Discrete Exterior Calculus im Maschinellen Lernen

Alexander Schier

31. Januar 2014

Diplomarbeit Informatik
Betreuer: Prof. Dr. Michael Griebel
INSTITUT FÜR NUMERISCHE SIMULATION

Mathematisch-Naturwissenschaftliche Fakultät der Rheinischen
Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

1 Einleitung

Heutzutage sind Methoden des maschinellen Lernens in vielen Bereichen zu finden, häufig ohne dass man es bewusst wahrnimmt, abgesehen davon dass einige Dinge einfach besser funktionieren. Wer eine E-Mail bekommt muss nicht mehr selber aussortieren ob sie unerwünscht ist, da diese Aufgabe von einem automatischem SPAM-Filter übernommen wird, welcher dem Nutzer Zeit und Nerven spart. Wenn man auf Smartphones Texte eingibt bekommt man intelligente Vervollständigungen der Wörter vorgeschlagen, welche zum Kontext passen, teilweise sogar schon ein passendes nächstes Wort bevor der erste Buchstabe getippt wird. Schrifterkennung wird in vielen Bereichen eingesetzt, auf der Post müssen Briefe nicht mehr per Hand sortiert werden, da Postleitzahlen von Computern inzwischen sehr gut erkannt werden können und auch bei der Digitalisierung von Büchern kann ein großer Teil des Textes automatisch umgewandelt werden. Auch bei der Erkennung von Sprache werden Silben von Computern automatisch erkannt und passend zu Wörtern zusammengesetzt, sodass es heute sogar möglich ist dem Computer ganze Texte zu diktieren, welche mit guter Genauigkeit erkannt werden.

Alle diese Probleme haben eines gemeinsam: Diese Systeme so einzustellen, dass sie ihre Funktion gut erfüllen wäre für einen Menschen eine aufwendige und komplizierte Arbeit, welche nur nach dem Versuch und Irrtum Prinzip möglich ist. Auf der anderen Seite sind zu diesen Problemstellungen viele korrekte Beispiele bekannt. Wer bisher selber die Werbung aus seinen E-Mails aussortiert hat, hat viele Beispiele für E-Mails, die er bekommen möchte und für welche, die aussortiert werden sollen. Ist ein Buch einmal digitalisiert worden, ist der originale Scan und der dazugehörige Text bekannt und bei Handschrifterkennung ist es kein Problem Handschriftproben verschiedener Personen zu sammeln.

Sind genug dieser Daten vorhanden, so ermöglicht ein Verfahren zum maschinellem Lernen es die Funktion, welche die Daten beschreibt, mit Hilfe dieser Beispieldaten zu erlernen. Der Rechner kann dazu aus einer großen Menge an möglichen Funktionen, welche Kandidaten für die Lösung des Problems sind, die Beste auswählen. Die Qualität eines solchen Kandidaten kann dabei leicht berechnet werden, indem die bekannte Lösung mit der berechneten Lösung verglichen wird. Ein Lernverfahren sucht damit in einer großen Menge von Funktionen nach der Funktion, welche möglichst wenig Fehler auf den bekannten Daten macht.

Die eigentliche Herausforderung bei einem solchem Lernverfahren ist allerdings nicht, die Trainingsdaten zu lernen, sondern später auf bisher unbekannten Daten, wie zum Beispiel einem noch nicht digitalisiertem Buch ein gutes Ergebnis zu liefern. Eine reine Bewertung nach dem Fehler auf den Trainingsdaten reicht also nicht, sondern die Funktion

muss zusätzlich darauf optimiert werden wenig Fehler zu machen, wenn sie mit neuen Daten arbeitet. Das Problem kann man sich leicht vorstellen, wenn man sich eine Funktion als Lösung vorstellt, welche eine Tabelle mit den bekannten Daten verwendet. Eine solche Funktion kann damit jede Eingabe aus den Trainingsdaten in der Tabelle finden und die korrekte Lösung liefern, aber mit einer solchen Lösung gibt es keinerlei Anhaltspunkte dafür, wie die Lösung auf Daten aussieht, welche nicht in der Tabelle stehen.

Daher ist es erwünscht, dass die Berechnung auf möglichst wenig Merkmalen der Daten aufbaut, damit die Erkennung robust gegenüber unwesentlichen Änderungen der Merkmale ist. So soll zum Beispiel die Ziffer acht nicht durch ihre einzelnen Bildpunkte erfasst werden, da sie dann bei einer etwas schieferen Schreibweise nicht mehr korrekt erkannt werden kann. Stattdessen könnte zum Beispiel ein Merkmal „Enthält zwei Kreise“ als Funktion verwendet werden, welche sie von anderen Ziffern unterscheiden kann, selbst wenn sie undeutlich geschrieben ist.

