - D^{-1} W \tag{10}$$

Es gibt zwei Varianten, eine normierte Laplace-Matrix zu definieren. Die erste ist eine symmetrische Matrix, und die zweite hängt mit *Random Walks* zusammen, worauf später noch näher eingegangen wird.

Die folgenden zwei Sätze, deren Beweise wiederum in Luxburg (2007) zu finden sind, fassen die wesentlichen Eigenschaften der normierten Laplace-Matrizen zusammen.

Satz 3

- *Für alle* $x \in \mathbb{R}^n$ *gilt:*

$$x^T L_{sym} x = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2 \tag{11}$$

- λ *ist Eigenwert von* L_{rw} *zum Eigenvektor u, genau dann, wenn* λ *Eigenwert von* L_{sym} *ist, und der zugehörige Eigenvektor ist* $D^{1/2}u$.
- λ *ist Eigenwert von* L_{rw} *zum Eigenvektor u, genau dann, wenn* λ *und u das verallgemeinerte Eigenwertproblem* $Lu = \lambda Du$ *lösen.*
- *0 ist Eigenwert von* L_{rw} *zum Eigenvektor* $\mathbb{1}$, *und 0 ist Eigenwert von* L_{sym} *zum Eigenvektor* $D^{1/2}\mathbb{1}$.
- L_{rw} *und* L_{sym} *sind positiv semidefinit und haben n nicht-negative reelle Eigenwerte.*

Satz 4 *G sei ein ungerichteter gewichteter Graph. Dann ist die Anzahl c der zusammenhängenden Komponenten* $A_1, \dots A_c \in V$ *von G gleich der Vielfachheit des Eigenwerts 0 von* L_{rw} *und* L_{sym}.
Der Eigenraum des Eigenwerts 0 wird für L_{rw} *aufgespannt von* $\mathbb{1}_{A_1}, \dots \mathbb{1}_{A_c}$ *und für* L_{sym} *von* $D^{1/2}\mathbb{1}_{A_1}, \dots D^{1/2}\mathbb{1}_{A_c}$

Auch hier stellt sich angesichts der verschiedenen Möglichkeiten die Frage, wann welche Laplace-Matrix gewählt werden sollte. Luxburg (2007) stellt fest, dass bei Ähnlichkeitsgraphen, die nahezu regulär sind, alle drei vorgestellte Laplace-Matrizen gleich gut funktionieren. (Ein Graph ist regulär, wenn alle Ecken gleich viele Nachbarn haben.) Für stark nicht-reguläre Graphen wird dort die vorrangige Wahl von L_{rw} empfohlen, und als zweite Wahl wird L_{sym} vorgeschlagen. Somit scheinen die normierten Laplace-Matrizen die bessere Wahl zu sein.

2.2.3 Algorithmen

Aus den drei oben beschriebenen Laplace-Matrizen lassen sich drei verschiedene Clustering-Algorithmen ableiten, deren Aufbau recht ähnlich ist. Hier soll nur einer davon näher beschrieben werden, und zwar der, der die Matrix L_{rw} verwendet. Luxburg (2007) bezeichnet diesen Algorithmus als 'Normalized spectral clustering according to Shi and Malik (2010)', da er im Wesentlichen auf Shi u. Malik (1997) basiert.

Definition 7 (Normalized spectral clustering acc. to Shi and Malik) *Als Eingabe werden die Ähnlichkeitswerte und die Anzahl k der Cluster benötigt.*

1. *Erzeuge den Ähnlichkeitsgraphen mit einer vorher gewählten Methode. Daraus resultiert die Adjazenzmatrix W.*
2. *Berechne $L = D - W$.*
3. *Berechne die k ersten verallgemeinerten Eigenvektoren $u_1, \ldots, u_k$ des verallgemeinerten Eigenwertproblems $Lu = \lambda Du$.*
4. *$U = [u_1, \ldots, u_k] \in \mathbb{R}^{n \times k}$, U ist also die Matrix, die als Spalten die verallgemeinerten Eigenvektoren enthält.*
5. *Sei y_i die i-te Zeile von U für $i = 1, \ldots, n$.*
6. *Wende einen k-means-Algorithmus auf $y_1, \ldots, y_n$ an.*

In der Definition taucht nun L_{rw} nicht auf, aber die verallgemeinerten Eigenvektoren, die im 3. Schritt berechnet werden, sind nach Satz 3 genau die Eigenvektoren von L_{rw}. Hier werden also die ursprünglichen Daten ersetzt durch Vektoren, die Einträge aus den Eigenvektoren von L_{rw} enthalten. Im letzten Schritt ist es nicht zwingend notwendig, einen k-means-Algorithmus zu verwenden. Nach Luxburg (2007) sollte jeder Clustering-Algorithmus hier funktionieren, wenn die Cluster gut voneinander getrennt sind.

Um zu verstehen, warum der Algorithmus funktioniert, ist es sinnvoll, den Idealfall zu betrachten, also den Fall, dass der Ähnlichkeitsgraph tatsächlich in k zusammenhängende Komponenten zerfällt. Nach Satz 4 ist dann k gerade die Vielfachheit des Eigenwertes 0, also die Dimension des Eigenraums von 0. Wenn nun die k ersten Eigenvektoren berechnet werden, sind das genau die, die den Eigenraum von 0 aufspannen. Satz 4 besagt, dass der Eigenraum von $\mathbb{1}_{A_1}, \ldots \mathbb{1}_{A_k}$ aufgespannt wird. Die Wahl einer Basis ist nicht eindeutig, aber jeder Vektor aus einer Basis des Eigenraums ist auch aus der Basis $\mathbb{1}_{A_1}, \ldots \mathbb{1}_{A_k}$ darstellbar. Also gilt für jede Spalte von U aus Schritt 4: $u_i = \sum\limits_{i=1}^{k} a_i \mathbb{1}_{A_i}$ für passende Koeffizienten a_i. Dass nun die Zeilen von U die Informationen über die Clusterzugehörigkeit nicht nur enthalten, sondern auch besser verfügbar enthalten als die ursprünglichen Daten, sieht man gut an folgendem Beispiel (auch für den Idealfall):
Es seien die Indikatorfunktionen gegeben durch:

$$\mathbb{1}_{A_1} = (1,0,0,1,1,0,0,0)^T \tag{12}$$

$$\mathbb{1}_{A_2} = (0,1,0,0,0,1,1,0)^T \tag{13}$$

$$\mathbb{1}_{A_3} = (0,0,1,0,0,0,0,1)^T \tag{14}$$

Das bedeutet also, o_1, o_4, o_5 gehören zum ersten Cluster, o_2, o_6, o_7 zum zweiten und o_3, o_8 zum dritten. Wenn nun tatsächlich die $\mathbb{1}_{A_i}$ als Basis für den Eigenraum gewählt werden würden, wären hier in Schritt 5 $y_1, \ldots, y_8$ als Zeilen von U: (1,0,0), (0,1,0), (0,0,1), (1,0,0), (1,0,0), (0,1,0), (0,1,0), (0,0,1). Dass hier jeder Clusteringalgorithmus zum gewünschten Ziel führt, ist klar. Es ist aber davon auszugehen, dass die Eigenvektoren nicht genau so berechnet werden. Nun seien also beispielsweise

$$u_1 = 2\mathbb{1}_{A_1} + \mathbb{1}_{A_3} = (2,0,1,2,2,0,0,1)^T \quad (15)$$

$$u_2 = \mathbb{1}_{A_1} + \mathbb{1}_{A_2} + \mathbb{1}_{A_3} = (1,1,1,1,1,1,1,1)^T \quad (16)$$

$$u_3 = 2\mathbb{1}_{A_2} + 3\mathbb{1}_{A_3} = (0,2,3,0,0,2,2,3)^T \quad (17)$$

Dann sind die mit k-means zu clusternden y_i wie folgt: (2,1,0), (0,1,2), (1,1,3), (2,1,0), (2,1,0), (0,1,2), (0,1,2), (1,1,3). Die Objekte, die ins gleiche Cluster gehören, werden also auch in diesem Fall durch exakt den gleichen Vektor repräsentiert.

In der Praxis wird der Idealfall kaum einmal auftreten, aber an obigem Beispiel wird deutlich, dass die Verbesserung der Separierbarkeit der Daten durch das normalisierte *Spectral Clustering* umso größer ist, je besser die Cluster voneinander getrennt sind. Je näher die Daten am Idealfall sind, desto ähnlicher sind sich die Objekte eines Clusters in der Darstellung als y_i.

Es gibt verschiedene Möglichkeiten, die Suche nach einem Clustering zu interpretieren, aus denen sich jeweils neue Ansätze für Clusteringmethoden ergeben. Im Folgenden sollen zwei dieser Interpretationsmöglichkeiten beschrieben werden, und zwar Schnitte von Graphen und *Random Walks*.

2.2.4 Schnitte von Graphen

Der erste Ansatz ist, die Suche nach einem Clustering zu interpretieren als die Suche nach einer Partition des Ähnlichkeitsgraphen. Diese Partition soll die Eigenschaft haben, dass Kanten innerhalb einer Gruppe (eines Clusters) hohe Gewichte haben, und Kanten zwischen Gruppen niedrige.

Eine Möglichkeit, dies mathematisch umzusetzen, ist das *Min-Cut*-Verfahren. Für $A, B \subset V$ definiere

$$W(A,B) = \sum_{\substack{i \in A \\ j \in B}} w_{ij} \quad (18)$$

Dann ist das *Min-Cut*-Verfahren für ein festes k die Minimierung von

$$cut(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} W(A_i, \bar{A}_i). \quad (19)$$

Hierbei wird einfach die oben genannte Bedingung umgesetzt, dass die Summe der Gewichte der Kanten zwischen zwei Clustern möglichst klein sein soll. Der Faktor $\frac{1}{2}$ ist mathematisch nicht notwendig, sondern wird nur aus Konsistenzgründen eingeführt, da sonst jede Kante doppelt gezählt wird. Dieser sehr intuitive Ansatz ist als Clusteringmethode nicht immer sinnvoll, da häufig nur einzelne Ecken von den anderen getrennt werden. Um dies zu verhindern, kann ein Normierungsfaktor eingesetzt werden, der die Wahl von großen Clustern belohnt. Daraus resultieren zwei bekannte Verfahren, *RatioCut* und *Ncut*, deren zu minimierende Funktionen sich nur in der Wahl dieses Faktors unterscheiden.

$$RatioCut(A_1, \dots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} \tag{20}$$

$$Ncut(A_1, \dots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{vol(A_i)} \tag{21}$$

Wobei $vol(A_i)$ die Summe der Grade der Ecken in A_i ist.

Während *MinCut* noch recht einfach zu lösen ist, sind *RatioCut* und *Ncut* NP-schwer. Um diese Verfahren dennoch verwenden zu können, bietet sich der Umstieg auf relaxierte Varianten der Probleme an. In Luxburg (2007) wird gezeigt, dass aus *Ncut* dann das oben beschriebene normierte *Spectral Clustering* wird, und aus *RatioCut* ergibt sich unnormiertes *Spectral Clustering*, was analog zum oben beschriebenen Algorithmus funktioniert - mit dem einzigen Unterschied, dass die Eigenvektoren von L statt von L_{rw} verwendet werden.

2.2.5 Random Walks

Definition 8 (Random Walk auf einem Graph) *Als* ***Random Walk*** *auf einem gewichteten Graphen $G = (V, E)$ bezeichnet man einen stochastischen Prozess auf den Ecken des Graphen. Die* ***Übergangswahrscheinlichkeit*** *von v_i zu v_j ist $p_{ij} = \frac{w_{ij}}{d_i}$. Definiere außerdem $P = (p_{ij})_{i,j=1,\dots,n}$*

Die Interpretation des Clusterings ist hier: Finde eine Partition des Ähnlichkeitsgraphen, so dass ein *Random Walk* darauf fast nur innerhalb eines Clusters bleibt und selten von Cluster zu Cluster springt. Die Benennung von L_{rw} wird hier nun klar, da ja $L_{rw} = I - D^{-1}W = I - P$ gilt. Die i-te Zeile von P enthält die Wahrscheinlichkeiten dafür, welche Ecke nach v_i gewählt wird. Daher ist P eine zeilenstochastische Matrix.

Aus dieser Interpretation von *Spectral Clustering* ergeben sich interessante Zusammenhänge zu Schnitten von Graphen.

Satz 5 *λ ist Eigenwert zum Eigenvektor u von L_{rw} genau dann, wenn $1 - \lambda$ Eigenwert von P ist, auch mit zugehörigem Eigenvektor u.*

Beweis: Angenommen, λ ist Eigenwert zum Eigenvektor u von L_{rw}. Das bedeutet:

$$\begin{aligned} Lu &= \lambda u && (22) \\ \Leftrightarrow (I - P)u &= \lambda u && (23) \\ \Leftrightarrow Pu &= Iu - \lambda u = u - \lambda u && (24) \\ \Leftrightarrow Pu &= (1 - \lambda)u && (25) \end{aligned}$$

Die letzte Zeile ist äquivalent dazu, dass $(1 - \lambda)$ Eigenwert von P zum Eigenvektor u ist. ■

Wenn der Graph zusammenhängend und nicht bipartit ist, hat der *Random Walk* auf G eine eindeutige stationäre Verteilung $\pi = (\pi_1, \dots, \pi_n)'$ mit

$\pi_i = \frac{d_i}{vol(V)}$. Auch eine Äquivalenz von *Ncut* und *Random Walks* wird in Shi u. Meila (2000) gezeigt:

Satz 6 *Sei G ein zusammenhängender, nicht bipartiter Graph. $(X_t)_{t\in\mathbb{N}}$ sei ein Random Walk, der in der stationären Verteilung π bei X_0 startet. Für disjunkte Mengen $A, B \subset V$ definiere $P(B|A) := P(X_1 \in B|X_0 \in A)$.*
Dann gilt: $Ncut(A, \bar{A}) = P(A|\bar{A}) + P(\bar{A}|A)$.

Ein Beweis dieses Satzes findet sich in Luxburg (2007).

Wenn nun für gegebene Datenpunkte erst Ähnlichkeiten berechnet wurden, dann ein Ähnlichkeitsgraph erzeugt, und schlussendlich mit einem geeigneten Algorithmus ein Clustering erstellt wurde, stellt sich die Frage, woran man erkennen kann, ob dieses Clustering gut ist. In den meisten Fällen gibt es keine korrekte Lösung, mit der das Ergebnis verglichen werden kann.

Um mehrere Clusterings der gleichen Daten miteinander zu vergleichen, stehen diverse Möglichkeiten zur Verfügung. Einige bekannte Kriterien zum Vergleich von Partitionen sind bespielsweise der *Fowlkes–Mallows Index* aus Fowlkes u. Mallows (1983), der *Rand Index* aus Rand (1971) oder *Variation of Information* aus Meilă (2007). Über die Güte eines Clusterings sagt dies jedoch nicht direkt etwas aus. Wenn keine korrekte Lösung zur Verfügung steht, hilft der Vergleich mehrerer Clusterings nicht weiter.

Eine Möglichkeit, Aussagen über die Güte eines Clusterings zu treffen, bietet der Silhouetten-Koeffizient von Kaufman u. Rousseeuw (1990).

Definition 9 (Silhouette) *Es sei o ein Objekt im Cluster A. Sei $d(A, o)$ der durchschnittliche Abstand eines Objekts zu allen anderen Objekten aus dem Cluster A, also:*

$$d(A, o) = \frac{\sum\limits_{\bar{o}\in A} d(o, \bar{o})}{|A|} \tag{26}$$

Berechne nun den durchschnittlichen Abstand zu den Objekten aller anderen Cluster und finde das Cluster mit dem geringsten durchschnittlichen Abstand. Dies sei Cluster B, und der Abstand sei $d(B, o)$:

$$d(B, o) = \min_{B\ Cluster,\ B\neq A} \frac{\sum\limits_{\bar{o}\in B} d(o, \bar{o})}{|B|} \tag{27}$$

*Dann ist die **Silhouette** von o definiert als:*

$$S(o) = \frac{d(B, o) - d(A, o)}{max(d(A, o), d(B, o))} \tag{28}$$

Die Silhouette $S(o)$ eines Objekts liefert also einen Wert zwischen -1 und 1. Wenn $S(o) \approx 1$ ist, kann das so interpretiert werden, dass $d(A, o)$ nahe bei 0 ist, und $d(B, o) > d(A, o)$, dass also das Objekt o dem Cluster zugeordnet wurde, dem es am nächsten liegt. Wenn $s(o) \approx -1$ ist, bedeutet das, dass $d(A, o) > d(B, o)$, was auf eine ungünstige Zuordnung von o schließen lässt. Somit ist die Silhouette also ein Maß für die Güte der Zuordnung eines Objektes zu einem bestimmten Cluster.

Definition 10 (Silhouetten-Koeffizient) *Sei C ein Clustering. Dann ist der **Silhouetten-Koeffizient** von C definiert als:*

$$s_C = \frac{1}{n} \sum_{i=1}^{n} S(o_i) \tag{29}$$

Letztlich liefert der Silhouetten-Koeffizient als Durchschnitt der Silhouetten aller Objekte also einen Anhaltspunkt für die Güte eines Clusterings unabhängig von der Anzahl der Cluster. Sofern sich die Anzahl der Cluster nicht direkt aus der Herkunft der Daten ergibt, lässt sich mithilfe eines solchen clusterzahl-unabhängigen Kriteriums auch feststellen, welche Anzahl von Clustern zu guten Ergebnissen führt. Bei der im nächsten Abschnitt behandelten *Spectral Clustering*-Methode PCCA+ ist ein solches Kriterium (die Schärfe) schon im Algorithmus enthalten. Andere Qualitätskriterien für Clusterings sind beispielsweise der *Davies-Bouldin Index* (Davies u. Bouldin (1979)) und der *Dunn-Index* (Dunn (1973)).

2.3 PCCA+

Dieser Abschnitt richtet sich, soweit nicht anders angemerkt, nach Röblitz u. Weber (2009) und Deuflhard u. Weber (2003). Zuerst sollen die der PCCA+ zugrunde liegenden Ideen und Konzepte beschrieben werden, im Anschluss wird die Struktur der algorithmischen Umsetzung erläutert.

2.3.1 Konzept der PCCA+

PCCA steht für **P**erron **C**luster **C**luster **A**nalysis und ist ein Clustering-Algorithmus, der 1998 von Deuflhard, Huisinga, Fischer und Schütte veröffentlicht wurde, siehe Deuflhard u. a. (1998). Die PCCA berechnet ein Clustering basierend auf der Vorzeichenstruktur von Eigenvektoren, was numerisch nicht robust ist. Daher entwickelten Deuflhard und Weber eine verbesserte Variante der PCCA, die Robuste Perron Cluster Analyse (PCCA+), erstmals veröffentlicht in Deuflhard u. Weber (2003).

Vom Aufbau her ist die PCCA+ dem oben beschriebenen *Normalized spectral clustering according to Shi and Malik* sehr ähnlich. Auch hier wird die Tatsache ausgenutzt, dass die Eigenvektoren von L_{rw} bzw von $I - L_{rw}$ die Informationen über die Clusterstruktur enthalten.

Der Algorithmus verwendet die Eigenvektoren zu einer Menge von k Eigenwerten nahe bei dem Eigenwert $\lambda = 1$, um k Cluster in einer zeilenstochastischen Matrix P zu identifizieren. Da der Eigenwert $\lambda = 1$ als Perron-Eigenwert bezeichnet wird, werden die zugehörigen Eigenvektoren häufig Perron-Eigenvektoren, und der von diesen aufgespannte Eigenraum Perron-Eigenraum genannt.

Um die Idee der PCCA+ zu verstehen, ist es sinnvoll, erst den bereits im vorherigen Abschnitt erwähnten Idealfall zu betrachten, bei dem die Cluster so gut von einander getrennt sind, dass der Ähnlichkeitsgraph tatsächlich in k Komponenten zerfällt. In diesem Fall hat P den k-fachen Perron-Eigenwert $\lambda_1 = \ldots = \lambda_k = 1$ und die Matrix P kann so permutiert werden, dass sie Blockstruktur hat, das heißt, dass auf der Diagonalen k Blockuntermatrizen liegen, und alle übrigen Einträge 0 sind. Die zugehörigen Eigenvektoren sind stückweise konstant auf den Blöcken von P.

Sei S_i die Indexmenge des i-ten Blocks. Dann wird der Eigenraum aufgespannt von den Indikatorvektoren $\chi_i^T = (0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0)$ für $i = 1, \ldots, k$, wobei $\chi_i(j) = 1 \Leftrightarrow j \in S_i$.

Angenommen die χ_i sind bekannt, dann kennt man also automatisch auch die Clusterzugehörigkeiten der Objekte. Daher werden die χ_i auch Zugehörigkeitsvektoren genannt. Sie lassen sich außerdem als charakteristische Funktionen der invarianten Teilmengen S_i interpretieren: Eine charakteristische Funktion ist so definiert, dass sie für eine bestimmte Teilmenge jeweils für Elemente dieser Teilmenge den Funktionswert 1 annimmt und sonst 0 ist.

Wenn nun aber, wie es in der Praxis üblicherweise der Fall ist, der Ähnlichkeitsgraph nicht in k Komponenten zerfällt, die Cluster also nicht so gut voneinander getrennt sind, kann ähnlich vorgegangen werden: Dann hat P eine Menge von k Eigenwerten, die nahe bei 1 liegen: $1 = \lambda_1 > \lambda_2 > \ldots > \lambda_k = 1 - \varepsilon$. Eben wurden die stückweise konstanten χ als charakteristische Funktionen interpretiert. In dem Fall, dass die Cluster nicht so gut getrennt sind, lässt sich eine solche Basis des Eigenraumes aber nicht mehr finden - die charakteristischen Funktionen mit den Werten 0 oder 1 sind also auf diesem Wege nicht mehr definierbar.

Der Ansatz, der die PCCA+ von der Vorgängermethode PCCA unterscheidet, ist nun dieser: Statt der charakteristischen Funktionen χ_i der invarianten Teilmengen werden fast charakteristische Funktionen $\tilde{\chi}_i$ der nun als 'fast invariante Teilmengen' bezeichneten S_i gesucht. Während χ Zugehörigkeitswerte eines harten Clusterings enthält, wird durch $\tilde{\chi}$ ein *fuzzy* Clustering definiert.

Zwei Bedingungen für $\tilde{\chi}$, die sich aus der Definition eines *fuzzy* Clusterings ergeben, sind die folgenden:

$$\tilde{\chi}_i(l) \geq 0 \qquad \text{(Positivität)} \tag{30}$$

$$\sum_{i=1}^{k} \tilde{\chi}_i(l) = 1 \qquad \text{(Zerlegung der Eins)} \tag{31}$$

Da die $\tilde{\chi}$ die Zugehörigkeiten zu den Clustern darstellen, ist klar, dass sie stets nicht-negativ sein müssen. In der Definition eines *fuzzy* Clusterings wurde zudem gefordert, dass die Summe der Zugehörigkeitswerte eines jeden Objektes 1 ist.

Es sei $\{X_1, \ldots, X_k\}$ eine Basis des Perron-Eigenraums und $X = [X_1, \ldots, X_k]$. Gesucht ist dann $\tilde{\chi}$ mit $\tilde{\chi} = XA$, beziehungsweise in Komponentenschreibweise $\tilde{\chi}_i = \sum_{j=1}^{k} \alpha_{ij} X_j$, wobei $A = (\alpha_{ij})_{i,j=1,\ldots,n}$ und $\tilde{\chi} = [\tilde{\chi}_1, \ldots, \tilde{\chi}_k]$.

Äquivalent lässt sich das Finden eines Clusterings aber auch beschreiben als Suche nach einer Transformationsmatrix A der Eigenraumbasis X, so dass $XA = \tilde{\chi}$ zeilenstochastisch ist. Die Einträge von A (oder $\tilde{\chi}$) sind also k^2 Unbekannte, für deren Wahl geeignete Kriterien festgelegt werden müssen.

Als mögliche Zielfunktionen stehen nun mehrere zur Wahl. Ursprünglich wurde die folgende Zielfunktion definiert und implementiert:

$$I_1(A) = \sum_{j=1}^{k} \max_{l=1,\ldots,n} \sum_{i=1}^{k} X(l,i) A(i,j) \tag{32}$$

Das Maximieren dieser Zielfunktion bedeutet, dass der maximale Zugehörigkeitswert eines jeden Clusters maximiert wird. Das Ziel hierbei ist, dass in jedem Cluster mindestens ein Objekt einen Zugehörigkeitswert ≈ 1 hat.

Eine andere Variante, um ein möglichst gutes Clustering zu finden, entstand aus der Idee der Metastabilität. Um die verwendete Zielfunktion zu motivieren, ist es hilfreich, deren Entstehung zu betrachten. Zunächst wurden in Deuflhard u. a. (1998) harte Clusterings betrachtet, und zwar vom Standpunkt der oben beschriebenen *Random Walk*-Interpretation. Für ein gegebenes hartes Clustering χ wurden nun die Wahrscheinlichkeiten, dass ein *Random Walk* innerhalb eines Clusters bleibt, aufsummiert und als *Metastabilität* bezeichnet:

Definition 11 (Metastabilität) *Es sei ein hartes Clustering χ gegeben, sowie eine Übergangsmatrix P und die stationäre Verteilung π.*
Weiterhin sei $D_c = diag(\chi^T \pi)$ und $D = diag(\pi)$. Dann ist die ***Metastabilität*** *des Clusterings definiert als $\mathrm{spur}(D_c^{-1}\chi^T DP\chi)$.*

Genauer gesagt enthält die $k \times k$-Matrix $D_c^{-1}\chi^T DP\chi$ die Übergangswahrscheinlichkeiten zwischen den Clustern, und daher werden mithilfe der Spur genau die Wahrscheinlichkeiten, innerhalb eines Clusters zu bleiben, summiert. Diese Matrix wird auch als vergröberte Übergangsmatrix bezeichnet. Weber (2006) übernahm diese Definition für *fuzzy* Clusterings.

Die Metastabilität wurde als zweite mögliche Zielfunktion implementiert:

$$I_2(A) = spur(D_c^{-1}\chi^T DP\chi) \tag{33}$$

Im Sinne der *Random Walk*-Interpretation ist die Maximierung dieses Wertes naheliegend: Wenn die Wahrscheinlichkeiten, dass ein *Random Walk* in einem Cluster bleibt, sehr hoch sind, bedeutet dies, dass der Ähnlichkeitsgraph 'fast' in k Komponenten zerfällt. Je höher die Metastabilität, desto 'besser' ist also das Clustering.

In Kube u. Weber (2007) wurde festgestellt, dass für diese Definition der vergröberten Übergangsmatrix Projektion und Propagation nicht vertauschbar sind. In diesem Zusammenhang meint Projektion den Übergang von der Übergangsmatrix $P \in \mathbb{R}^{n\times n}$ zur vergröberten Übergangsmatrix $P_c \in \mathbb{R}^{k\times k}$, also die Projektion der ursprünglichen n Objekte auf ihre k Cluster. Mit Propagation ist der iterative Übergang von einer Verteilung x_k zu x_{k+1} gemeint, der definiert ist durch $x_{k+1}^T = x_k^T P$. Die Propagation kann nicht nur auf einer Verteilung $x_k \in \mathbb{R}^n$ der ursprünglichen n Objekte stattfinden, sondern auch auf einer Verteilung $y_k \in \mathbb{R}^n$ auf den k Clustern, dann gilt $y_{k+1}^T = y_k^T P_c$ mit $y_0 = \chi^T x_0$. Wenn Propagation und Projektion vertauschbar wären, würde gelten $y_{k+1} = \chi^T x_{k+1}$. Kube u. Weber (2007) stellten allerdings fest, dass diese Gleichheit nicht gilt. Um diese Inkonsistenz aufzulösen, wurde eine neue vergröberte Übergangsmatrix $\tilde{P}_c = (\tilde{\chi}^T D\tilde{\chi})^{-1}\tilde{\chi}^T DP\tilde{\chi}$ definiert, durch deren Verwendung Projektion und Propagation vertauschbar sind. Entsprechend wurde auch die Metastabilität neu definiert als $spur(\tilde{P}_c)$. Leider zeigte sich, dass diese neue Metastabilität als Maß für die Güte eines Clusterings nicht geeignet ist, da $spur(\tilde{P}_c) = spur(\Lambda)$ gilt, wobei Λ Eigenwerte von P enthält und somit $spur(\tilde{P}_c)$ nicht vom Clustering abhängt.

Um also ein neues Maß für die Qualität eines Clusterings zu definieren, wird die Schärfe eines Clusterings eingeführt.

Definition 12 (Schärfe) *Die* ***Schärfe*** *eines fuzzy Clusterings $\tilde{\chi}$ in k Cluster ist definiert als $\frac{1}{k}\mathrm{spur}(D_c^{-1}\tilde{\chi}^T D\tilde{\chi})$.*

Motiviert wurde diese Definition durch die Überlegung, dass $P_c = \tilde{P}_c$ genau dann gilt, wenn $\tilde{\chi}$ *crisp* ist, also wenn $\tilde{\chi}$ aus den Indikatorvektoren besteht. $P_c = \tilde{P}_c$ ist aber äquivalent zu $D_c^{-1}\tilde{\chi}^T D\tilde{\chi} = Id$. Daraus ergibt sich, dass die Schärfe tatsächlich ein Maß für die Qualität eines Clusterings ist: Aus der Maximierung der Schärfe folgt, dass $spur(D_c^{-1}\tilde{\chi}^T D\tilde{\chi})$ möglichst nah an k und somit $D_c^{-1}\tilde{\chi}^T D\tilde{\chi}$ nah an der Identität ist, und dass daher die Zugehörigkeiten $\tilde{\chi}$ möglichst nahe an den Indikatorvektoren sind. (Der größtmögliche Wert der Schärfe ist somit 1.)

Es lässt sich mithilfe von Lemma 3.6. aus Weber (2006) zeigen, dass

$$spur(D_c^{-1}\tilde{\chi}^T D\tilde{\chi}) = \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{A(i,j)^2}{A(1,j)} \tag{34}$$

gilt. Die in der Implementierung der PCCA+ verwendete Zielfunktion ist daher

$$\tilde{I}_2(A) = \sum_{i=1}^{k}\sum_{j=1}^{k} \frac{A(i,j)^2}{A(1,j)} \tag{35}$$

Da im Rahmen dieser Arbeit auch eine Optimierungsmethode verwendet wurde, die die Jacobimatrix gebraucht, wurde diese Zielfunktion abgeleitet. Die erste Zeile und Spalte der Matrix A werden in der Implementierung der PCCA+ durch das im nächsten Abschnitt beschriebene m-File *fillA.m* überschrieben, daher musste nur nach $A(i,j)$ für $i,j \neq 1$ abgeleitet werden. Es sei l derjenige Index, für den $\sum\limits_{i=2}^{k} A(i,j)X(l,i)$ minimal ist. Dann gilt:

$$\frac{d\tilde{I}_2}{dA(i,j)} = \frac{2A(i,j)}{A(1,j)} + A(i,j)^2\frac{X(l,i)}{A(1,j)^2} \text{ für } i,j = 2,\dots,k \tag{36}$$

Die Maximierung der Funktion $\tilde{I}_2(A)$ bewirkt also, dass die aus der optimalen Matrix A berechneten $\tilde{\chi}$ so *crisp* wie möglich sind. Tatsächlich wird allerdings nicht diese Funktion maximiert, sondern $-\tilde{I}_2$ wird minimiert, da die verwendeten Optimierungsalgorithmen zur Minimierung von Funktionen konzipiert sind.

Um eine Matrix A zu finden, für die auch die obigen Bedingungen (30) und (31) an $\tilde{\chi}$ erfüllt sind, werden diese umformuliert als Bedingungen an A. Die dadurch definierte Menge von Matrizen sei als Menge F der zulässigen Matrizen bezeichnet. Die Bedingungen lauten dann:

$$A(1,j) = \min_{l=1,\dots,n} \sum_{i=2}^{k} A(i,j)X(l,i) \quad \text{für alle } j = 1,\dots,k,\ l = 1,\dots,n \tag{37}$$

$$A(i,1) = \delta_{i,1} - \sum_{j=2}^{k} A(i,j) \quad \text{für alle } i = 1,\dots,k \tag{38}$$

Hierbei entspricht die erste der Positivität und die zweite der Zerlegung der Eins. Bei der ersten Bedingung fällt auf, dass diese ursprünglich eine Ungleichheitsbedingung war. Der Übergang zu einer Gleichheitsbedingung lässt sich mit

Lemma 3.5 aus Weber (2006) begründen, welches im Wesentlichen besagt, dass auf den Ecken der konvexen Menge F immer für mindestens ein Objekt der Zugehörigkeitswert 0 ist.

In Lemma (3.7) aus Weber (2006) wird gezeigt, dass die Zielfunktionen I_1 und I_2 konvex sind, und dass es Ecken der zulässigen Menge F gibt, die die Funktionen maximieren, dass also das Maximum nicht irgendwo in F angenommen wird, sondern auf einer Ecke. Um die Verwendung der Zielfunktion $\tilde{I}_2$ auch theoretisch zu begründen, muss dieses Lemma auch für $\tilde{I}_2$ bewiesen werden.

Zunächst wird, dem Beweis in Weber (2006) folgend, gezeigt, dass die $\tilde{I}_2$ konvex ist.

Satz 7 *$\tilde{I}_2$ ist konvex auf F.*

Beweis: Die Hessematrix von $f(a, b) = a^2 b^{-1}$ hat für $b > 0$ nichtnegative Eigenwerte. Nach Satz (3.6) in Weber (2006) ist $b = A(1, j)$ positiv. Sei also $a = A(i, j)$, dann hat die Hessematrix von $\tilde{I}_2$ nichtnegative Eigenwerte, ist also positiv-semidefinit. Eine Funktion ist aber genau dann konvex, wenn ihre Hessematrix positiv semidefinit ist. Somit ist gezeigt, dass $\frac{A(i,j)^2}{A(1,j)}$ (als Funktion von A) konvex ist. Da die Summe konvexer Funktionen stets auch konvex ist, folgt die Konvexität von $\tilde{I}_2(A) = \sum\limits_{i=1}^{k} \sum\limits_{j=1}^{k} \frac{A(i,j)^2}{A(1,j)}$. ■

Satz 8 *Die Menge F der zulässigen Matrizen hat optimale Ecken, die $\tilde{I}_2$ maximieren.*

Beweis: (nach Weber (2006))
$\tilde{I}_2$ ist stetig und beschränkt. Daher hat $\tilde{I}_2$ ein Supremum. Das Supremum einer konvexen Funktion in einem geschlossenen konvexen Polytop wird auf einer Ecke des Polytops angenommen. ■

Aus diesen beiden Sätzen folgt, dass die Suche nach A beschränkt werden kann auf die Ecken von F.

2.3.2 Algorithmische Umsetzung

Zunächst soll der grobe Ablauf des Algorithmus beschrieben werden, bevor auf die einzelnen Schritte genauer eingegangen wird.

Der Input ist eine zeilenstochastische Matrix P, wie beispielsweise die Übergangsmatrix eines *Random Walks.* Von dieser Matrix P werden k Eigenpaare benötigt, die nun berechnet werden müssen. Die zu berechnenden Eigenpaare sind diejenigen, bei denen der Eigenwert nahe bei 1 liegt.

Hierauf folgend wird ein Subalgorithmus gestartet, der einen Simplex als Startwert für die im nächsten Schritt folgende Minimierung berechnet. Die Minimierung liefert eine Matrix A, aus der die Zugehörigkeiten χ gewonnen werden. Dann werden die berechneten Zugehörigkeiten zwischen 0 und 1 wieder umgewandelt in eine Zugehörigkeitsmatrix, deren Werte genau 0 oder 1 betragen, also in ein hartes Clustering, das jedes Objekt genau einem Cluster zuordnet. Der Ablauf der PCCA+ ist in Abbildung 4 schematisch dargestellt.