Der Ansatz, die Lösung möglichst einfach zu halten, wird Occam's Razor [BEHW87] genannt, was eine Metapher dafür ist, dass unnötige Komplexität wie mit einem Rasiermesser abgeschnitten wird, sodass nur der Kern der These erhalten bleibt. Dieses Sparsamkeitsprinzip, welches ursprünglich eine Aussage über verschiedene wissenschaftliche Erklärungen der gleichen Daten war, spielt beim Lernen von Funktionen eine Rolle, da auch eine maschinell gelernte Funktion eine Erklärung der gelernten Daten darstellt.

Die Einhaltung dieses Prinzips wird beim maschinellem Lernen dadurch umgesetzt, dass neben dem Fehler auch die Komplexität der Funktion, welche die Daten „erklärt“ bestraft wird, wodurch einfachere Funktionen, als die Funktionen bevorzugt werden können, auch wenn eine komplexe Funktion einen etwas geringeren Fehler aufweist.

So wird eine Funktion, welche die acht über diese einfache Beschreibung erkennt, eine wirklich verschmierte Ziffer aus den Trainingsdaten nicht mehr erkennen, dafür aber bei einer späteren Anwendung auf ungelernten Ziffern eine bessere Erkennung für deutlich geschriebene Zahlen erlauben. Das Verhältnis, wie groß der Fehler beim Training sein darf und wie komplex die Funktion werden darf lässt sich dabei mit einem Regularisierungsfaktor balancieren.

Es ist möglich diese Regularisierung mit Hilfe des sogenannten semi-supervised learnings weiter zu verbessern, indem man zu den vorhandenen Trainingsdaten weitere Daten ohne bekannte Lösung hinzunimmt. Diese sind häufig in großer Menge vorhanden, wie zum Beispiel durch eine Menge noch nicht digitalisierter Buchscans. Diese Daten lassen sich zwar nicht dazu nutzen den Fehler der Funktion beim Lernen zu messen, aber sie können genutzt werden um die Trainingsbeispiele besser einzuordnen. So ermöglichen beispielsweise handschriftliche Ziffern ohne bekannte Lösung eine Gruppierung zwischen den Zahlen, sodass beim Lernen der Trainingsbeispiele mit Musterlösung dann der ganzen Gruppe eine bestimmte Ziffer als Lösung zugeordnet werden kann.

Technisch werden die Eingabedaten als hochdimensionale Punkte modelliert und Gemeinsamkeiten zwischen den Punkten bedeuten, dass sie auf einer gemeinsamen Mannigfaltigkeit (einer „Form“ im Raum) liegen. Hat eine Eingabe beim Lernen zum Beispiel drei Eigenschaften, so kann man sie sich als Punkt der innerhalb eines Würfels liegt vor-

stellen. Sind sich verschiedene Daten ähnlich, liegen die Punkte nah beieinander, sind sie unterschiedlich haben sie eine größere Distanz. Eine Klassifikation, zum Beispiel nach verschiedenen Ziffern, versucht nun diese Gruppen zu erkennen und voneinander zu trennen. Dabei definiert eine Gruppe zum Beispiel das Merkmal „Die Ziffer ist eine Acht“.

Liegt ein Punkt hinter der Grenze zwischen den anderen Gruppen und dieser Gruppe, stellt er also vermutlich eine Acht dar. Ein Operator, welcher die Ähnlichkeit innerhalb einer solchen Gruppe beschreibt und das Lernen aus diesen Punkten ermöglicht, ist der Laplaceoperator. Dieser lässt sich aus den Trainingspunkten und den gegebenen Punkten ohne Trainingslösung auf verschiedene Weisen berechnen, und kann für die Regularisierung der Komplexität verwendet werden.

Mit Punkten im dreidimensionalem Raum werden auch Modelle im Computer beschrieben, womit zum Beispiel die Figuren in Animationsfilmen berechnet werden. Mit solchen Modellen lassen sich aber nicht nur Modelle darstellen, sondern sie werden auch verwendet um damit Simulationen zu berechnen. So kann heutzutage vor dem eigentlichem Crashtest eines Auto-Prototypen in einer Art „virtuellem Labor“ eine Simulation durchgeführt werden, mit welcher bereits einige technische Mängel vor dem eigentlichem Test ermittelt werden können, um die entsprechenden technische Details vorher noch zu verändern. Durch solche Simulationen sind weniger Tests nötig, was der Automobilfirma Zeit und Geld spart, da weniger reale Prototypen gebaut werden müssen.

Eine der Herausforderungen einer solchen Simulation ist, dass durch eine Modellierung Fehler entstehen. Zum Beispiel die Motorhaube des Autos ist gekrümmt, wodurch sie nicht mit endlich vielen Punkten und Dreiecken im Computer dargestellt werden kann, weswegen jedes Modell nur eine Annäherung an die Realität sein kann. Gerade bei der Modellierung physikalischer Prozesse gibt es daher verschiedene Ansätze mit dieser Vereinfachung umzugehen, welche Unterschiede aufweisen an welchen Stellen Fehler entstehen und wie sie sich bei weiteren Rechnungen fortpflanzen.

Ein Ansatz für solche Simulationen ist der *Discrete Exterior Calculus* (DEC), welcher es erlaubt auf Objekten wie Dreiecken, Kanten und Punkten abstrakt zu rechnen. Dazu wird für viele Operationen nur die Beziehung zwischen den Objekten verwendet, ohne konkrete Werte, wie die Lage der Punkte im Raum, mit einfließen zu lassen. Erst nach den abstrakten Rechnungen werden die Werte hinzugenommen, wodurch der Fehler durch das vereinfachte Modell erst an dieser Stelle in die Rechnung einfließt. Die Elemente des Modells übernehmen dabei Aufgaben, welche intuitiv passend erscheinen. So ist die Änderung des Werts zwischen zwei Punkten auf der Kante, welche diese beiden Punkte verbindet definiert und bei einer Strömungssimulation in zwei Dimensionen ist der Fluss des Wassers, welches von einem Dreieck über eine Kante hinweg in ein angrenzendes Dreieck fließt auf dieser Kante gespeichert.

Dass die Operatoren alle auf den Beziehungen zwischen Objekten definiert sind, ohne dass zur Anwendung des Operators numerische Werte verwendet werden müssen, bedeutet mathematisch, dass wichtige Integralsätze, wie der Satz von Green und der Satz von Stokes exakt stimmen und an dieser Stelle keine Fehler entstehen, welche sich in der weiteren Rechnung fortpflanzen.

Neben dem Lösen von PDEs, um physikalische Simulationen auf Modellen durchzuführen, liefert der DEC dadurch, dass er einen Laplaceoperator definiert, auch Hilfsmittel zum maschinellem Lernen. Der Laplaceoperator des DEC hat den Vorteil, dass er durch das gegebene Modell eindeutig definiert ist und sehr gut an die Daten des Modells angepasst ist, während andere Laplaceoperatoren beim maschinellem Lernen aus den Punkten generiert werden und die Qualität des Ergebnisses stark von der Art der Erzeugung abhängt.

Die Kombination der beiden Ansätze ist, dass mit dem DEC eine Simulation auf einem 3D-Modell gerechnet werden kann, aus welcher dann eine Funktion der Ergebnisse gelernt wird, mit der es möglich ist Daten zu berechnen auf Punkten, die nicht simuliert wurden. Dauert zum Beispiel eine Simulation auf einem genauem Modell zu lange, lässt sich auf einem etwas gröberem Modell die Simulation berechnen und dann das Ergebnis als Trainingsdaten beim maschinellem, Lernen zu verwenden, um den Wert auf den Punkten des feinen Modells zu interpolieren.

Der Vorteil einer Regularisierung mit dem DEC-Laplace ist hierbei, dass Punkte die nah beieinander liegen, aber im Modell nicht verbunden sind, sich beim Lernen nicht gegenseitig beeinflussen. Berechnet man die Wärmeleitung auf einem gebogenen Blech, bei welchem sich zwei entgegengesetzte Enden nah kommen ohne sich zu berühren, dann überträgt sich die Wärme hauptsächlich entlang des Blechs, da die Wärmeleitung durch die Luft deutlich geringer als die des Metalls ist. In einem typischem Lernverfahren würden sich die nahen Punkte der beiden Enden des Blechs gegenseitig stark beeinflussen, während sie sich in der Simulation nur sehr gering beeinflusst haben. Verwendet man zur Regularisierung den DEC-Laplace, welcher aus dem Modell selber berechnet wurde, nutzt auch das Lernverfahren die Form des Modells aus und berechnet damit eine Funktion, welche dieses Problem nicht aufweist.

Eigene Beiträge

Diese Arbeit baut auf den Arbeiten von Ulrike von Luxburg [VL07] zum Graph-Laplace, Mikhail Belkin [BN04] zum semi-supervised learning und Anil N. Hirani [Hir03] zum Discrete Exterior Calculus auf und erweitert die dort beschriebenen Themen des DEC und des semi-supervised learnings um die Betrachtung einer Kombination beider Verfahren.

Eigene Beiträge dieser Arbeit sind:

- Evaluierung des semi-supervised learnings mit einem Graph-Laplace aus den Daten.
- Berechnung von Beispielfunktionen mit einer eigenen Implementation des semi-supervised learnings.
- Vergleich eines Graph-Laplaceoperators mit dem Laplaceoperator des Discrete Exterior Calculus.
- Berechnung verschiedener PDEs mit einer eigenen Implementation eines DEC-Frameworks.

- Betrachtung verschiedener Möglichkeiten Graphen und Simplex-Komplexe aus hochdimensionalen Punkten zu konstruieren.
- Verwendung des DEC-Laplaceoperators beim semi-supervised learning von Daten auf Meshes.