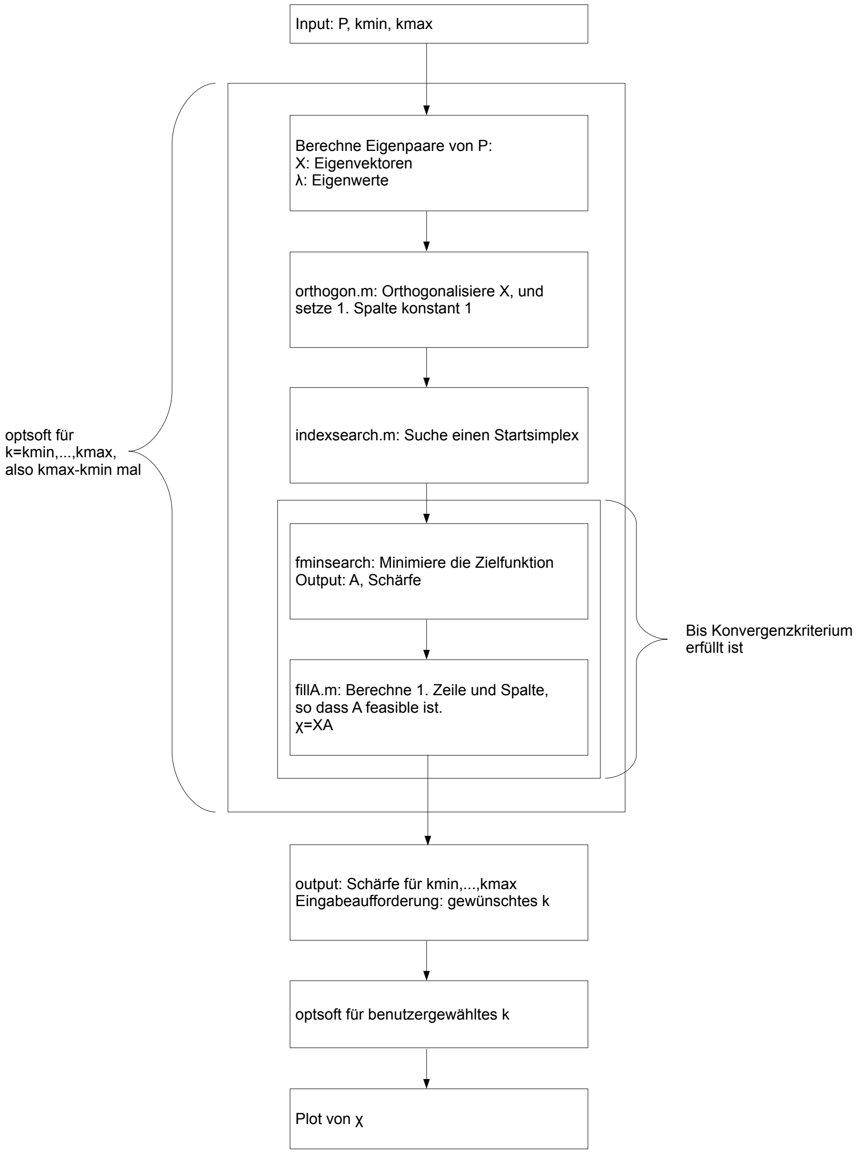

Abbildung 4: Flussdiagramm für die Implementierung der PCCA+

Im Allgemeinen liegen Datenpunkte nicht immer in der hier benötigten Form vor. Meist muss erst eine geeignete Matrix P berechnet werden. Eine der im vorigen Abschnitt beschriebenen Laplace-Matrizen kann verwendet werden, um eine solche Matrix aus Daten zu erzeugen, und zwar durch $P = I - L_{rw}$. Dass die so entstandene Matrix P den Eigenwert 1 hat, ergibt sich direkt aus Satz 3.

In der MATLAB-Implementierung der PCCA+ werden die Eigenpaare von P mit dem MATLAB-Befehl 'eigs' berechnet, welcher den Eigenwertlöser IRAM aufruft, der in einem folgenden Kapitel genauer beschrieben wird. Der erste Eigenvektor braucht dabei eigentlich nicht berechnet zu werden, da er stets der konstante Einsvektor ist, was - wie bereits erwähnt - aus Satz 3 folgt. Das m-File *orthogon.m* setzt daher die erste Spalte von X auf den konstanten Einsvektor und orthonormalisiert die Eigenvektoren.

Zur Berechnung eines Startwertes für die Optimierung der Zielfunktion wurde ein Subalgorithmus entwickelt. Da viele Minimierungsalgorithmen nur lokal gute Konvergenzeigenschaften haben, hängt der Erfolg der Minimierung stark davon ab, dass ein guter Startwert vorhanden ist. Um den verwendeten Subalgorithmus zu verstehen, ist es sinnvoll, eine geometrische Interpretation der PCCA+ zu erläutern.

Diese geometrische Interpretation des Clusterings erschließt sich, indem die Zeilen von $\tilde{\chi}$ als Punkte in einem Simplex σ_k interpretiert werden, dessen Ecken die Einheitsvektoren $e_1, \ldots, e_k$ sind. Dass die Zeilen von $\tilde{\chi}$ tatsächlich innerhalb des Simplex liegen, ist durch die Positivität und die Zerlegung der Eins begründet. Die lineare Abbildung A^{-1} bildet dann σ_k auf einen weiteren Simplex $\tilde{\sigma_k}$ ab: $\tilde{\sigma_k} = \sigma_k A^{-1}$. Die Ecken von $\tilde{\sigma_k}$ ergeben sich somit als $A^{-1}(i,:)$ für $i = 1, \ldots, k$. Die Punkte, die durch die Zeilen von X definiert sind, liegen dann innerhalb von $\tilde{\sigma_k}$.

In Weber (2006) wird nun gezeigt: Wenn in jedem Cluster mindestens ein Objekt den Zugehörigkeitswert 1 hat, dann ist die entsprechende Zeile von $\tilde{\chi}$ eine Ecke von σ_k, und daher die entsprechende Zeile von X eine Ecke von $\tilde{\sigma_k}$. Wenn der Wert 1 nicht genau angenommen wird, sondern der größte Zugehörigkeitswert nur nahe bei 1 ist, lassen sich die Ecken von $\tilde{\sigma_k}$ durch Zeilen von X approximieren. Diese Eigenschaft wird ausgenutzt, um einen günstigen Startwert für die Minimierung zu finden. Die Routine *indexsearch.m* sucht daher diejenigen Indizes, für die die Zeilen von X die Ecken von $\tilde{\sigma_k}$ bestmöglich approximieren. Ausgegeben werden also k Ecken $X(l_1), \ldots, X(l_k)$. Daraus wird dann der Startwert für A berechnet als $A = \begin{pmatrix} X_1(l_1) & \ldots & X_k(l_1) \\ \vdots & \ddots & \vdots \\ X_1(l_k) & \ldots & X_1(k_k) \end{pmatrix}^{-1}$, wobei die erste Zeile und Spalte jeweils gelöscht werden müssen. (Der Grund hierfür wird deutlich, wenn die Optimierung genauer beschrieben wird.)

Der Subalgorithmus zur Berechnung des Startwertes funktioniert wie folgt: Als Startwert wähle $\bar{X}' = (0, \ldots, 0)$

1. Suche $X(l_1)$, denjenigen Datenpunkt mit dem größten Abstand zu $\bar{X}'$, also mit maximalem $||X(l)||_2$. Setze $\mathcal{X}_1 = X(l_1)$.

2. Für $i = 2, \ldots, k$ suche denjenigen Index l_i, für den $||X(l_i) - \mathcal{X}_{i-1}||$ maximal ist.
   Die k Ecken von $\tilde{\sigma}_{k-1}$ seien $X(l_1), \ldots, X(l_k)$.

Der Aufwand für jeden der Schritte ist $O(n)$.

Minimiert wird die konvexe Funktion $-\tilde{I}_2(A)$ auf der linear beschränkten Menge F. Um die Nebenbedingungen 37 und 38 zu erfüllen, wurde das m-File *fillA.m* implementiert. Dieses überschreibt zu einer gegebenen Matrix A die erste Zeile und Spalte, so dass die beiden Bedingungen genau erfüllt werden. Es wird demnach die erste Zeile überschrieben mit $\min\limits_{l=1,\dots,n} \sum\limits_{i=2}^{k} A(i,j)X(l,i)$ für alle $j = 1,\dots,k,\ l = 1,\dots,n$ und die erste Spalte mit $-\sum\limits_{j=2}^{k} A(i,j)$ für alle $i = 1,\dots,k$. Dies wird in jedem Minimierungsschritt durchgeführt, so dass die Erfüllung der Nebenbedingungen erzwungen wird.

Numerisch umgesetzt wird die Optimierung in der bisherigen MATLAB-Implementierung mittels des MATLAB-Befehls *fminsearch*, der den Nelder-Mead-Simplex-Algorithmus verwendet.

Bei der bisher verwendeten Zielfunktion wird außerdem stets für A die Erhaltung der Norm der Startmatrix erzwungen. Dies war notwendig, um zu verhindern, dass bei voranschreitender Minimierung die Matrix degeneriert. (In der im Rahmen dieser Arbeit entwickelten Variante der PCCA+ mit NLSCON war dieser Schritt nicht mehr notwendig.)

Nachdem die Zielfunktion minimiert wurde, muss aus den berechneten $\tilde{\chi}$ wieder eine diskrete Zugehörigkeitsmatrix berechnet werden, damit jedes Objekt genau einem Cluster zugeordnet wird. Dies wird umgesetzt, indem für jedes Objekt o_i dasjenige Cluster C_j ausgewählt wird, für das der Zugehörigkeitswert am größten ist: $o_i \in C_j \Leftrightarrow \tilde{\chi}(i,j) = max_l \tilde{\chi}(i,l)$.

Außerdem bietet die MATLAB-Implementierung der PCCA+ die Möglichkeit, für die Anzahl der Cluster, sofern nicht sowieso bekannt, den hinsichtlich der Schärfe besten Wert ausgeben zu lassen. Das m-File *optimize_metastab.m* berechnet für alle Clusteranzahlen zwischen einem eingegebenen Minimum und Maximum die Schärfe, und lässt anschließend den Benutzer die Clusterzahl wählen.

Desweiteren ist nicht nur die hier beschriebene Zielfunktion $\tilde{I}_2$ implementiert, sondern es kann auch die oben definierte Zielfunktion I_1 ausgewählt werden. Für alle Versuche im Rahmen dieser Arbeit wurde aber $\tilde{I}_2$ gewählt.

3 Eigenwertlöser

In diesem Kapitel soll der Eigenwertlöser IRAM beschrieben werden, der verwendet wurde, um zu testen, ob PCCA+ mit weniger Eigenwertiterationen auskommt. Da viele Details und Beweise, die IRAM zugrunde liegen, hier den Rahmen sprengen würden, sei für eine sehr viel ausführlichere Darstellung auf Sorensen (1996) oder Lehoucq u. a. (1997) verwiesen, auf denen auch dieses Kapitel, soweit nicht anders angemerkt, basiert.

3.1 IRAM

Die **I**mplicitly **R**estarted **A**rnoldi **M**ethod (IRAM) ist eine Methode, um numerisch Eigenwerte und Eigenvektoren einer Matrix zu berechnen. IRAM wurde entwickelt von R. B. Lehoucq und D. C. Sorensen und erstmals 1996 veröffentlicht in Lehoucq u. Sorensen (1996). Der Algorithmus basiert auf der 1951 von W. E. Arnoldi entwickelten Arnoldi-Methode, die insbesondere für dünnbesetzte und sehr große Matrizen gut geeignet ist, und unterscheidet sich von dieser vor allem durch zwischengeschaltete Neustarts der Iterationen, was numerisch diverse Vorteile gegenüber der herkömmlichen Methode hat.

IRAM ist eine sehr verbreitete Methode zur Eigenwertberechnung, und wird beispielsweise in dem Fortran-Softwarepaket ARPACK verwendet, das in MATLAB laut Wright u. Trefethen (2001) seit Version 6 direkt verwendet wird. Wenn man in MATLAB mit dem Befehl 'eigs' ein Eigenwertproblem lösen lässt, wird ARPACK und somit IRAM aufgerufen.

Sorensen (1996) beschreibt IRAM als im Wesentlichen auf zwei Methoden aufbauend: Dem *Implicitly Shifted QR Algorithmus* und der *k-step Arnoldi Faktorisierung*. IRAM lässt sich demnach als verkürzte Form des *Implicitly Shifted QR Algorithmus* beschreiben. Die sonst bei Arnoldi- und Lanczos-Prozessen auftretenden numerischen Probleme können von IRAM vermieden werden.

IRAM berechnet eine vom Benutzer vorgegebene Anzahl von k Eigenpaaren, wobei die Eigenwerte nach bestimmten Kriterien gewählt werden können, beispielsweise größter Realteil oder größter Betrag.

In diesem Abschnitt werden zunächst einige wichtige Begriffe definiert. Dann wird die Arnoldi-Methode vorgestellt, deren Weiterentwicklung IRAM ist. Danach werden die beiden von IRAM verwendeten Konzepte, der *Implicitly Shifted QR* Algorithmus und die *k-step Arnoldi Faktorisierung*, beschrieben, und im Anschluss der IRAM-Algorithmus vorgestellt.

3.1.1 Grundbegriffe und Definitionen

Soweit nicht anderes gekennzeichnet, richten sich die grundlegenden Definitionen in diesem Abschnitt nach Golub u. Van Loan (1996).

Definition 13 (Eigenwerte und -vektoren) *Als* ***Eigenwert*** $\lambda \in \mathbb{C}$ *zum* ***Eigenvektor*** $x \in \mathbb{C}^n$ *einer quadratischen Matrix* $A \in \mathbb{C}^{n \times n}$ *werden all jene* λ *und* $x \neq 0$ *bezeichnet, für die gilt:*

$$Ax = \lambda x \tag{39}$$

Eigenvektoren sind also Vektoren, die von A auf ein Vielfaches von sich selbst abgebildet werden, deren Richtung sich folglich durch die Anwendung der linearen Abbildung A nicht ändert.

Die Menge aller Eigenwerte von A nennt man das Spektrum $\sigma(A)$ von A. Zu jedem Eigenwert gibt es nicht nur einen Eigenvektor, sondern beliebig viele, da aus $Ax = \lambda x$ sofort $A(\mu x) = \lambda(\mu x)$ folgt, und somit alle skalaren Vielfachen eines Eigenvektors auch Eigenvektor zum gleichen Eigenwert sind.

Desweiteren können auch paarweise linear unabhängige Vektoren Eigenvektoren zum gleichen Eigenwert sein. Dann wird dieser Eigenwert als entartet bezeichnet. Der von den Eigenvektoren eines Eigenwerts aufgespannte Raum heißt Eigenraum, und seine Dimension nennt man die geometrische Vielfachheit des Eigenwerts.

Definition 14 (Ähnlichkeitstransformation) *Es seien A und B quadratische Matrizen. Wenn eine reguläre Matrix X existiert, mit $B = X^{-1}AX$, so werden A und B ähnlich genannt, und X wird als **Ähnlichkeitstransformation** bezeichnet.*

Satz 9 *Ähnliche Matrizen haben die gleichen Eigenwerte.*

Beweis: Es gelte $Ax = \lambda x$, es ist also λ Eigenwert von A zum Eigenvektor x. Sei B eine zu A ähnliche Matrix mit $A = XBX^{-1}$.

$$\begin{aligned} Ax &= \lambda x && (40)\\ \Rightarrow XBX^{-1}x &= \lambda x && (41)\\ \Rightarrow BX^{-1}x &= X^{-1}\lambda x = \lambda X^{-1}x && (42) \end{aligned}$$

Somit ist dann λ auch Eigenwert von B zum Eigenvektor $X^{-1}x$. ■

Definition 15 (invarianter Unterraum) *Ein Unterraum $S \subset \mathbb{C}^n$ wird als **invariant** unter A (oder A-invariant) bezeichnet, wenn gilt: $x \in S \Rightarrow Ax \in S$.*

Die Eigenräume einer Matrix A sind stets A-invariant. Denn jeder Vektor im Eigenraum ist selbst Eigenvektor von A. Sei also v aus dem Eigenraum des Eigenwerts λ von A. Dann gilt $Av = \lambda v$, und da auch λv im Eigenraum liegt, ist der Eigenraum A-invariant.

Definition 16 (unitäre Matrix) *Eine Matrix $A = (a_{ij})_{i,j=1,\dots,n}$ heißt **unitär**, wenn gilt: $A^H A = I_n$, wobei $A^H = \bar{a}_{ji}$ die zu A adjungierte Matrix ist.*
Unitär ähnlich nennt man zwei Matrizen, die ähnlich sind, und deren Transformationsmatrix unitär ist.

Definition 17 (Hessenbergmatrix) *Eine Matrix H heißt **obere Hessenbergmatrix** genau dann, wenn $H_{ij} = 0$ für alle $i > j + 1$ gilt.*
*Eine Matrix H heißt **untere Hessenbergmatrix** genau dann, wenn H^T obere Hessenbergmatrix ist.*

Wenn im Folgenden von einer Hessenbergmatrix die Rede ist, ist stets eine obere Hessenbergmatrix gemeint. Der praktische Nutzen von Hessenbergmatrizen besteht vor allem darin, dass ihre Eigenwerte ausgesprochen effizient berechnet werden können. Eine häufig verwendete Strategie von Eigenwertlösern ist daher, die ursprüngliche Matrix mittels Ähnlichkeitstransformationen auf Hessenbergform zu bringen und dann die Eigenwerte der Hessenbergmatrix zu berechnen. Dies kann beispielsweise mit dem QR-Algorithmus geschehen. Eine Matrix lässt sich mittels Ähnlichkeitstransformationen auf Hessenbergform bringen, wenn ihre Eigenwerte alle einfache Eigenwerte sind.

Definition 18 (Schurzerlegung) *Sei $A \in \mathbb{C}^{n \times n}$. Dann existiert eine unitäre Matrix $Q \in \mathbb{C}^{n \times n}$ so dass gilt:*

$$Q^H A Q = T = D + N \tag{43}$$

T ist eine obere Dreiecksmatrix, deren Diagonalelemente $\lambda_1, \dots, \lambda_n$ die Eigenwerte von A sind, und $D = diag(\lambda_1, \dots, \lambda_n)$ ist eine Diagonalmatrix. N ist eine strikte obere Dreiecksmatrix, das bedeutet $N(i,j) = 0$ für alle $i \geq j$.