Aufbau der Arbeit

Diese Arbeit gliedert sich in vier Abschnitte, welche jeweils die Theorie der verwendeten Methoden erläutern um danach selbst gerechnete Ergebnisse zu zeigen:

- *Graphen*: Wiederholung der wichtigsten Graph-Definitionen, welche beim DEC verwendet werden, Konstruktion von Graphen aus (hochdimensionalen) Punkten und schließlich die Definition eines Graph-Laplaceoperators.
- *Discrete Exterior Calculus*: Beschreibung des Discrete Exterior Calculus, Definition effizienter Datenstrukturen und Methoden zur Implementation des DEC, Vorstellung von Möglichkeiten Simplex-Komplexe aus hochdimensionalen Daten zu erzeugen, Anwendung des DEC zum Lösen von PDEs verschiedener physikalischer Probleme und Vorstellung von Lösungen, die mit einer eigenen Implementation berechnet wurden.
- *Maschinelles Lernen*: Vorstellung des semi-supervised learnings mit Hilfe von Kernen und einer Regularisierung über einen Graph-Laplaceoperator, Betrachtung der Unterschiede zwischen Graph-Laplace- und DEC-Laplaceoperatoren und das Ergebnis von Beispielrechnungen mit einem selber implementierten Programm zum Lernen aus Trainingsdaten und Datenpunkten ohne bekannten Trainingswert.
- *DEC und maschinelles Lernen*: Vorstellung eines Laplacekerns aus den Eigenwerten und Eigenvektoren eines Graph-Laplaceoperators, Vergleich des Graph-Laplacekerns aus einem generiertem Graphen und des DEC-Laplaceoperators und Lernen von Daten auf dreidimensionalen Modellen mit Hilfe von semi-supervised learning mit einer Laplace-Regularisierung.

2 Graphen

2.1 Einleitung

Die Graphentheorie ermöglicht es, Beziehungen zwischen Objekten zu modellieren, wobei es an die Menge der Objekte eines Graphen keine besondere Anforderung gibt. Anschauliche Beispiele für Mengen, auf denen mit Graphen interessante Dinge berechnet werden können, sind zum Beispiel Städte, Personengruppen, Webseiten und Bücher (z.B. wer zitiert wen). Zu der Menge der Objekte gehört eine Menge von Beziehungen an welche je nach Graph verschiedene Anforderungen gelten können.

Eine Einführung und elementare Algorithmen auf Graphen finden sich in [BM76], [Kön01]. An dieser Stelle werden nun einige grundlegende Definitionen und Algorithmen kurz definiert, welche für die weiteren Teile der Arbeit relevant sind.

2.2 Formale Definition

Ein Graph $G = (V, E)$ hat eine Menge von Objekten V (die Knoten des Graphen) und eine Menge E von Beziehungen (die Kanten des Graphen) zwischen jeweils zwei Objekten. Die Kanten sind eine Teilmenge $V \times V$, wobei für bestimmte Graphen weitere Einschränkungen gelten.

Aus verschiedenen Arten die Kantenmenge zu definieren folgen unterschiedliche Arten von Graphen, die für verschiedene Zwecke genutzt werden.

Die Kantenmenge speichert Tupel von jeweils zwei Knoten $[v_i, v_j]$ mit $v \in V$ Die Knotenmenge ist eine geordnete Liste der Objekte, welche die Knoten des Graphen bilden, wobei für Graphalgorithmen nur der Index und ggf. die Gewichtung des Knoten nötig ist.

Komplement

Für eine Teilmenge $V' \subseteq V$ der Knotenmenge V ist das Komplement definiert als $\overline{V'} = V \setminus V'$. Insbesondere gilt $\overline{\emptyset} = V$ und $\overline{V} = \emptyset$.

Gerichtete und ungerichtete Graphen

Ein Graph mit einer Kantenmenge $E = \{[v_1, v_2] | v_1, v_2 \in V\}$ mit $(v_1, v_2) \neq (v_2, v_1)$, bei der die Reihenfolge der Knoten der Kante eindeutig gespeichert wird, heißt *gerichtet*.

Ein ungerichteter Graph hat eine Kantenmenge $E = \{\{v_1, v_2\} | v_1, v_2 \in V\}$, bei welcher Kanten als eine Menge $\{v_1, v_2\}$ gegeben sind. Eine solche Menge definiert nur eine Verbindung zwischen den Knoten v_i, v_j, aber keine Richtung.

Schreibweise für Kanten

Wir schreiben Kanten mit den Eckpunkten v_1, v_2 bei gerichteten Graphen als $[v_1, v_2]$, und bei ungerichteten Graphen als Menge $\{v_1, v_2\}$. Ist es irrelevant ob der Graph gerichtet ist, schreiben wir ggf. auch $[v_1, v_2]$, da die Kante in beliebiger Orientierung betrachtet werden kann.

Sei $e = [v_1, v_2] \in E$ eine Kante und $v \in V$ ein Knoten, dann definieren wir $v \in E$ als Relation, dass v ein Endpunkt einer Kante ist, d.h. $\exists e = [v_1, v_2] \in E$, sd. $v = v_1 \vee v = v_2$. Eine gerichtete Kante e zeigt von v_1 auf v_2, und wir nennen v_1 den Startpunkt und v_2 den Endpunkt der Kante.

Gewichte und Knotengrad

Ein Graph mit Kantengewichten weist jeder Kante ein Gewicht zu. Dieses kann zum Beispiel durch eine erweiterte Kantenmenge $E \subseteq V \times V \times \mathbb{R}$ oder durch eine Funktion $E \to \mathbb{R}$ gespeichert werden. Weiterhin heißt ein Gewicht von 0, dass die Kante nicht existiert. Knoten können ebenfalls Gewichte zugeordnet werden, welche z.B. mit durch eine Abbildung $V \to \mathbb{R}$ definiert werden.

Der Knotengrad ist definiert als die Anzahl Kanten e, die den Knoten v als einen ihrer Endpunkte haben, d.h. $\text{grad}(v) = |\{e | e \in E, w \in V \text{ mit } e = \{v, w\}\}|$. Für einen gerichteten Graphen definieren wir den Eingangsgrad eines Knotens v als die Anzahl der Kanten $e = [v_1, v_2],$ mit $v = v_2$ und den Ausgangsgrad eines Knotens als die Anzahl der Kanten $e = [v_1, v_2],$ mit $v = v_1$.

Pfade

Zwei Knoten v_i, v_j sind durch einen Pfad verbunden, wenn es Kanten $[v_i, v_{i+1}], \ldots, [v_{j-1}, v_j]$ gibt, sodass die Knoten verbunden sind. Ist der Graph gerichtet, müssen die Kanten dabei gemäß ihrer Orientierung durchlaufen werden. Pfad(v, w) bezeichnet einen (beliebigen) Pfad von v nach w.

- Die Länge eines Pfades ist die Summe von Kantengewichten der durchlaufenen Kanten. Ist der Graph ungewichtet, nehmen wir die Kantengewichte als 1 an und die Länge entspricht gerade der Anzahl der Kanten.
- Ein Pfad, der am gleichem Knoten endet wie er beginnt wird Kreis genannt. Ein Graph, der keine Kreise enthält wird als *kreisfrei* bezeichnet.
- Der *vollständige Graph* einer Knotenmenge V besitzt zwischen allen Paaren von Knoten eine Kante. D.h. es gilt $\{v_1, v_2\} \in E \; \forall v_1, v_2 \in V$.
- Ein Knoten hat eine Schleife (Eigenkante), wenn es eine Kante $e \in E$ gibt mit $e = (v_1, v_2),\ v_1 = v_2 \in V$. Wenn in einem Graph keine Schleifen hat, gilt $E = \{(v_1, v_2) | v_1, v_2 \in V,\ v_1 \neq v_2\}$.

Zusammenhangskomponenten

Eine Teilmenge von Punkten $C \subseteq V$ ist eine *Zusammenhangskomponente* (ZHK) von G, wenn alle Knoten innerhalb der ZHK durch Pfade verbunden sind, und es keinen Pfad

gibt, der einen Knoten außerhalb der ZHK mit einem Knoten in der ZHK verbindet:

$$\begin{aligned} &\exists \text{Pfad}(v, w) \quad \forall v, w \in C \\ &\nexists \text{Pfad}(v, w) \quad \forall v \in C, w \in \overline{C} \end{aligned}$$

Baum

Ein Baum ist ein zusammenhängender kreisfreier Graph. Knoten mit Grad 1 heißen Blätter, alle anderen Knoten sind innere Knoten.

Ein gerichteter Baum ist ein gerichteter kreisfreier Graph, in dem genau ein Knoten Eingangsgrad 0 hat und der Eingangsgrad aller anderen Knoten 1 ist. Der Knoten mit Eingangsgrad 0 heißt die Wurzel des Baums, alle Knoten mit Ausgangsgrad 0 heißen Blätter.

Bemerkung: Dass der Graph zusammenhängend ist folgt daraus, dass er kreisfrei ist und genau eine Wurzel hat.

Ein *minimaler Spannbaum* eines ungerichteten zusammenhängenden Graphen ist ein Teilgraph, der einen ungerichteten Baum der Knoten des Graphen mit minimaler Summe der Kantengewichte ergibt. Damit enthält ein Spannbaum die minimale Anzahl Kanten, die nötig sind, damit die Menge der Knoten einen zusammenhängenden Graphen ergibt.

Beispiele

Ungerichtete Graphen

Ein Beispiel für einen ungerichten Graph mit Kantengewichten ist ein Städtenetz, wie es in Grafik 2.1 zu sehen ist. Dabei entspricht eine Kante der Verbindung von zwei Städten durch eine Straße. Dabei wird angenommen, dass die Straßen keine Einbahnstraßen sind, welche gerichteten Kanten entsprechen würden.