Die Schurzerlegung ist nicht eindeutig. Zu jeder Reihenfolge der Eigenwerte von A gibt es eine Zerlegung, so dass die Eigenwerte in genau dieser Reihenfolge auf der Diagonalen von T stehen.

Als partielle Schurzerlegung bezeichnet man eine Zerlegung, bei der nur die k ersten Spalten betrachtet werden. Es sei $Q_k \in \mathbb{C}^{n \times k}$ die Matrix, die die k ersten Spalten von Q enthält, und $T_k \in \mathbb{C}^{k \times k}$ die prinzipale Submatrix von T, also jene Matrix, die man erhält, wenn man bei T die $n - k$ letzten Spalten und Zeilen löscht. Dann ist die partielle Schurzerlegung für ein festes k: $AQ_k = Q_k T_k$. Die Spalten von Q_k werden Schurvektoren genannt.

Wenn man das Ziel hat, beispielsweise die k betragsmäßig größten Eigenwerte von A zu berechnen, gibt es eine zugehörige partielle Schurzerlegung, bei der eben diese Eigenwerte auf der Diagonalen von T_k stehen. Genau das tut der weiter unten beschriebene IRAM-Algorithmus: Er berechnet eine partielle Schurzerlegung, die zu bestimmten Eigenwerten gehört.

Definition 19 (Krylovunterraum) *Der k-dimensionale* ***Krylovunterraum*** *zu einer Matrix A und einem Vektor v ist definiert als:*

$$K_k(A, v) = span(v, Av, A^2v, \dots, A^{k-1}v) \tag{44}$$

Definition 20 (Ritz-Paare) *Ein Vektor x und ein Skalar θ werden nach Lehoucq u. a. (1997)* ***Ritz-Paar*** *(bzw. x Ritz-Vektor und θ Ritz-Wert) genannt, wenn sie die folgende Galerkin-Bedingung erfüllen:*

$$\langle w, Ax - x\theta \rangle = 0 \qquad \forall w \in K_k(A, v) \tag{45}$$

Ein für die in diesem Abschnitt vorgestellten Algorithmen wesentliches Konzept ist das Shifting. Shiftingstrategien werden dazu genutzt, die Konvergenz von Krylovraumverfahren zu beschleunigen. Ihre Funktionsweise beruht auf folgender Feststellung: $A - \mu I$ hat die gleichen Eigenvektoren wie A, und λ ist Eigenwert von A genau dann, wenn $\lambda - \mu$ Eigenwert von $A - \mu I$ ist. Wenn die shifts (in diesem Fall μ) günstig gewählt werden, kann die Konvergenz damit stark beschleunigt werden.

3.1.2 k-step Arnoldi Faktorisierung und Implicitly Shifted QR

Der **Implicitly Shifted QR Algorithmus** produziert eine Folge von unitären Ähnlichkeitstransformationen Q_j, die die Matrix A iterativ auf obere Dreiecksgestalt bringen. Die Eigenwerte einer Dreiecksmatrix lassen sich einfach auf der Diagonalen ablesen.

Als Input werden benötigt: Die $n \times n$-Matrix A, deren Eigenpaare berechnet werden sollen, eine unitäre Transformationsmatrix V_1, sowie eine obere Hessenbergmatrix H_1 mit: $AV_1 = V_1 H_1$. Dann funktioniert der Algorithmus wie folgt: Für $j = 1, 2, \ldots$ bis zum Erreichen eines Konvergenzkriteriums:

1. Wähle Shift $\mu = \mu_j$.
2. Berechne die QR-Zerlegung von $H_j - \mu I$, d.h. finde eine unitäre Matrix Q_j und eine obere Dreicksmatrix R_j mit $H_j - \mu I = Q_j R_j$.
3. $H_{j+1} := Q_j^H H_j Q_j$
$V_{j+1} := V_j Q_j$

Aus der Iterationsvorschrift ergibt sich, dass H_j jeweils unitär ähnlich zu A ist: Anfangs gilt $A = V_1 H_1 V_1^H$, da V unitär ist. Dann wird $H_2 = Q_1^H H_1 Q_1$ und $V_2 = V_1 Q_1$ gesetzt. Wenn man das in die erste Gleichung einsetzt, ergibt sich $A = V_2 Q_1^H Q_1 H_2 Q_1^H Q_1 V_2^H = V_2 H_2 V_2^H$, weil die Q_i unitär gewählt werden. Dies lässt sich induktiv fortsetzen. Da auch die V_i als Produkt zweier unitärer Matrizen unitär sind und zudem regulär, ist also jedes H_i unitär ähnlich zu A. Konvergenz bedeutet in diesem Fall, dass die Subdiagonalelemente von H_j mit der Hessenbergmatrix H_1 beginnend gegen Null konvergieren. Wenn die Subdiagonale konstant Null ist, ist H_j eine obere Dreiecksmatrix, und daher $A = V_j H_j V_j^H$ eine Schurzerlegung. Wie oben beschrieben, können dann die Eigenwerte von A einfach aus der Diagonalen von H_j abgelesen werden.
Die Schurvektoren konvergieren mit verschiedenen Raten. Eine effiziente und stabile Implementierung ist in Golub u. Van Loan (1996) zu finden.

Vorteile dieser Methode sind die schnelle Konvergenz und der effiziente Speicherverbauch. Nachteile sind, dass die Methode für große Matrizen nicht geeignet und nur schwer zu parallelisieren ist.

Bei sehr großen Problemen ist es häufig der Fall, dass die Matrix dünnbesetzt ist oder so strukturiert, dass der Aufwand für die Berechnung eines Matrix-Vektor-Produkts Av eher proportional zu n als zu n^2 ist. Durch wiederholtes Anwenden von Ähnlichkeitstransformationen wird diese Struktur schnell zerstört, so dass der Aufwand für das Matrix-Vektor-Produkt dann n^2 ist. Daher ist ein naheliegender Ansatz, eine Methode zu suchen, die Matrix-Vektor-Produkte nur direkt mit A durchführt und nicht mit zu A ähnlichen Matrizen. Eine Methode, die diese spezielle Struktur der Matrizen erhält, ist die *Power Method*. Nach Golub u. Van Loan (1996) ist der Algorithmus wie folgt:
Sei $A \in \mathbb{C}^{n \times n}$ diagonalisierbar.
Input: Startvektor $v_0 \in \mathbb{C}^n$.
Für $k = 1, 2, \ldots$:

1. $z_k = Av_{k-1}$
2. $v_k = \frac{z_k}{||z_k||_2}$
3. $\nu_k = [v_k]^H A v_k$

Wenn man die die Schritte 1 und 2 zusammenfasst, ergibt sich $v_k = \frac{Av_{k-1}}{||Av_{k-1}||_2}$. Es wird also in jedem Iterationsschritt eigentlich nur die aktuelle Eigenvektornäherung (v_{k-1}) mit der ursprünglichen Matrix A multipliziert, und das Ergebnis normiert. Betrachte die ersten Iterierten:

$$v_1 = \frac{Av_0}{||Av_0||_2} \tag{46}$$

$$v_2 = \frac{Av_1}{||Av_1||_2} = \frac{A^2 v_0}{||Av_0||_2||Av_1||_2} \tag{47}$$

$$v_3 = \frac{Av_2}{||Av_2||_2} = \frac{A^3 v_0}{||Av_0||_2||Av_1||_2||Av_2||_2} \tag{48}$$

$$\vdots \tag{49}$$

Somit spannen die berechneten Vektoren zusammen mit v_0 den Krylovraum $K_k(A, v_0) = span(v_0, Av_0, A^2v_0, \dots A^{k-1}v_0)$ auf.

Sei λ_1 der betragsmäßig größte Eigenwert von A, auch als dominanter Eigenwert bezeichnet. Dann konvergiert die *Power Method* unter der Bedingung, dass λ_1 dominanter und einfacher Eigenwert ist, und dass v_0 eine Komponente in die Richtung des zugehörigen dominanten Eigenvektors x_1 hat, gegen $\nu_k = \lambda_1$ und $v_k = x_1$. Die zweite Bedingung wird in der Praxis üblicherweise erfüllt, da sich durch Rundung in fast jedem Fall sowieso ergibt, dass der x_1-Anteil in v_0 nicht Null ist.

Die Konvergenzrate ist $|\lambda_2|/|\lambda_1|$, wobei λ_2 der betragsmäßig zweitgrößte Eigenwert von A ist. Daher ist die *Power Method* gut geeignet, wenn zu erwarten ist, dass die Lücke zwischen $|\lambda_2|$ und $|\lambda_1|$ groß ist.

Die *Power Method* hat jedoch die Nachteile, wie beschrieben nur den dominanten Eigenwert zu berechnen und in manchen Fällen langsam oder auch gar nicht zu konvergieren. Dennoch können aus der in der *Power Method* berechneten Folge von Vektoren Informationen extrahiert werden, um schnelle Algorithmen unter anderem zur Eigenwertberechnung zu entwickeln. Diese Klasse von Algorithmen wird Krylovraumverfahren genannt. Bekannte Beispiele sind der CG-Algorithmus, das Lanczos-Verfahren oder BiCGSTAB.

Ihre Funktionsweise beruht auf der Beobachtung, dass die von der *Power Method* berechneten Vektoren nicht nur Informationen über die Eigenvektoren zum dominanten Eigenwert enthalten, sondern auch zu anderen Eigenvektoren. So können beispielsweise Linearkombinationen dieser Vektoren konstruiert werden, um die Konvergenz zu einem anderen Eigenvektor zu erzwingen.

Auch das Arnoldi-Verfahren ist ein Krylovraumverfahren. Dieser Teilabschnitt richtet sich nach Saad (2003).

Laut Saad (2003) wurde das **Arnoldi-Verfahren** 1951 von Arnoldi als Methode entwickelt, eine dichte Matrix mit einer unitären Transformation auf Hessenbergform zu bringen. Später entdeckte man, dass die bereits 1951 vorgeschlagene Strategie, aus den Eigenwerten der Hessenbergmatrix die Eigenwerte der ursprünglichen Matrix zu iterieren, zu einer effizienten Methode der Eigenwertberechnung für große dünnbesetzte Matrizen führt.

Der Arnoldi-Algorithmus berechnet eine Orthonormalbasis des Krylovunterraums. Im Gegensatz zur *Power Method* werden die Vektoren $A^i v$ aber nicht direkt berechnet, da dies zu den gleichen numerischen Schwierigkeiten führen würde, wie sie auch bei der *Power Method* auftreten.

In Saad (2003) sind zwei praktische Implementierungen des Verfahrens zu finden. Hier soll nur eine exakt arithmetische Variante des Arnoldi-Verfahrens

vorgestellt werden:
Wähle einen Vektor v_1 mit $||v_1||_2 = 1$.
Für $j = 1, \dots, m$:

1. Berechne $h_{ij} = \langle Av_1, v_i \rangle$ für $i = 1, \dots, j$
2. Berechne $w_j = Av_j - \sum\limits_{i=1}^{j} h_{ij} v_i$
3. $h_{j+1,j} = ||w_j||_2$
4. Wenn $h_{j+1,j} = 0$ ist, dann stoppe
5. $v_{j+1} = w_j / h_{j+1,j}$

In Schritt 2 wird der Arnoldivektor aus der vorigen Iteration (v_j) mit A multipliziert, dann werden die Richtungskomponenten der vorigen v_i abgezogen, und somit der neue Arnoldivektor orthogonalisiert. Sofern das Ergebnis ungleich Null ist, wird es danach normiert. Diese Orthonormalisierung entspricht dem gebräuchlichen Gram-Schmidt-Verfahren.

Wenn der Algorithmus nicht vor der m-ten Iteration in Schritt 4 stoppt, dann bilden die Vektoren $v_1, \dots, v_m$ eine Orthonormalbasis des Krylovraums $K_m(A, v_1)$.

V_m sei die Matrix, die als Spalten die v_i enthält. Wenn die im letzten Schritt berechneten h_{ij} als die nicht-Null-Einträge einer Hessenbergmatrix aufgefasst werden, und von dieser Matrix die letzte Zeile gelöscht wird, ergibt sich eine Matrix, die hier H_m heißen soll. Es gilt dann $AV_m = V_m H_m + w_m e_m^T$. Diese Zerlegung von A wird *m-step Arnoldi Faktorisierung* genannt und wird weiter unten genauer beschrieben.

Wenn das Verfahren in Schritt 4 bei der j-ten Iteration stoppt, dann ist $w_j = 0$ und somit gilt $AV_j = V_j H_j$. Dann sind die Eigenwerte von H_j auch Eigenwerte von A. Wenn der Algorithmus so abbricht, gilt außerdem, dass jede Methode, die auf der Projektion auf den Krylovunterraum $K_j(v_1, A)$ basiert, exakt ist. Daher wird dieser Fall als 'lucky breakdown' bezeichnet. Wenn das Verfahren nicht in Schritt 4 stoppt, dann konvergieren einige der Eigenwerte von H_m gegen Eigenwerte von A. Laut Wikipedia (2012) sind das typischerweise die extremen Eigenwerte von A.

Ein nah verwandtes Verfahren ist der Lanczos-Algorithmus. Der wesentliche Unterschied zur Arnoldi-Methode ist, dass die Matrizen H_k im Lanczos-Algorithmus tridiagonal sind, und dass nur drei Vektoren gespeichert werden müssen. Für den genauen Algorithmus und seine Eigenschaften siehe Saad (2003). Wegen der Ähnlichkeit der beiden Verfahren, werden sie oft als Arnoldi-/Lanczos-Verfahren zusammengefasst benannt.

Das Arnoldi-Verfahren hat einige gravierende numerische Nachteile und ist außerdem sehr speicheraufwändig. In manchen Fällen sind sehr viele Iterationen nötig, wodurch es numerisch schwierig wird, die Orthogonalität der v_j beizubehalten. Der Verlust der Orthogonalität führt zu weiteren numerischen Problemen, deren Behebung oder Vermeidung Ziel diverser Varianten des Verfahrens (beziehungsweise des Lanczos-Verfahrens) ist. Wenn die v_j keinen invarianten Unterraum von A aufspannen, so ist (exakt arithmetisch) w_i niemals Null.

Das zweite für IRAM wesentliche Konzept ist die oben bereits erwähnte **k-step Arnoldi Faktorisierung**. Diese Faktorisierung ist eine Zerlegung der Matrix A wie folgt:
$AV_k = V_k H_k + f_k e_k^T$ wobei $V_k^H f_k = 0$, H_k eine obere Hessenbergmatrix mit nicht-negativen Subdiagonalelementen ist, und V_k orthonormale Spalten hat. Wenn A hermitesch ist, ist H_k eine reelle, symmetrische Tridiagonalmatrix, und die Faktorisierung wird Lanczos-Faktorisierung genannt. Die Spalten von V_k heißen Arnoldi- oder Lanczos-Vektoren und sind eine Orthonormalbasis des Krylovunterraums $K_k(A, v_1)$. H_k ist die orthogonale Projektion von A auf eben diesen Unterraum.

Eine andere Schreibweise für die *k-step Arnoldi Faktorisierung* ist

$$AV_k = (V_k, v_{k+1}) \begin{pmatrix} H_k \\ \beta_k e_k^T \end{pmatrix} \tag{50}$$

mit $\beta_k = ||f_k||$ und $v_{k+1} = \frac{1}{\beta_k} f_k$. Das zuvor beschriebene Arnoldi-Verfahren resultiert in einer Arnoldi-Faktorisierung, womit auch die naheliegende Frage nach der numerischen Berechnung einer solchen Faktorisierung beantwortet ist.

Die Faktorisierung kann beispielsweise benutzt werden, um Näherungslösungen eines linearen Gleichungssystems $Ax = b$ mit $b = v_1\beta_0$ zu berechnen. Das GMRES-Verfahren (Generalized minimal residual method) zur Lösung großer dünnbesetzter linearer Gleichungssysteme basiert auf der Arnoldi-Faktorisierung.

3.1.3 IRAM: Der Algorithmus und seine Eigenschaften

Implizites Neustarten ist eine Methode, um die relevanten Informationen aus großen Krylovunterräumen zu extrahieren und gleichzeitig die numerischen Schwierigkeiten und Speicherplatzprobleme zu umgehen, die beim nicht-neugestarteten Ansatz auftreten würden. Umgesetzt wird dies, indem die interessanten Informationen in einem k-dimensionalen Unterraum komprimiert werden. Hierfür wird die oben vorgestellte *Implicity Shifted QR Methode* verwendet.

Das grobe Schema von IRAM ist wie folgt:
Input: A, V, H, f mit $AV_m = V_m H_m + f_m e_m^T$, eine *m-step Arnoldi Faktorisierung* von A, sowie m.
Bis zur Erfüllung eines Konvergenzkriteriums führe die folgenden Schritte aus:

1. Berechne die Eigenwerte von H_m: $\lambda_1, \ldots, \lambda_m$.
   Sortiere diese in gewünschte und unerwünschte Eigenwerte, je nach Benutzerkriterium.

2. Berechne $m-k = p$ Schritte der *Implicitly Shifted QR Methode* angewandt auf Q_m, wobei als shifts $(\mu_1, \ldots, \mu_k)$ beispielsweise die unerwünschten Eigenwerte verwendet werden:

   - Für $j = 1, \ldots, p$:
     Berechne die QR-Zerlegung von $H_m - \mu_j I$, also Q_j und R_j.
     $H_m := Q_j^H H_m Q_j$
     $V_m := V_m Q_j$

   Ergebnis: $H_m Q_m = Q_m H_m^+$

3. $f_k := V_{k+1}\beta_k + f_m\sigma_k$
   $V_k := V_m(1:n, 1:k)$
   $H_k := H_m(1:k, 1:k)$

4. Beginnend mit der *k-step Arnoldi-Faktorisierung* $AV_k = V_kH_k + f_ke_k^T$, wende p weitere Arnoldi-Schritte an, um eine m-step Faktorisierung zu erhalten: $AV_m = V_mH_m + f_me_m^T$.