Gerichtete Graphen

Ein Beispiel für einen gerichteten Graphen ist der Webgraph. Unter dem Webgraph versteht man einen Graphen, der die Linkstruktur zwischen verschiedenen Webseiten modelliert. Ein Beispiel für eine solche Linkstruktur ist in Grafik 2.1 zu sehen, welche einen Graphen zeigt, der die Struktur von Links zwischen unterschiedlichen Domains modelliert.

Wenn auch Links von der Webseite auf sich selber gezählt werden, kann der Graph Schleifen enthalten. Erfasst man jeden Hyperlink einzeln, bekommt man einen Graphen mit Mehrfachkanten, da eine Seite mit mehreren Links auf eine andere Seite verlinken kann.

Bäume

Ein Beispiel für einen (gerichteten) Baum ist die Verzeichnisstruktur eines Computers. Jeder Ordner und jede Datei liegt in genau einem Ordner und hat daher Eingangsgrad 1, außer dem Wurzelverzeichnis „/“, welches die Wurzel des Baums ist.

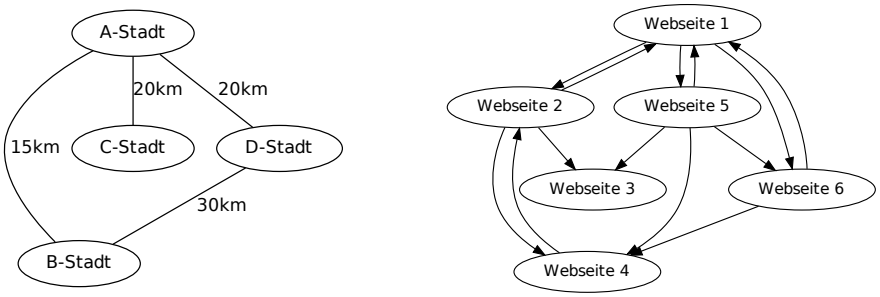


Abb. 2.1: **Links:** Ein Straßennetz zwischen Städten ist ein ungerichteter Graph, dessen Kantengewichte die Länge der Strecken sind. **Rechts:** Beispiel für einen gerichteten Graphen: Die Linkstruktur zwischen verschiedenen Internet-Domains.

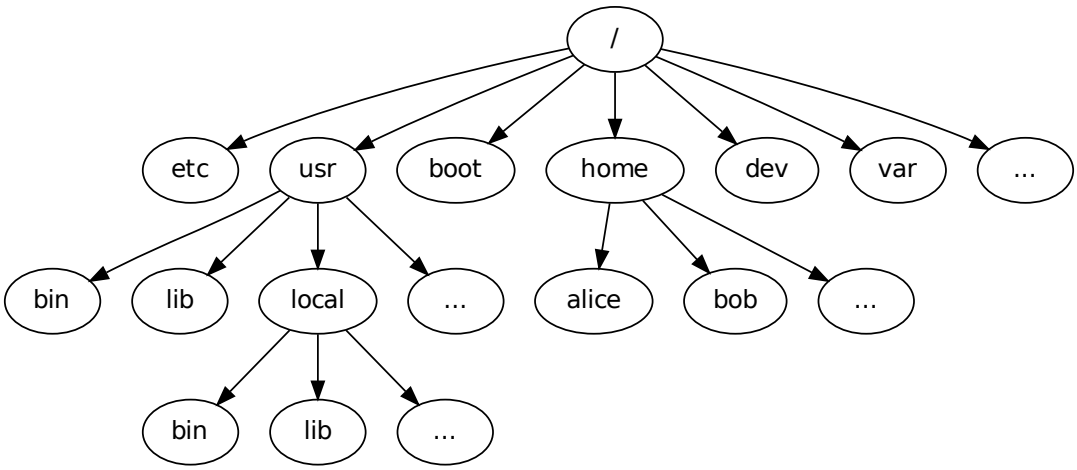


Abb. 2.2: Die Linux-Verzeichnisstruktur ist ein Beispiel für einen Baum.

$$V = [v_1 = \text{A-Stadt}, v_2 = \text{B-Stadt}, v_3 = \text{C-Stadt}, v_4 = \text{D-Stadt}]$$
$$\mathbf{A} = \begin{pmatrix} 0 & 15 & 20 & 20 \\ 15 & 0 & 0 & 30 \\ 20 & 0 & 0 & 0 \\ 20 & 30 & 20 & 0 \end{pmatrix}$$

Abb. 2.3: Die Adjazenzmatrix für den Städtegraph aus Abbildung 2.1

2.3 Speicherung von Graphen

Die Speicherung der Knoten kann mit einer einfachen Liste erfolgen und jeder Knoten erhält seine Position in der Liste als eindeutigen Index. Eventuelle Knotengewichte können in einer Liste mit der gleichen Indexmenge gespeichert werden.

Zur Speicherung der Kantenmenge bieten sich je nach Anwendungszweck verschiedene Strukturen an, unter anderem eine Liste der Kanten $[v_1, v_2, \text{Gewicht}]$, oder eine implizite Speicherung indem je Knoten eine Liste für die ausgehenden und/oder eingehenden Kanten gespeichert wird, $E_v = ((v_1, \text{Kanten-Gewicht}), (v_2, \text{Kanten-Gewicht}), \dots)$.

Zur Speicherung eines Graphen mit Kantengewichten und ohne Mehrfachkanten kann auch eine Adjazenzmatrix $\mathbf{A}$ definiert werden, bei der an der Stelle A_{ij} das Gewicht der Kante $[v_i, v_j]$ steht.

Ist der Graph ungerichtet, dann ist die Matrix $\mathbf{A}$ symmetrisch und wenn der Graph nicht gewichtet ist, gilt $\mathbf{A}_{v_1,v_2} \in \{0, 1\}$.

Wir verwenden für die folgenden Algorithmen die Adjazenzmatrix wie oben definiert, da diese damit effizient zu implementieren sind.

Ein Beispiel für eine solche Adjazenzmatrix zum Graphen aus Abbildung 2.1 ist ein Abbildung 2.3 zu sehen.

2.4 Algorithmen für Graphenprobleme

- *Tiefen-/Breitensuche*: Eine Tiefen- bzw. Breitensuche sucht Knoten in einem Baum, ausgehend von der Wurzel. Die Tiefensuche verzweigt sich bei jedem Knoten zuerst in die Unterknoten, während die Breitensuche erst alle Nachbarknoten abarbeitet.

- *Dijkstra-Algorithmus*: Ein Greedy-Algorithmus um in einem Graphen mit nichtnegativen Kantengewichten alle kürzesten Pfade zu finden. Eine formale Beschreibung findet sich in [BM76, Seite 19ff]. In einer naiven Implementierung mit einer Liste ergibt sich eine Laufzeit von $\mathcal{O}(|V|^2+|E|)$, mit einem Fibonacci-Heap lässt sich die Laufzeit auf $\mathcal{O}(|V| \log |V| + |E|)$ reduzieren[FT87]. Eine effiziente Berechnung ist auch mit Hilfe dynamischer Programmierung möglich[Sni06].

- *Bellman-Ford Algorithmus*: Der Bellman-Ford Algorithmus berechnet die kürzesten Wege von einem Startknoten aus, und kann im Gegensatz zum Algorithmus von Dijkstra auch mit negativen Kantengewichten umgehen. Der Algorithmus wählt aus der Menge der Kanten jeweils die minimale Kante, welche mit den bisherigen Kanten keinen Kreis bildet, bis der Spannbaum konstruiert ist (und damit jede weitere Kante einen Kreis erzeugen würde). Die Konstruktion ist nicht deterministisch, aber jedes Ergebnis ist ein minimaler Spannbaum, d.h. die Summe der Kantengewichte ist gleich. Die Laufzeit des Algorithmus liegt in $\mathcal{O}(|V| \cdot |E|)$ um von einem Startknoten aus die Wege zu allen anderen zu finden. Für alle kürzesten Wege ergibt sich entsprechend $\mathcal{O}(|V|^2 \cdot |E|)$. Eine formale Beschreibung findet sich in [Bel56].

- *Kruskal-Algorithmus*: Der Kruskal-Algorithmus berechnet einen minimalen Spannbaum, wie er in Abschnitt 2.2 definiert ist. Sind alle Kantenlängen unterschiedlich, ist der Spannbaum eindeutig definiert, ansonsten kann es verschiedene Spannbäume geben, bei denen aber die Summe der Kantengewichte gleich ist. Die Laufzeit ist $\mathcal{O}(|E| \cdot \log^* |V|)$, wobei log^* die minimale Anzahl an Anwendungen des Logarithmus ist, die benötigt wird damit $\log(\log(\ldots \log(n))) \leq 1$ gilt. Die formale Definition findet sich im ursprünglichem Paper [Kru56].

Weitere interessante Graphenprobleme

Eines der bekanntesten Graphenprobleme ist das Problem des Handlungsreisenden[Flo56].

Die Aufgabe ist aus einer Menge von Städten, die in beliebiger Reihenfolge besucht werden dürfen, die kürzeste Rundreise zu bestimmen. Die Problemstellung kann zum Beispiel so formuliert werden, dass in einem vollständigem Graphen der kürzeste Pfad von einem Knoten zu sich selber gesucht wird, welcher jeden Knoten $v \in V$ enthält[1].

Das Problem ist NP-vollständig[Pap77] und daher ist nicht bekannt ob es in polynomieller Zeit lösbar ist.

Es gibt eine einfache Approximation, die einen Pfad konstruiert, der höchsten zweimal mal so lang wie die optimale Lösung ist.