Der Ablauf von IRAM lässt sich also wie folgt zusammenfassen: Nachdem je nach Benutzerkriterium die Eigenwerte in 'erwünscht' und 'unerwünscht' eingeteilt wurden, wird die *Implicitly Shifted QR Methode* angewendet, jedoch nicht bis zur Konvergenz, sondern nur $p = m - k$ Schritte. Die so erzeugten Matrizen der Größe m werden auf Matrizen der Größe k verkleinert (eingeschränkt) und danach mittels des Arnoldi-Verfahrens wieder auf Größe m erweitert.

IRAM berechnet eine partielle Schurzerlegung, so dass genau die gewünschten Eigenwerte ausgegeben werden. Die anfangs gegebene Arnoldi Faktorisierung der Länge $m = k + p$ wird in eine Faktorisierung der Länge k komprimiert, die die relevanten Informationen enthält. Um genau die relevanten Informationen zu erhalten und den Einfluss der unerwünschten Eigenwerte zu verringern, werden p shifts angewandt. In obigem Algorithmus ist das Schritt 2. Das Ergebnis des Shiftings ist

$$AV_m = V_m^+H_m^+ + f_me_m^TQ \tag{51}$$

mit $V_m^+ = V_mQ$, $H_m^+ = Q^HH_mQ$ und $Q = Q_1Q_2\dots Q_p$. Die Q_j sind die im *Implicitly Shifted QR* Algorithmus benutzten orthogonalen Matrizen zum Shift μ_j. Die Q_j sind Hessenbergmatrizen, daher sind die ersten k Einträge von e_m^T 0. Daraus folgt, dass die k führenden Spalten in Gleichung 51 auch in Arnoldi-Relation sind. Die aktualisierte Arnoldi Faktorisierung erhält man durch das Reduzieren auf diese ersten k Spalten: $AV_k^+ = V_k^+H_k^+ + f_k^+e_k^T$, wobei das aktualisierte Residuum $f_k^+ = V_m^+e_{k+1}\hat{\beta}_k + f_m\sigma$ ist.

Davon ausgehend werden nun p zusätzliche Schritte des Arnoldiprozesses durchgeführt, so dass am Ende des Iterationsschritts eine *m-step Arnoldi Faktorisierung* steht, da ja das Arnoldi-Verfahren eine Arnoldi-Faktorisierung erzeugt.

Die standardmäßig in ARPACK verwendete Shifting Strategie ist die 'Exact Shift Strategy': Berechne $\sigma(H_k)$ und sortiere in eine erwünschte Menge von Eigenwerten Ω_w und eine unerwünschte Menge Ω_u. Die k Ritzwerte in Ω_w werden als Approximationen der gewünschten Eigenwerte betrachtet, und die p Ritzwerte in Ω_u werden als shifts verwendet.

In exakter Arithmetik ist nach dem QR-Schritt $\sigma(H_k) = \Omega_w$, und der aktualisierte Startvektor v_1 ist eine Linearkombination der zugehörigen Ritzvektoren. IRAM mit der 'Exact Shift Strategy' berechnet also eine spezifische Wahl von Koeffizienten γ_j für einen neuen Startvektor als Linearkombination der Ritzvektoren als aktuell beste Näherungen der gesuchten Eigenvektoren.

Wenn $m = n$ ist, so ist $f = 0$ und dann unterscheidet sich obiger Algorithmus nicht von dem *Implicitly Shifted QR* Algorithmus. Aber auch wenn $m < n$ ist, so sind die k ersten Spalten von V und $H(1:k, 1:k)$ hier die gleichen, wie beim *Implicitly Shifted QR* Algorithmus mit den gleichen Shifts. Dies ist die Begründung dafür, dass IRAM, wie oben bereits erwähnt, als Abkürzung des *Implicitly Shifted QR* Algorithmus gesehen werden kann. Der wesentliche Unterschied ist die Wahl der Shifts: Der *Implicitly Shifted QR* Algorithmus wählt diese so, dass die Subdiagonalelemente von H von unten nach oben gegen Null gehen, und bei IRAM geschieht dies von oben nach unten.

Der Speicheraufwand für IRAM ist $2nk + O(k^2)$. Die berechneten Schur-Basisvektoren sind bis auf Maschinengenauigkeit orthogonal. Wie bereits erwähnt, ist IRAM in MATLAB bereits mit dem Befehl *eigs* bzw. dem Softwarepaket ARPACK vorhanden. Hier kann der Code allerdings nicht direkt eingesehen werden. Eine weitere MATLAB-Implementierung von D. C. Sorensen ist online verfügbar, siehe Sorensen (2011).

3.1.4 Wahl von m

Die Tatsache, dass als Input der Wert m benötigt wird, wurde bisher nicht weiter diskutiert. m ist die Dimension des Suchraums.

Bei dem MATLAB-Befehl *eigs* findet die Wahl von m automatisiert statt und ist nicht vom Benutzer beeinflussbar. Wenn aber eine andere Version von IRAM verwendet wird, muss m festgelegt werden. Nun stellt sich natürlich die Frage, nach welchen Kriterien m gewählt werden sollte. Generell muss gelten $k < m << n$. Im MATLAB-Code aus Sorensen (2011) steht als Kommentar, dass empfohlen wird, ein $m > 2k$ zu wählen.

Wenn m sehr klein gewählt wird, ist zu erwarten, dass ein einzelner Iterationsschritt verhältnismäßig schnell geht (weil ja der Suchraum klein ist), aber dafür viele Schritte benötigt werden. Umgekehrt ist bei der Wahl eines relativ großen Wertes zu erwarten, dass wegen der großen Suchraumdimension jede Iteration lange dauert, aber dafür nur wenige Iterationen gemacht werden. Insgesamt kann also damit gerechnet werden, dass es im Sinne einer optimalen Laufzeit ein m gibt, für das das Produkt aus Laufzeit pro Iteration und Anzahl der Iterationen minimal ist.

Um diese Vermutungen an einem numerischen Beispiel zu testen, wurde mit dem MATLAB-Befehl *rand* eine Matrix der Größe 100 erzeugt, und für alle m zwischen $k + 1 = 26$ und $n - 1 = 99$ IRAM gestartet. Abbildung 5 zeigt, dass die Anzahl der Iterationen tatsächlich mit steigendem m geringer wird, und die Dauer der Iterationen sich genau umgekehrt verhält. Dass die Wahl eines m größer als $2k = 50$ zumindest in diesem Beispiel im Sinne einer geringen Laufzeit sinnvoll ist, zeigt Abbildung 6.

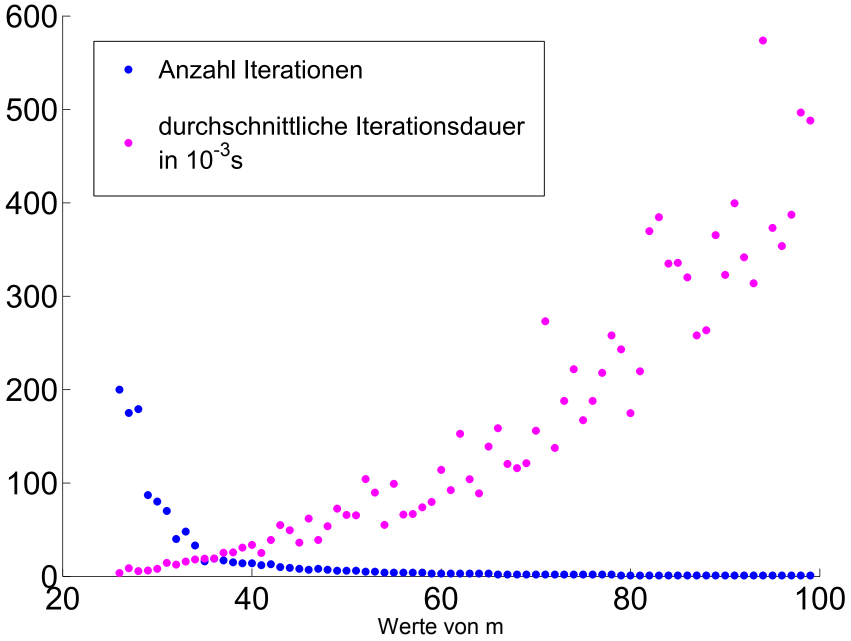

Abbildung 5: Durchschnittliche Iterationsdauer und Anzahl Iterationen vs. m

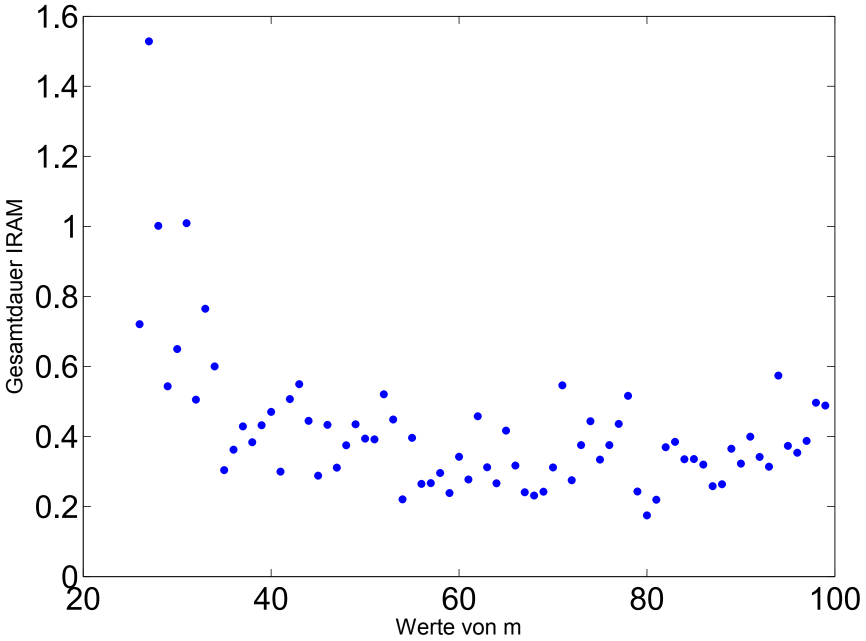

Abbildung 6: Laufzeit von IRAM in Sekunden vs. m

4 Optimierungsmethoden

In der bisherigen Version von PCCA+ wurde der Minimierungsschritt mit dem MATLAB-Befehl *fminsearch* durchgeführt. Wenn die Anzahl der Objekte oder Cluster jedoch etwas größer ist, ist die Rechenzeit von PCCA+ doch beträchtlich, was vor allem durch *fminsearch* verursacht wird. Bei Versuchen mit verschiedenen Datensätzen beanspruchte *fminsearch* stets mehr als 99 % der Laufzeit der PCCA+. (Genaueres dazu ist im entsprechenden Abschnitt des Kapitels 'Methoden und Ergebnisse' zu finden.) Daher war ein weiterer Ansatz zur Optimierung von PCCA+ die Verwendung eines anderen Minimierers.

In diesem Abschnitt soll zuerst die dem Befehl *fminsearch* zugrunde liegende Nelder-Mead-Methode dargestellt werden, und danach der Algorithmus, der nun stattdessen verwendet wird: NLSCON.

4.1 *fminsearch*

Dieser Abschnitt richtet sich nach der MATLAB-Dokumentation aus MATLAB (2009), soweit nicht anders gekennzeichnet.

Der MATLAB-Befehl *fminsearch* ist zum Minimieren von Funktionen gedacht. Er verwendet den in Lagarias u. a. (1998) beschriebenen Nelder-Mead Simplex Algorithmus. Dieser Algorithmus ist eine direkte Suchmethode, die ohne numerische oder analytische Gradienten auskommt. Laut Nelder u. Singer (2009) gewann die Nelder-Mead-Methode vor allem in den 1970ern und 1980ern an Popularität, da sie zu dieser Zeit wegen ihrer Einfachheit und dem geringen Speicherverbrauch sehr gut für die Verwendung auf Minicomputern geeignet war. Doch auch heutzutage gehört der Algorithmus zu den meistverwendeten direkten Suchmethoden.

Für diesen Algorithmus ist der Begriff des Simplex wesentlich.

Definition 21 (Simplex) *Es seien $x_1, \ldots, x_{n+1} \in \mathbb{R}^n$, so dass die Menge $\{x_i - x_1 \mid i = 2, \ldots, n+1\}$ linear unabhängig ist. Als **n-dimensionalen Simplex** (oder n-Simplex) bezeichnet man dann die konvexe Hülle dieser Punkte, also die Menge $S = \{\alpha_1 x_1 + \alpha_2 x_2 + \ldots \alpha_{n+1} x_{n+1} \mid \sum_{i=1}^{n+1} \alpha_i = 1,\ \alpha_i \geq 0\ \forall i = 1, \ldots, n+1\}$.*

Somit ist ein n-Simplex durch die Vektoren $x_1, \ldots, x_{n+1}$ charakterisiert. Diese Vektoren werden die Ecken des Simplex genannt. Für $n = 2$ ist ein Simplex daher ein Dreieck und für $n = 3$ eine Pyramide.

Der hier vorgestellte Simplex-Algorithmus sucht das Minimum einer Funktion durch die Konstruktion und iterative Einschränkung eines Simplex. In jedem Suchschritt wird ein neuer Punkt erzeugt, der im aktuellen Simplex oder nahe dabei liegt. Dann wird der Wert der zu minimierenden Funktion an diesem neuen Punkt verglichen mit dem Funktionswert an den Simplexecken. Üblicherweise wird dann eine der Ecken durch den neuen Punkt ersetzt. Somit ist ein neuer aktueller Simplex entstanden. Dies wird wiederholt, bis der Durchmesser des Simplex kleiner als eine vordefinierte Toleranz ist.

Wenn als Startpunkt x_0 gegeben ist, wird der Startsimplex folgendermaßen erzeugt: Für $i = 1, \ldots, n$ addiere 5% von $x_0(i)$ auf x_0. Jeder so erzeugte Punkt unterscheidet sich also nur in einer Komponente von x_0. Diese Punkte ergeben zusammen mit x_0 den Startsimplex. Falls $x_0(i) = 0$ ist, wird stattdessen

0.00025 verwendet. Dann wird der Algorithmus zur Minimierung der Funktion f wie folgt ausgeführt:

1. Setze als $x(i)$ die aktuellen Simplexecken.
2. Sortiere die $x(i)$ so um, dass $x(1)$ den kleinsten Funktionswert hat und $x(n+1)$ den größten. Also $f(x(1)) \leq f(x(2)) \leq \ldots \leq f(x(n+1))$.
3. Erzeuge r, den sogenannten reflektierten Punkt: $r = 2m - x(n+1)$, wobei $m = \sum\limits_{i=1}^{n} x(i)/n$.
   Berechne $f(r)$.
4. Wenn $f(x(1)) \leq f(r) < f(x(n))$, dann ersetze $x(n+1)$ durch r und gehe zu 1.
5. Wenn $f(r) < f(x(1))$ ist, dann berechne den erweiterten Punkt $s = m + 2(m - x(n+1))$ und $f(s)$.
   a) Wenn $f(s) < f(r)$, dann ersetze $x(n+1)$ durch s und gehe zu 1.
   b) Sonst ersetze $x(n+1)$ durch r und gehe zu 1.
6. Wenn $f(r) \geq f(x(n))$ ist, berechne eine Kontraktion von m und der günstigeren Wahl von r und $x(n+1)$:
   a) Wenn $f(r) < f(x(n+1))$ (also wenn r die bessere Wahl ist), berechne $c = m + (r - m)/2$ und $f(c)$.
      - Wenn $f(c) < f(r)$, dann ersetze $x(n+1)$ durch c und gehe zu 1.
      - Sonst gehe zu 7.
   b) Wenn $f(r) \geq f(x(n+1))$, berechne $cc = m + (x(n+1) - m)/2$ und $f(cc)$.
      - Wenn $f(cc) < f(x(n+1))$, dann ersetze $x(n+1)$ durch cc und gehe zu 1.
      - Sonst gehe zu 7.
7. Für $i = 2, \ldots, n+1$ berechne $v(i) = x(1) + (x(i) - x(1))/2$ und $f(v(i))$. Setze $x(1), v(2), \ldots, v(n+1)$ als neue Simplexecken.

Schritt 7 wird nur ausgeführt, wenn weder r, noch s, noch c oder cc einen niedrigeren Funktionswert als $x(n)$ haben. Dann werden alle Punkte bis auf $x(1)$ neu berechnet, und zwar so, dass die neuen Ecken jeweils den Mittelpunkt zwischen $x(1)$ und dem alten Punkt bilden. Somit werden alle Punkte näher an $x(1)$, welcher ja aktuell der Punkt mit dem kleinsten Funktionswert ist, herangerückt, siehe Abbildung 7. Daher wird dieser Fall in der MATLAB Dokumentation auch als 'Shrink', also als Schrumpfen des Simplex bezeichnet.

Der Fall, dass $x(n+1)$ durch r ersetzt wird, wird als 'Reflect' bezeichnet. Geometrisch ist der berechnete Punkt r das Ergebnis der Punktspiegelung von $x(n+1)$ an m, also an dem arithmetischen Mittel aller anderen Ecken.

Der Fall, dass s ausgewählt wird, heißt 'Expand', was sich damit erklären lässt, dass durch diesen Schritt der Simplex erweitert wird.

c steht für 'Contract outside' und cc für 'Contract inside'. Dass in diesen Fällen jeweils eine Kontraktion berechnet wird, wurde bereits erwähnt. Wie die Bezeichnung schon erahnen lässt, handelt es sich einmal um eine Kontraktion innerhalb des Simplex (Kontraktion von $x(n+1)$ und m) und einmal außerhalb des Simplex (Kontraktion von r und m). In Abbildung 8 sind an einem Beispiel die Positionen der Punkte r, s, c und cc zu erkennen.

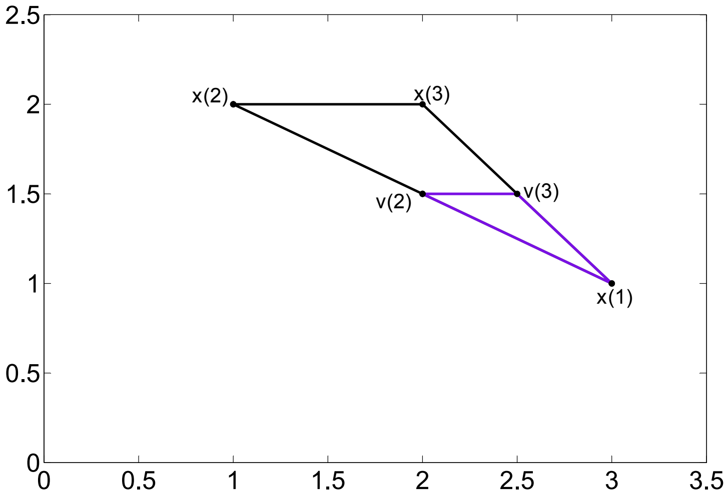

Abbildung 7: Beispiel für den Fall 'Shrink'

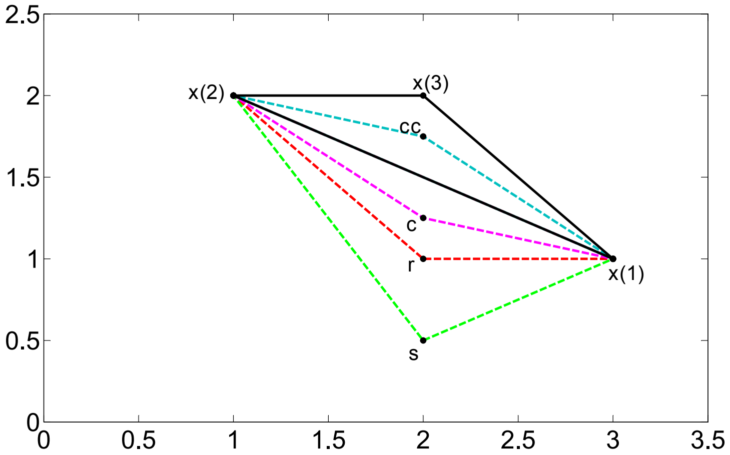

Abbildung 8: Beispiel für die Positionen der Punkte s, r, c und cc

Die Konvergenz des Verfahrens ist nicht bewiesen. Ein Zeichen für schlechte Konvergenz ist das wiederholte Ausführen von Schritt 7. Die Konvergenzrate ist höchstens linear und somit ausgesprochen langsam. Ein wesentlicher Vorteil des

Nelder-Mead-Algorithmus liegt darin, dass er Extrema findet, ohne dafür die Ableitung der zu minimierenden Funktion zu benötigen. Dieser Vorteil ist insofern bedeutsam, als sich so auch nicht-differenzierbare Funktionen minimieren lassen.

Nelder u. Singer (2009) beschreiben die folgenden Vor- und Nachteile der Methode: Dass der Algorithmus in der Praxis über so lange Zeit hinweg so viel verwendet wird, lässt sich wohl vor allem damit begründen, dass sich dieser Simplex-Algorithmus in numerischen Tests als gut darin erwiesen hat, den Funktionswert mit relativ wenigen Funktionsauswertungen recht gut zu reduzieren. So gut dies klingt, gibt es auch Nachteile und numerische Probleme: Auch für glatte und 'well-behaved' Funktionen kann es zum numerischen Breakdown kommen, was wohl von dem Mangel an Konvergenztheorie widergespiegelt wird. Es kann also passieren, dass trotz einer sehr hohen Anzahl von Iterationen keine nennenswerte Verbesserung (also Reduzierung) des Funktionswertes stattfindet, obwohl die Funktion keineswegs bereits ein lokales Minimum erreicht hat.

4.2 NLSCON

Der Algorithmus NLSCON (für 'solution of nonlinear (NL) least squares (S) problems with nonlinear constraints (CON)') ist zum Lösen nichtlinearer kleinste-Quadrate-Probleme konzipiert und insbesondere für numerisch sensitive Fälle mit oder ohne Nebenbedingungen geeignet. Das eingesetzte Verfahren ist ein gedämpfter affin invarianter Gauss-Newton-Algorithmus mit Rang-Strategie. Was das genau bedeutet, soll im Folgenden erklärt werden, wobei die Darstellungen und Definitionen Deuflhard u. Hohmann (2002) und Deuflhard (2004) folgen.

4.2.1 Newton-Verfahren

Wie erwähnt gehört NLSCON zu den Gauss-Newton-Verfahren. Um diese Klasse von Verfahren zu verstehen, soll zunächst das grundlegende Newton-Verfahren beschrieben werden.

Die Problemstellung ist hierbei die folgende: Gesucht ist die Nullstelle einer nichtlinearen skalaren Funktion f, also die Lösung x^* von $f(x) = 0$.

Es sei ein Startwert x_0 gegeben. Als erste Näherung für den Schnittpunkt von f mit der x-Achse, also x^*, wird beim Newton-Verfahren nun der Schnittpunkt der Tangente p an f im Punkt x_0 mit der x-Achse gewählt. Dies sei die erste Iterierte x_1. Die Tangente p ergibt sich als $p(x) = f'(x_0)x + f(x_0) - f'(x_0)x_0$. Sofern $f'(x_0) \neq 0$ ist, hat dann p die Nullstelle $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. So wird die Iteration nun fortgesetzt, daher ergibt sich als Iterationsvorschrift

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{52}$$

Wenn ein System $F(x) = 0$ nichtlinearer Gleichungen gegeben ist, kann der Algorithmus leicht erweitert werden:

$$F'(x_k)\Delta x_k = -F(x_k) \text{ mit } x_{k+1} = x_k + \Delta x_k \tag{53}$$

Δx_k wird als Newton-Korrektur bezeichnet.

Eine Eigenschaft des Verfahrens ist es, dass die Transformation $F \to AF$ für eine beliebige invertierbare Matrix $A \in \mathbb{R}^{n \times n}$ nicht nur zu einem arithmetisch äquivalenten Problem führt, sondern sogar zu den gleichen Iterierten x_i, sofern der Startwert x_0 fest ist. Daher wird das Verfahren *affin-invariant* genannt.

Das Newton-Verfahren hat stets eine Lösung x^* und konvergiert lokal quadratisch. Die Konvergenz hängt also vom Startwert x_0 ab. Wenn dieser zu weit von der Lösung x^* entfernt ist, ist im schlechtesten Fall keine Konvergenz gegeben. Da die Lösung nicht bekannt ist, lässt sich dies aber nicht im Voraus testen. Es gibt aber Methoden, um schon während der Iteration zu prüfen, ob das Verfahren konvergiert.

Für diesen Zweck wurde der natürliche Monotonietest eingeführt. Dieser stellt eine Bedingung zur Verfügung, anhand derer geprüft werden kann, ob die aktuellen Iterierten konvergieren, oder ob besser abgebrochen und mit einem anderen Startwert neu gestartet werden sollte.

Definition 22 (Natürlicher Monotonietest) *Es gelte $\Theta < 1$ und die vereinfachte Newton-Korrektur $\overline{\Delta x}^{k+1}$ sei definiert als die Lösung von $F'(x^k)\overline{\Delta x}_{k+1} = -F(x_{k+1})$. Dann heißt die Bedingung*

$$||\overline{\Delta x}_{k+1}|| \leq \Theta ||\Delta x_k|| \tag{54}$$

natürlicher Monotonietest*.*

Um den natürlichen Monotonietest in jedem Schritt durchführen zu können, muss also stets zusätzlich das Gleichungssystem $F'(x_k)\overline{\Delta x}_{k+1} = -F(x_{k+1})$ gelöst werden. Da dieses jedoch die gleiche Matrix $(F'(x_k))$ und einen anderen Funktionswert $(F(x_{k+1}))$ hat, ist der Aufwand hierfür nicht allzu groß.

Für das Newton-Verfahren wird in Deuflhard u. Hohmann (2002) geraten, das Verfahren abzubrechen, sobald $||\overline{\Delta x}_{k+1}|| > \frac{1}{2}||\Delta x_k||$ gilt, also der natürliche Monotonietest für $\Theta = \frac{1}{2}$ nicht mehr erfüllt wird. In diesem Fall sollte mit einem neuen x_0 gestartet werden.

Um die lokale Konvergenz zu erweitern, kann das Verfahren *gedämpft* werden. Dies bedeutet zunächst, dass ein Dämpfungsfaktor $0 < \lambda_k \leq 1$ eingeführt wird.

Definition 23 (Gedämpftes Newton-Verfahren) *Der Dämpfungsfaktor λ_k wird in die Iterationsvorschrift des normalen Newton-Verfahrens eingeschoben. Dann lautet die neue Iterationsvorschrift:*

$$F'(x_k)\Delta x_k = -F(x_k) \text{ mit } x_{k+1} = x_k + \lambda_k \Delta x_k \tag{55}$$

Als Dämpfungsstrategie, also als Strategie für die Wahl der λ_k, wird vorgeschlagen, λ_k so zu wählen, dass der natürliche Monotonietest für $\Theta := 1 - \frac{\lambda_k}{2}$ erfüllt ist. Das bedeutet, es sollte gelten $||\overline{\Delta x}_{k+1}(\lambda_k)|| \leq (1 - \frac{\lambda_k}{2})||\Delta x_k||$, wobei $\overline{\Delta x}_{k+1} = -F'(x_k)^{-1}F(x_k + \lambda_k \Delta x_k)$ die vereinfachte Newton-Korrektur des gedämpften Verfahrens ist.

Desweiteren sollte ein geeigneter Schwellwert λ_{min} gewählt werden, um zu verhindern, dass unnötig viele Iterationen durchgeführt werden, bei denen sich die Iterierten nur unwesentlich unterscheiden. Wenn die λ_k sehr klein werden würden, gilt $x_k \approx x_{k+1}$. Sobald $\lambda_k < \lambda_{min}$ ist, sollte daher die Iteration abgebrochen werden. Detaillierte Ausführungen zur Wahl einer geeigneten Dämpfungsstrategie sind in Deuflhard (2004) zu finden.

4.2.2 Gauß-Newton-Verfahren

Das Gauß-Newton-Verfahren ist eine Methode zur Lösung nichtlinearer Ausgleichsprobleme. Das bedeutet, dass zu einer Modellfunktion ϕ, in der die Parameter $x \in \mathbb{R}^n$ nichtlinear vorkommen, und zu einer Menge von Daten $b \in \mathbb{R}^m$ die Parameter x so gewählt werden sollen, dass ϕ möglichst nah an den Daten ist. 'Möglichst nah' bedeutet in diesem Falle, dass die Fehlerquadrate minimiert werden sollen. (Falls x nur linear in ϕ eingeht, bieten sich Methoden der linearen Ausgleichsrechnung an.)

Definiere die zu minimierende Funktion als $g(x) := ||F(x)||_2^2$, also ist F die Funktion, die in Abhängigkeit von x den Abstand der Datenpunkte zu ϕ misst. Es gilt also $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$, wobei D eine offene Menge sei. Weiterhin sei F zweimal stetig differenzierbar. Da die Tatsache, dass F der Abstand der Datenpunkte von der Modellfunktion ϕ ist, nicht in die Herleitungen und Beweise eingeht, kann die in diesem Abschnitt behandelte Methode nicht nur für die Lösung nichtlinearer Ausgleichsprobleme verwendet werden, sondern ist allgemeiner auch anwendbar, um eine nichtlineare Funktion zu minimieren.

Üblicherweise stehen mehr Datenpunkte zur Verfügung als Parameter zu bestimmen sind (also $m > n$), daher wird in der Literatur zuweilen nur dieser Fall betrachtet. In dem Fall, der für diese Arbeit relevant ist, ist es jedoch genau umgekehrt, es gilt $m < n$, das System ist also unterbestimmt.

Das zu lösende System nichtlinearer Gleichungen ist $F'(x)^T F(x) = 0$. Es werden nur lokale innere Minima $x^* \in D$ von g gesucht, die diese zwei hinreichenden Bedingungen erfüllen: $g'(x^*) = 0$, und $g''(x^*)$ ist positiv definit. Um daraus einen iterativen Algorithmus zu erhalten, wird das Newton-Verfahren mit diesen Bedingungen kombiniert (für eine ausführliche Darstellung siehe Deuflhard u. Hohmann (2002)), und man erhält schließlich das *ungedämpfte Gauß-Newton-Verfahren*:

$$\Delta x_k = -F'(x_k)^- F(x_k) \text{ mit } x_{k+1} = x_k + \Delta x_k \tag{56}$$

Hierbei ist $F'(x)^-$ die äußere Inverse von $F'(x)$. Die äußere Inverse einer Matrix A ist definiert als diejenige Matrix A^- mit $A^- A A^- = A^-$. Diese Bedingung wird auch als 3. Penrose-Axiom bezeichnet.

Die obige Gleichung ist äquivalent zu folgendem linearen Ausgleichsproblem:

$$||F'(x_k)\Delta x + F(x_k)||_2 = min \tag{57}$$

Aus dem ursprünglichen nicht-linearen Ausgleichsproblem ist also eine Folge linearer Ausgleichsprobleme geworden.

Wie bereits beim Newton-Verfahren erläutert wurde, ist Dämpfung eine Möglichkeit, den Konvergenzbereich eines Verfahrens zu vergrößern. Es werden wiederum $0 < \lambda_k \leq 1$ in die Iterationsvorschrift eingeschoben. Das *gedämpfte Gauß-Newton-Verfahren* ist dann definiert durch:

$$\Delta x_k = -F'(x_k)^- F(x_k) \quad \text{mit} \quad x_{k+1} = x_k + \lambda_k \Delta x_k \tag{58}$$

Das Verfahren konvergiert unter bestimmten Voraussetzungen gegen ein x^* mit $F'(x^*)^- F(x^*) = 0$. Die Folge der ungedämpften Gauß-Newton-Iterierten ist wohldefiniert, und die Iterierten bleiben innerhalb einer Kugel um x^*. Die Konvergenzrate sowie der Radius dieser Kugel hängen davon ab, wie stark nichtlinear das Problem ist. Im Fall von $F(x^*) = 0$, was als *kompatibel* bezeichnet wird, ist die Konvergenz quadratisch.

Die Existenz von x^* gilt explizit auch im rangdefekten Fall. Die Eindeutigkeit jedoch nicht: Die Lösung ist nur dann eindeutig, wenn voller Rang vorliegt. Ansonsten existiert eine Lösungsmannigfaltigkeit, deren Dimension der Rangdefekt ist.

Der für das Newton-Verfahren vorgestellte natürliche Monotonietest lässt sich auch für das Gauß-Newton-Verfahren anpassen und liefert dann entsprechende Aussagen über die Konvergenz während der Iteration.

4.2.3 NLSCON

NLSCON ist eine numerische Umsetzung der Gauß-Newton-Methode. Wie eingangs bereits beschrieben, handelt es sich bei NLSCON um ein gedämpftes Verfahren.

Bisher noch nicht erläutert wurde der Begriff der *Rang-Strategie*. Die Rang-Strategie kommt (unter anderem in NLSCON) zum Einsatz, um zu verhindern, dass das Verfahren wegen zu kleiner Dämpfungsfaktoren abbricht. Dies wird verhindert, indem der Rang künstlich reduziert wird, was häufig zu einer Erhöhung der λ_k führt. Sofern die erste Ableitung der Zielfunktion nicht verfügbar ist, kann NLSCON auch ohne diese arbeiten, indem die Jacobimatrix numerisch approximiert wird.

5 Methoden und Ergebnisse

Es wurden zwei Ansätze unternommen, um die Laufzeit der PCCA+ zu verbessern. In diesem Kapitel sollen die dafür angewandten Methoden dargestellt, und die Ergebnisse präsentiert werden.

Für alle Laufzeitangaben in dieser Arbeit gilt, dass die Rechnungen in MATLAB (Version R2009b) auf einem aktuellen Notebook mit einem Intel Core 2 Duo T6570 Prozessor und 4GB RAM durchgeführt wurden.

5.1 Erzeugung der Testdaten

Um eine große Anzahl von Testdaten, deren korrekte Clusterings bekannt sind, zur Verfügung zu haben, wurde in MATLAB ein m-File *testdata.m* zur Erzeugung von Testdaten erstellt. Eingabewerte sind k (Anzahl Cluster), n (Anzahl Objekte) und d (Dimension), sowie optional s (Skalierung der Standardabweichung). Wenn *testdata.m* gestartet wird, geschieht Folgendes:

Als erstes werden mit der MATLAB-Funktion *randi* k zufällige d-dimensionale Punkte erzeugt, deren Einträge jeweils zwischen 0 und $3k$ liegen. Diese Punkte dienen als Repräsentanten der k Cluster. Dann werden die Mächtigkeiten der Cluster zufällig festgelegt, wobei jedes Cluster mindestens $ceil(n/(2k))$ Punkte enthält. Dieser Wert wurde gewählt, damit kein Cluster so klein wird, dass es von den verwendeten Algorithmen nicht mehr als solches identifizierbar ist.

Dann wird für die am Anfang gewählten k Punkte der Abstand zu den jeweils anderen berechnet, und das Minimum von allen Abständen in der Variablen *dming* zwischengespeichert. Zu jedem der k anfangs erzeugten Punkte wird nun ein Cluster generiert, indem Folgendes so oft wiederholt wird, bis die zuvor festgelegte Größe des jeweiligen Clusters erreicht ist: Mit dem MATLAB-Befehl *normrnd* wird ein d-dimensionaler Punkt zufällig (normalverteilt) erzeugt, wobei der Mittelwert $\mu = 0$ ist. Sofern s eingegeben wurde, wird $dming/s$ als Standardabweichung verwendet. Wenn kein Wert eingegeben wird, ist die Standardabweichung $dming/6$. Die Wahl dieses Wertes beruht auf der Tatsache, dass 99,73% aller normalverteilen Werte im Intervall $\pm 3\sigma$ liegen, und auf der Überlegung, dass sich die Cluster meist nicht überschneiden sollen. (Wenn im Folgenden kein Wert für s angegeben ist, wurde somit $s = 6$ verwendet.) Der so erzeugte Punkt wird auf den ursprünglich gewählten Clusterrepräsentanten addiert, so dass schrittweise eine Punktwolke um jeden der k Repräsentanten entsteht, siehe Abbildung 9 und 10.

Zuletzt wird eine Permutationsmatrix der Größe n zufällig erzeugt, und die Reihenfolge der zuvor gewählen n Punkte wird durch Multiplikation mit dieser vertauscht.

Um die Daten direkt in dem Format zur Verfügung zu haben, welches die PCCA+ benötigt, wurde die Routine *stochmat.m* geschrieben, die zu gegebenen Datenpunkten eine zeilenstochastische Matrix P berechnet. Hierbei wird die Gauß'sche Ähnlichkeitsfunktion angewendet, um die paarweisen Ähnlichkeiten der Objekte zu berechnen. Hierfür ist die Eingabe des Wertes σ notwendig, der somit auch zu den Eingabewerten zählt, wenn im Folgenden Daten für die Benutzung der PCCA+ generiert wurden. Aus der Ähnlichkeitsmatrix S und der Grad-Matrix D wird dann P berechnet: $P = D^{-1}S$. (Um D zu berechnen, werden die Ähnlichkeiten als Gewichte verwendet.)

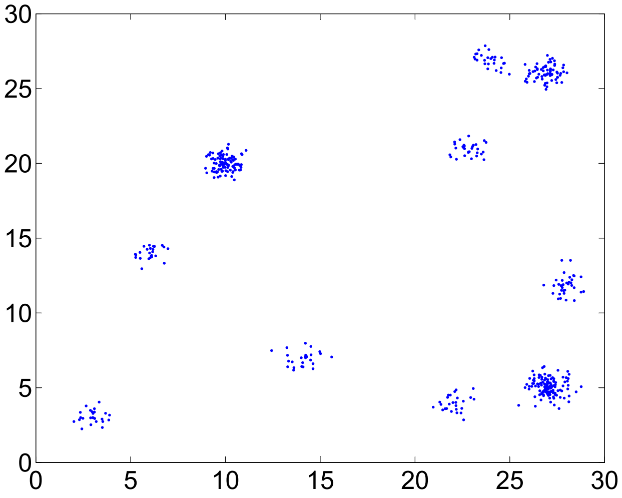

Abbildung 9: Ergebnisplot von testdata.m für $k = 10$, $n = 500$, $d = 2$

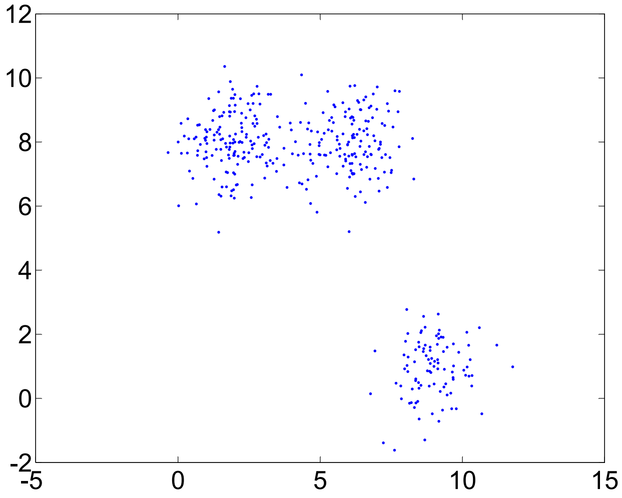

Abbildung 10: Ergebnisplot von testdata.m für $k = 3$, $n = 400$, $d = 2$, $s = 4$

Der Vorteil dieses Tools ist die Verfügbarkeit von beliebig vielen Datensätzen mit einigen nach Wunsch einstellbaren Parametern, sowie bekannten korrekten Lösungen. Bei der Verwendung der so erzeugten Daten ist demnach ein prozentualer Fehler berechenbar, so dass verschiedene Algorithmen direkt verglichen werden können.

Als Nachteil muss jedoch bedacht werden, dass die erzeugten Daten keinesfalls als repräsentativ für alle möglichen Daten gesehen werden können, auf die Clusteringalgorithmen anwendbar sind. Auf Grund der Art und Weise der Datenerzeugung stellen die Ergebnisse nur einen bestimmten Datentyp dar, so dass Vergleiche von Algorithmen anhand dieser Daten auch nicht als allgemeingültig verstanden werden sollten. Eine Möglichkeit, die Vielfältigkeit der erzeugten Daten zu erhöhen, wäre beispielsweise, nicht nur Punktwolken zu erzeugen, sondern auch Cluster von anderer Form, beispielsweise linienförmige Cluster.

5.2 Verkürzte Eigenwertberechnung

Der erste Ansatz, die Laufzeit der PCCA+ zu verringern, war motiviert durch die Überlegung, dass die Berechnung von Eigenpaaren sehr teuer ist und so stark wie möglich eingeschränkt werden sollte. Die Idee war nun, herauszufinden, ob die Informationen, die in der PCCA+ aus den Eigenpaaren benötigt werden, bei Verwendung eines iterativen Eigenwertlösers schon vor dem letzten Iterationsschritt vorhanden sind.

Um zu evaluieren, ob dieser Ansatz erfolgversprechend ist, wurde zunächst ein geeigneter Eigenwertlöser gesucht.

5.2.1 Auswahl des Eigenwertlösers

Damit die Eigenwertberechnung überhaupt früher abgebrochen werden kann, musste ein iterativer Eigenwertlöser ausgewählt werden. Auch sollte die Anzahl der berechneten Eigenpaare flexibel sein, da die PCCA+ nur so viele Eigenpaare benötigt, wie Cluster gesucht sind. Eine dritte wesentlich wichtige Eigenschaft war die gleichzeitige Berechnung der Eigenpaare. Beispielsweise besitzt der Eigenwertlöser von Jacobi-Davidson (siehe Sleijpen u. Van Der Vorst (2000)) die beiden ersten geforderten Eigenschaften, iteriert die Eigenwerte aber nacheinander und nicht gleichzeitig, so dass das Abbrechen der Iterationen nicht sinnvoll wäre, oder nur bei der Berechnung des letzten zu berechnenden Eigenwerts Iterationsschritte eingespart werden würden.

Ein Eigenwertlöser, der all diese Bedingungen erfüllt, und zudem als MATLAB-Implementierung frei verfügbar ist, ist der bereits beschriebene Algorithmus IRAM.

5.2.2 Vorgehensweise und Ergebnisse

Nachdem IRAM als Methode der Wahl feststand, wurde die MATLAB-Implementierung der PCCA+ von Sorensen (2011) dahingehend abgewandelt, dass nach jedem Iterationsschritt von IRAM die aktuellen Iterierten der Eigenpaare übergeben, und daraus ein Clustering berechnet wurde, so wie sonst aus den letzten Iterierten das Clustering konstruiert wurde. Nach dem Ende der Iteration wurde überprüft, ob es einen Iterationsschritt vor dem letzten gab, bei dem das Clustering bereits dem 'fertigen' Clustering aus dem letzten Schritt entsprach. Daraus wurde dann die Anzahl der Iterationsschritte berechnet, die hätten eingespart werden können.

In Abbildung 11 sind für zufällig erzeugte Datensätze die Anzahl der Iterationen des Eigenwertlösers geplottet gegen die Nummer desjenigen Iterationsschrittes, ab dem das Clustering sich nicht mehr ändert. (Außerdem wurde eine Hilfsdiagonale geplottet.) An diesem Plot ist zu sehen, dass in manchen Fällen Iterationsschritte hätten eingespart werden können, aber in manchen auch nicht. Wünschenswert wäre ein Ergebnis gewesen, bei dem alle Punkte in diesem Plot

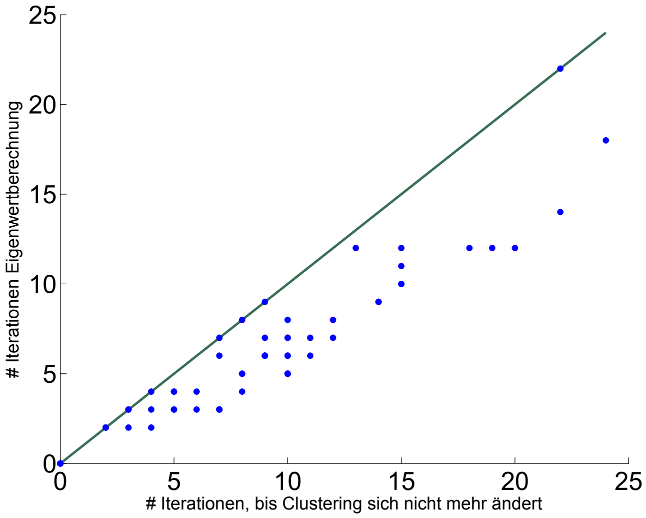

Abbildung 11: Jeder Punkt stellt das Ergebnis für ein zufällig erzeugtes Beispiel dar

unterhalb der Diagonalen liegen. Dennoch ist das Ergebnis nicht als kompletter Misserfolg zu sehen, denn es ist ja erkennbar, dass in manchen Fällen durchaus Iterationen hätten eingespart werden können. Nun stellen sich zwei Fragen:

- Lassen sich diese Fälle klassifizieren?
- Lässt sich vor Ende der Eigenwertiteration erkennen, wann das Clustering fertig ist?

Um zu testen, womit die Eigenschaft, dass Iterationen gespart werden können, zusammenhängt, wurden 300 Testdatensätze mit dem bereits beschriebenen Tool erzeugt. Hierbei wurde die Anzahl der Cluster zufällig zwischen 3 und 11 gewählt, die Anzahl der Objekte zwischen 200 und 800, die Dimension zwischen 2 und 5, s zwischen 3 und 8, und σ hatte stets den Wert 3. Auf diese Datensätze wurde IRAM angewandt und dabei, wie bereits beschrieben, stets überprüft, ab welchem Iterationsschritt das Ergebnis der PCCA+ gleich bleibt. Dabei wurde der Eingabewert m bei IRAM stets so klein wie möglich gewählt, also $m = k+1$, da sonst sehr häufig der Fall auftrat, dass IRAM nur eine einzige Iteration benötigte.

Als erstes wurde die Schärfe als möglicher Indikator für eine Iterationseinsparmöglichkeit getestet. In Abbildung 12 ist zu sehen, dass hier jedoch kein Zusammenhang besteht. Sowohl für sehr kleine, als auch für sehr große Schärfewerte gibt es Fälle, bei denen der Anteil einsparbarer Iterationen klein ist, sowie Fälle, bei denen er groß ist. Weiterhin zeigt dieser Plot, dass anscheinend nicht beliebig viele Iterationen gespart werden können, sondern dass die Obergrenze dafür etwa bei 60 % zu liegen scheint. Für diesen Plot wurden alle Datensätze ausgelassen, bei denen IRAM nur eine Iteration benötigte, da in diesem Fall kein Einsparpotential vorhanden ist.

Abbildung 13 zeigt wiederum die Schärfe geplottet gegen den Anteil der einsparbaren Iterationen. Diesmal sind die Punkte jedoch für feste Eingabewerte des Datenerzeugungstools entstanden. Alle Punkte entsprechen einem Datensatz mit $n = 793$, $d = 3$, $k = 7$, $\sigma = 3$ und $s = 5$. Auch hier ist kein Zusammenhang erkennbar.

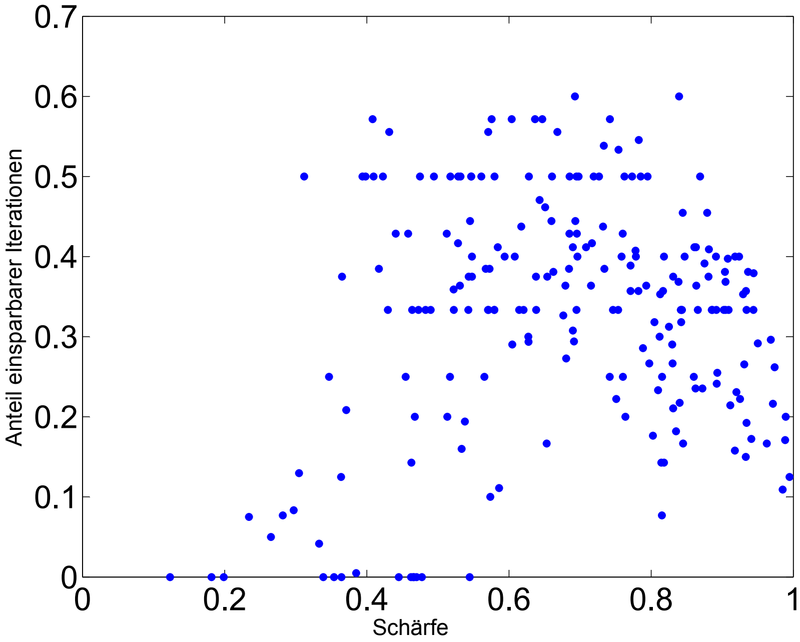

Abbildung 12: Schärfe und prozentual eingesparte Iterationen

Abbildung 14 zeigt die Anzahl der Iterationen von IRAM geplottet gegen die Schärfe. Auch hier ist kein Zusammenhang erkennbar. Die Vermutung, dass die Schärfe ein Indikator dafür sein könnte, wie aufwändig die Eigenwertberechnung ist, hat sich somit nicht bestätigt.

Ein interessanter Zusammenhang zeigt sich jedoch, wenn man für die einzelnen Clusteranzahlen die durchschnittlich einsparbaren Iterationen, sowie die durchschnittliche Anzahl von IRAM-Iterationen betrachtet. Die Werte in der folgenden Tabelle entsprechen pro Cluster jeweils dem Durchschnitt von 20 Datensätzen, die ansonsten zufällig erzeugt wurden.

Anzahl Cluster	% einsparbarer It.	IRAM-Iterationen
3	0	1
4	0	1
5	0	1
6	0.4050	11.9500
7	0.3522	18.3500
8	0.2907	37.6000
9	0.2978	25.7500
10	0.2583	39.6000
11	0.1739	46.6500

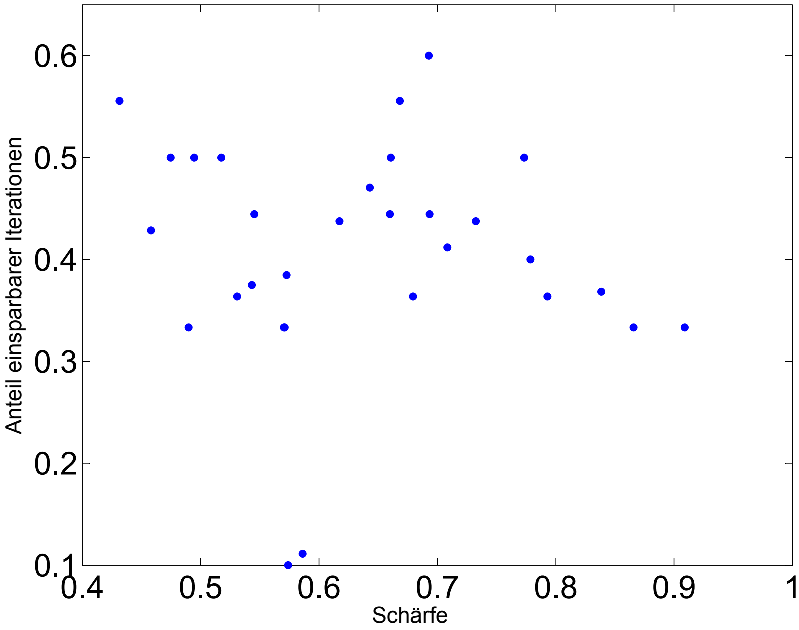

Abbildung 13: Schärfe und prozentual eingesparte Iterationen für feste Werte

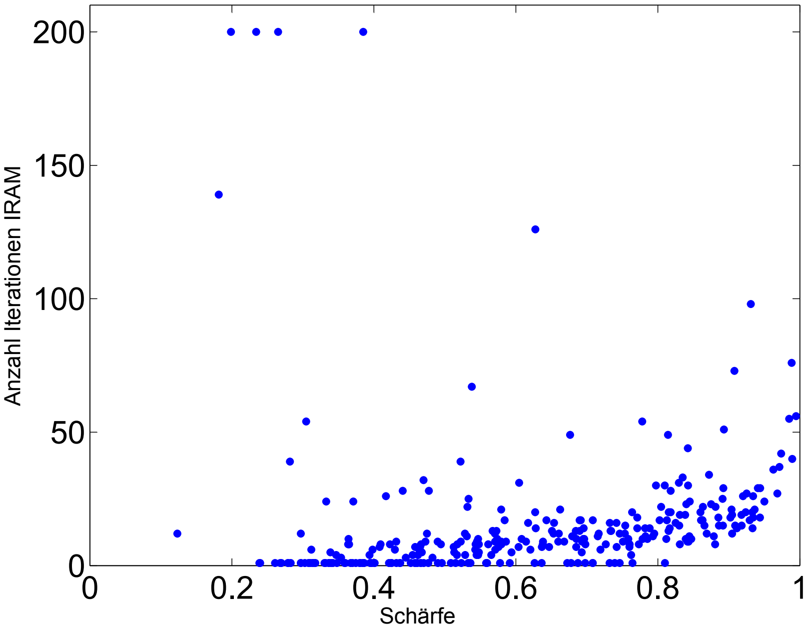

Abbildung 14: Schärfe und Anzahl IRAM-Iterationen

Hier ist zu sehen, dass zwar die Anzahl der benötigten Iterationen für die Eigenwertberechnung mit der Anzahl der Cluster steigt (bis auf die Schwankung bei $k = 9$, die damit zu erklären sein könnte, dass 20 Datensätze zu wenige sind, um einen repräsentativen Durchschnitt zu erhalten), dass aber der Anteil der einsparbaren Iterationen bei größerer Clusterzahl kleiner wird. Dass bei den Clusteranzahlen 3 bis 5 keine Iterationen gespart werden können, liegt schlicht daran, dass immer nur eine Iteration notwendig ist.

Die Frage nach der Klassifizierbarkeit derjenigen Probleme, bei denen eine Einsparmöglichkeit besteht, kann hier also zunächst nicht positiv beantwortet werden, da kein klares Kriterium gefunden wurde, anhand dessen sich ein solches Problem im Voraus identifizieren lässt.

Auch die zweite Frage kann daher nicht zufriedenstellend beantwortet werden. Ein naheliegender Vorschlag für ein Kriterium zum vorzeitigen Abbruch der Eigenwertiterationen ist, dass sich von einem Iterationsschritt zum nächsten das Clustering nicht ändert. Hierbei sollte jedoch bedacht werden, dass dafür die Minimierung der Zielfunktion mehrfach durchgeführt werden muss, was vermutlich den Laufzeitgewinn durch die eingesparten Eigenwertiterationen aufwiegen würde, so dass dadurch letztlich keine Verbesserung festzustellen wäre.

5.3 Austausch des Minimierers

Der zweite Ansatz zur Verbesserung der Laufzeit der PCCA+ ist ein sehr praktischer: Es sollte die Optimierungsmethode ausgetauscht werden, mittels derer die PCCA+ die in den vorherigen Kapiteln beschriebene Zielfunktion minimiert. Bisher wurde dafür *fminsearch* verwendet. Bei mehreren Versuchen mit mittels des bereits beschriebenen Datenerzeugungstools zufällig konstruierten Datensätzen zeigte sich, dass *fminsearch* stets mehr als 99 % der Laufzeit der PCCA+ in Anspruch nahm, was genügend Motivation für einen Wechsel zu einer schnelleren Methode liefern sollte. Die im entsprechenden Kapitel beschriebene Methode NLSCON wurde als neuer Kandidat gewählt, da sie, wie auch *fminsearch* die Möglichkeit bietet, Extrema einer Funktion zu berechnen, ohne deren Ableitung als Input zu benötigen.

Während *fminsearch* (also der Nelder-Mead-Algorithmus) gänzlich ohne Ableitung auskommt, wird bei NLSCON die Jacobimatrix im Algorithmus benötigt; falls diese nicht eingegeben wird, kann NLSCON diese aber auch numerisch approximieren. Somit ist es die Entscheidung des Benutzers, ob die Jacobimatrix, sofern vorhanden, eingegeben wird oder nicht. Da die Ableitung der Zielfunktion $\tilde{I}_2(A)$ verfügbar ist, wurden für diese Arbeit beide Varianten implementiert. Somit stehen für Laufzeitvergleiche drei Versionen der PCCA+ zur Verfügung: PCCA+ mit *fminsearch*, mit NLSCON, und mit NLSCON und Jacobimatrix (im Folgenden abgekürzt mit NLSCON (JF)).

Für den Laufzeitvergleich wurden drei Datensätze erstellt:
Test1 mit $k = 8, n = 200, d = 3, \sigma = 3$
Test2 mit $k = 8, n = 600, d = 3, \sigma = 3$
Test3 mit $k = 3, n = 1000, d = 3, \sigma = 3$.
Da die Daten mithilfe des bereits beschriebenen Tools erzeugt wurden, stand eine 'korrekte' Lösung zur Verfügung, daher konnte auch der relative Fehler eines Clusterings berechnet werden, also der Anteil der falsch zugeordneten Objekte.

Die Ergebnisse für Laufzeit (in Sekunden), Schärfe und Fehler sind in der folgenden Tabelle dargestellt:

	fminsearch	NLSCON	NLSCON (JF)
Test1 Zeit	113.76	36.66	0.54
Test1 Schärfe	0.5259	0.4783	0.3958
Test1 Fehler	0%	0%	0%
Test2 Zeit	201.27	128.98	1.27
Test2 Schärfe	0.8272	0.7872	0.7803
Test2 Fehler	0.17%	0.17%	0.17%
Test3 Zeit	4.61	12.63	1.48
Test3 Schärfe	0.5755	0.5664	0.5439
Test3 Fehler	0.4%	0.6%	0.4%
Test4 Zeit	228.11	28.29	0.52
Test4 Schärfe	0.2828	0.2484	0.2325
Test4 Fehler	1.0%	2.5%	3.0%

fminsearch liefert anscheinend stets die größte Schärfe. Bei Test1 und Test2 jedoch ist die Laufzeit von *fminsearch* sehr viel größer als die von NLSCON und NLSCON (JF). Bei Test3 war *fminsearch* schneller als NLSCON. Die kürzeste Laufzeit lieferte stets NLSCON (JF).

Außerdem ist zu beobachten, dass der Fehler stets sehr klein ist, und die Unterschiede in der Schärfe sich nicht direkt auf den Fehler übertragen. Insofern lässt sich hier sagen, dass die im Vergleich zu NLSCON (JF) deutlich höhere Laufzeit von *fminsearch* nicht durch deutlich bessere Ergebnisse gerechtfertigt wird. Die drei Beispiele legen nahe, NLSCON (JF) bevorzugt zu verwenden. Um zu testen, ob sich diese Beobachtung auch auf Fälle übertragen lässt, bei denen die Schärfe niedriger ist, und daher größere Fehler zu erwarten sind, wurde ein weiterer Testdatensatz (Test4) erzeugt. Eingabewerte waren $k = 8, n = 200, d = 3, \sigma = 5$.

Aus der Verwendung eines größeren σ resultiert hier eine kleinere Schärfe als bei den vorigen Beispielen, und anders als zuvor setzt sich die Abweichung in der Schärfe auch als größerer Fehler in den Ergebnissen fort. In diesem Beispiel ist zwar NLSCON (JF) weiterhin sehr viel schneller, das Ergebnis ist aber auch schlechter.

Insgesamt zeigt sich, dass NLSCON (JF) in allen Beispielen am schnellsten war, und *fminsearch* im Vergleich eine bis zu 400mal längere Laufzeit hatte. Bei *fminsearch* wurde die Laufzeit bei größeren Clusterzahlen teilweise sehr groß, während bei NLSCON und NLSCON (JF) zwar auch ein Zusammenhang von Clusteranzahl und Laufzeit feststellbar war, der Einfluss auf die Laufzeit jedoch anscheinend geringer ist, was das Clustern deutlich komfortabler macht.

Zu bedenken ist jedoch, dass NLSCON (JF) stets die geringste Schärfe lieferte, dass also die Optimierung betreffend *fminsearch* das eindeutig bessere Ergebnis liefert. Da die von der PCCA+ berechneten Clusterings jedoch zunächst *fuzzy* sind und danach in harte Clusterings umgewandelt werden, stellte sich an dieser Stelle die Frage, ob die harten Clusterings sich aufgrund der offensichtlich unterschiedlichen 'optimalen' A, die die Minimierer finden, voneinander unterscheiden. Die Betrachtung der Fehlerwerte zeigt hier in manchen Beispielen keinen Unterschied, jedoch kann es anscheinend bei eher schlechter Trennung der Cluster, also bei geringer Schärfe, vorkommen, dass das Clustering durch *fminsearch* deutlich weniger Fehler zeigt.

In dem Fall, dass das Ziel einer Berechnung nicht das harte Clustering ist, sondern tatsächlich das *fuzzy* Clustering interessiert, sollten NLSCON und NLSCON (JF) nicht ohne weitere Untersuchungen der Differenzen der *fuzzy* Clusterings verwendet werden, da diese hier nicht betrachtet wurden.

Es war außerdem zu beobachten, dass in den Output-Dateien von NLSCON sowie NLSCON (JF) stets stand, dass keine Konvergenz stattgefunden hatte. Dies deckt sich mit der Beobachtung, dass *fminsearch* das 'bessere Minimum' findet. Der Grund dafür, dass der Algorithmus hier nicht konvergiert, könnte darin liegen, dass während der Minimierung wiederholt *fillA.m* aufgerufen wird und den aktuellen Wert von A so anpasst, dass die Nebenbedingungen erfüllt sind.

6 Fazit

Das Ziel dieser Arbeit war, mehrere Ansätze zur Laufzeitverbesserung der PCCA+ auf ihre praktische Umsetzbarkeit zu überprüfen.

Der erste dieser beiden Ansätze war die Verkürzung der Eigenwertberechnung. Diese stellte sich als nicht sehr erfolgversprechend heraus. Zusammenfassend lässt sich zum ersten Ansatz sagen, dass weder im Allgemeinen Eigenwertiterationen eingespart werden können, noch dass sich eine Klasse von Problemen oder Datensätzen identifizieren ließ, für die Iterationen gespart werden können. Aus dem einzigen festgestellten Zusammenhang, nämlich dem zwischen dem Anteil der einsparbaren Iterationen und der Anzahl der Cluster, lässt sich jedoch kein direkter praktischer Nutzen ziehen.

Eingangs wurde die These aufgestellt, dass der Anteil einsparbarer Iterationen mit der Schärfe steigt. Dies konnte nicht bestätigt werden. Ebenso stellte sich die dieser These zugrunde liegende Annahme, dass bei größerer Schärfe weniger Iterationen für die Eigenwertberechnung notwendig sind, als nicht korrekt heraus, da kein Zusammenhang zwischen diesen Werten festgestellt wurde.

Der zweite Optimierungsansatz war der Austausch des Minimierers. Statt *fminsearch* wird nun NLSCON mit oder ohne Ableitung verwendet. Die Ergebnisse der Laufzeitanalyse zeigen, dass bei guter Trennung der Cluster die Verwendung von NLSCON mit Ableitung empfohlen werden kann, da dadurch bei nahezu gleichen Ergebnissen viel Zeit gespart werden kann. Bei schlechterer Trennung der Cluster sollte abgewogen werden zwischen einerseits Laufzeitersparnis (bei Verwendung von NLSCON mit Ableitung) und andererseits maximal guten Ergebnissen (bei Verwendung von *fminsearch*).

Die vielen Durchläufe der PCCA+, die für die Überprüfung des ersten Ansatzes notwendig waren (es wurde schließlich in jedem Iterationsschritt von IRAM einmal geclustert), wurden durch den Wechsel des Minimierers spürbar schneller. Verwendet wurde dafür NLSCON ohne Ableitung.

Für den Fall, dass die Anzahl der Cluster nicht bekannt ist, ist die PCCA+ mit NLSCON (JF) auf jeden Fall eine Verbesserung: Es kann mithilfe von NLSCON (JF) sehr schnell die beste Clusteranzahl bestimmt, und im Anschluss für diese ein weiterer Durchlauf mit *fminsearch* gestartet werden, um die Qualität des Clusterings zu optimieren.

Als Grundlage der Laufzeitanalyse wurden in dieser Arbeit nur einige Bespiele betrachtet. Eine Analyse der Laufzeiten für eine deutlich größere Anzahl von Datensätzen könnte klarere Hinweise dafür geben, in welchen Fällen und Anwendungen welche Methode empfehlenswert ist.

Wie erwähnt liefert die zuvor verwendete Methode *fminsearch* zuverlässig die größeren Schärfe-Werte. Inwiefern sich die *fuzzy* Clusterings unterscheiden, könnte ebenfalls Ziel zukünftiger Untersuchungen sein.

Ein weiterer Ansatz, die Laufzeit der PCCA+ zu verbessern, wäre möglicherweise, die Minimierungsmethode nochmals auszutauschen und dann eine Methode zu wählen, bei der Minimierung mit Nebenbedingungen möglich ist. Das wiederholte Durchführen von *fillA.m*, das im Verdacht steht, die Konvergenz von NLSCON zu verhindern, könnte somit vermieden werden. Eine Methode, die sich hier anbietet, ist der *Cutting Plane Algorithmus* (siehe Horst u. a. (2000)), der zur Minimierung konkaver quadratischer Funktionen mit linearen Nebenbedingungen konzipiert ist, also für genau solche Probleme wie das hier auftretende, da die Maximierung der konvexen Funktion $\tilde{I}_2$ der Minimierung der konkaven Funktion $-\tilde{I}_2$ entspricht.

Desweiteren bietet sich die genauere Betrachtung der in White u. Shalloway (2007) definierten Zielfunktion an, die sich von der hier verwendeten darin unterscheidet, wie der Abstand der Matrix $D_c^{-1}\tilde{\chi}^T D\tilde{\chi}$ von der Identität gemessen wird. Diese Zielfunktion könnte in die Implementierung der PCCA+ integriert werden.

Literatur

[Davies u. Bouldin 1979] Davies, D. L. ; Bouldin, D. W.: A Cluster Separation Measure. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1 (1979), April, Nr. 2, S. 224–227

[Deuflhard 2004] Deuflhard, P.: *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms.* Springer, 2004 (Springer Series in Computational Mathematics)

[Deuflhard u. Hohmann 2002] Deuflhard, P. ; Hohmann, A.: *Numerische Mathematik. I: Eine algorithmisch orientierte Einführung.* 3. de Gruyter Lehrbuch. Berlin: de Gruyter, 2002

[Deuflhard u. a. 1998] Deuflhard, P. ; Huisinga, W. ; Fischer, A. ; Schütte, C.: Identification of Almost Invariant Aggregates in Reversible Nearly Uncoupled Markov Chains / ZIB. Takustr.7, 14195 Berlin, 1998 (SC-98-03). – Forschungsbericht

[Deuflhard u. Weber 2003] Deuflhard, P. ; Weber, M.: Robust Perron Cluster Analysis in Conformation Dynamics / ZIB. Takustr.7, 14195 Berlin, 2003 (03-19). – Forschungsbericht

[Donath u. Hoffman 1973] Donath, W. E. ; Hoffman, A. J.: Lower bounds for the partitioning of graphs. In: *IBM J. Res. Dev.* 17 (1973), September, S. 420–425

[Dunn 1973] Dunn, J. C.: A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. In: *Journal of Cybernetics* 3 (1973), Nr. 3, S. 32–57

[Fiedler 1973] Fiedler, M.: Algebraic connectivity of graphs. In: *Czechoslovak Mathematical Journal* 23 (1973), Nr. 98, S. 298–305

[Fowlkes u. Mallows 1983] Fowlkes, E. B. ; Mallows, C. L.: A Method for Comparing Two Hierarchical Clusterings. In: *Journal of the American Statistical Association* 78 (1983), Nr. 383, S. 553–569

[Golub u. Van Loan 1996] Golub, G. H. ; Van Loan, C. F.: *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition).* The Johns Hopkins University Press, 1996

[Horst u. a. 2000] Horst, R. ; Pardalos, P.M. ; Van Thoai, N.: *Introduction to Global Optimization.* Springer, 2000 (Nonconvex Optimization and Its Applications)

[Jain u. a. 1999] Jain, A. K. ; Murty, M. N. ; Flynn, P. J.: Data clustering: a review. In: *ACM Comput. Surv.* 31 (1999), September, S. 264–323

[Kaufman u. Rousseeuw 1990] Kaufman, L. ; Rousseeuw, P. J.: *Finding Groups in Data An Introduction to Cluster Analysis.* New York : Wiley Interscience, 1990

[Kube u. Weber 2007] Kube, S. ; Weber, M.: A coarse graining method for the identification of transition rates between molecular conformations. In: *J. Chem. Phys.* 126 (2007), Nr. 2

[Lagarias u. a. 1998] Lagarias, J. C. ; Reeds, J. A. ; Wright, M. H. ; Wright, P. E.: Convergence properties of the Nelder-Mead simplex method in low dimensions. In: *SIAM Journal of Optimization* 9 (1998), S. 112–147

[Lehoucq u. Sorensen 1996] Lehoucq, R. B. ; Sorensen, D. C.: Deflation Techniques for an Implicitly Restarted Arnoldi Iteration. In: *SIAM Journal on Matrix Analysis and Applications* 17 (1996), Nr. 4, S. 789–821

[Lehoucq u. a. 1997] Lehoucq, R. B. ; Sorensen, D. C. ; Yang, C.: *ARPACK Users Guide: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods.* 1997

[Luxburg 2007] Luxburg, U.: A tutorial on spectral clustering. In: *Statistics and Computing* 17 (2007), December, S. 395–416

[MATLAB 2009] MATLAB: *version 7.9.0.529 (R2009b).* Natick, Massachusetts : The MathWorks Inc., 2009

[Meilă 2007] Meilă, M.: Comparing Clusterings—an information based distance. In: *Journal of Multivariate Analysis* 98 (2007), Nr. 5, S. 873–895

[Nelder u. Singer 2009] Nelder, S. ; Singer, John: Nelder-Mead algorithm. In: *Scholarpedia* 4 (2009), Nr. 2, S. 2928. – [Online; aufgerufen am 15.04.2012]

[Rand 1971] Rand, W. M.: Objective criteria for the evaluation of clustering methods. In: *Journal of the American Statistical Association* 66 (1971), Nr. 336, S. 846–850

[Röblitz u. Weber 2009] Röblitz, S. ; Weber, M.: Fuzzy spectral clustering by PCCA+. In: *WIAS Report* (2009), Nr. 26

[Saad 2003] Saad, Y.: *Iterative Methods for Sparse Linear Systems, 2nd edition.* Philadelpha, PA : SIAM, 2003

[Shi u. Malik 1997] Shi, J. ; Malik, J.: Normalized Cuts and Image Segmentation. In: *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97).* Washington, DC, USA : IEEE Computer Society, 1997, S. 731–

[Shi u. Meila 2000] Shi, J. ; Meila, M.: Learning segmentation by random walks. In: *In Advances in Neural Information Processing*, MIT Press, 2000, S. 470–477

[Sleijpen u. Van Der Vorst 2000] Sleijpen, G. L. G. ; Van Der Vorst, H. A.: A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems. In: *SIAM J. Matrix Anal. Appl* 17 (2000), S. 401–425

[Sorensen 1996] Sorensen, D. C.: *Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations.* 1996

[Sorensen 2011] SORENSEN, D. C.: *CAAM 551: Advanced Numerical Linear Algebra: Matlab Code.* `http://www.caam.rice.edu/~caam551/MatlabCode/matlabcode.html`. Version: 2011. – [Online; aufgerufen am 15.04.2012]

[Volkmann 2000] VOLKMANN, M.: *Clusteranalyse.* `http://imihome.imi.uni-karlsruhe.de/nclusteranalyse_b.html`. Version: 2000

[Weber 2006] WEBER, M: *Meshless Methods in Confirmation Dynamics*, Diss., 2006

[White u. Shalloway 2007] WHITE, B. ; SHALLOWAY, D.: Practical Macrostate Data Clustering. 2007 (physics/0703238). – Forschungsbericht

[Wikipedia 2012] WIKIPEDIA: *Arnoldi iteration — Wikipedia, The Free Encyclopedia.* `http://en.wikipedia.org/wiki/Arnoldi_iteration`. Version: 2012. – [Online; aufgerufen am 10.05.2012]

[Wright u. Trefethen 2001] WRIGHT, T. G. ; TREFETHEN, L. N.: Large-scale computation of pseudospectra using ARPACK and eigs. In: *SIAM J. Sci. Comput* 23 (2001), S. 591–605