Dazu wird der minimale Spannbaum der Punkte zu konstruiert und dieser dann mit einer Tiefensuche durchlaufen. Dabei wird jede Kante zweimal besucht und die Rundreise beginnt und endet an der Wurzel. Da der Spannbaum die Summe der Kantengewichte minimiert, ist der Pfad höchsten doppelt so lang wie die optimale Lösung.

Pagerank

Ein prominenter Graphenalgorithmus ist der PageRank-Algorithmus, den Page, Lawrence, und Weitere in [PBMW99] beschrieben haben, welcher das World Wide Web als gerichteten Graphen modelliert. Dabei entsprechen die Knoten einzelnen Webseiten, und die Kanten den Hyperlinks zwischen verschiedenen Webseiten.

Das Gewicht einer Kante von einem Knoten v auf einen Knoten u errechnet sich aus dem Quotienten des (bekannten) Pageranks der Webseite v und der Anzahl der Seiten

[1] Wobei die Kantengewichte die Länge der Wege darstellen

auf die v verlinkt. Aus einer Summe über alle v, welche auf die Webseite u verlinken, lässt sich dann der Pagerank der Webseite u berechnen.

Die Berechnung des Pageranks aus einem gegebenen Graphen ist ein rekursives Problem, welches mit einer beliebig gewählten Menge an Knotengewichten als Startbedingung[2] gegen eine eindeutige Lösung der Knotengewichte (der Pagerank) und der Kantengewichte konvergiert.

Pagerank kann als wiederholte Matrix-Vektor-Multiplikation einer stochastischen Matrix mit den aktuellen Pagerank-Werten effizient implementiert werden [Hav99].

Der Pagerank ist heute einer der wichtigsten Faktoren des Rankings der Suchmaschine Google.

2.5 Graphen aus Punkten

In vielen Fällen ist eine Punktemenge V zum Beispiel aus einer Messung vorhanden, aber keine zugehörige Kantenmenge. Um auf einer solchen Menge mit Graphalgorithmen zu arbeiten, muss man also zunächst eine Kantenmenge definieren. Im Folgenden nehmen wir an, dass eine Distanzfunktion $d : V \times V \to \mathbb{R}^+$ gegeben ist.

Damit kann man nun einen Graphen konstruieren, welcher gemäß diesem Distanzmaß „ähnliche“ Punkte verbindet. Es gibt verschiedene Ansätze, welche Punkte verbunden werden sollen, wie zum Beispiel ein minimaler Spannbaum, der r-Graph, der kNN-Graph und der Graph der relativen Nachbarn.[Tou80, MVLH08]

Methoden zur Graphenkonstruktion

Der vollständige Graph

Eine sehr einfach Wahl ist der vollständige Graph, bei welchem die Beziehungen zwischen den Knoten nur durch die Ähnlichkeitsfunktion definiert ist.

Der r-Graph

Der r-Graph (In der Literatur teilweise auch ϵ-Graph genannt) hat einen Radius r, welcher definiert ob Punkte als benachbart gelten. Dann werden die Punkte v_i, v_j verbunden, wenn $d(v_i, v_j) \leq r$ ist.

r kann frei gewählt sein, oder ggf. aus den Daten berechnet werden, denkbar wäre zum Beispiel eine Funktion der minimalen, durchschnittlichen oder maximalen Distanz zwischen zwei Punkten als Grundlage zur Berechnung von r zu verwenden.

k nächste Nachbarn

Der kNN-Graph verbindet einen Punkt mit seinen k nächsten Nachbarn. Eine Variante ist der symmetrische (gegenseitige) kNN-Graph, welcher zwei Punkte nur dann verbindet, wenn beide jeweils den anderen in der Menge ihrer k nächsten Nachbarn haben.

[2] Im allgemeinen beeinflusst die Wahl der Startbedingung nur die Konvergenzrate, nicht die Lösung

Minimaler Spannbaum

Ein minimaler Spannbaum, wie in Abschnitt 2.2 definiert, ist auch ein möglicher Graph, den man aus einer Punktemenge generieren kann.

Gewichtung

Die Gewichtung der Kanten eines solchen Graphen kann ebenfalls unterschiedlich definiert werden. Die einfachsten Möglichkeiten sind:

- Binäre Gewichtung: Wenn zwei Punkte verbunden sind, ist das Gewicht 1, sonst 0.
- Das Gewicht ist (umgekehrt) proportional zur Distanz.
- Das Gewicht ist (umgekehrt) proportional zur Anzahl Nachbarn eines Knotens.

Ein typisches Beispiel für die Gewichtung einer Kante zwischen zwei Punkten durch die Distanz ist $d(x_i, x_j) := \exp(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2})$. Durch die Gaußglocke unterscheiden sich leicht unterschiedlich gewichtete Kanten zwischen Punkten, die sich nah sind, deutlich stärker voneinander als Kanten zwischen Punkten die weit auseinander liegen. Der Nachteil des Ansatzes ist z.B. beim vollständigem Graphen, dass die Matrix vollbesetzt ist.

Aus den anderen Möglichkeiten einen Graphen zu generieren ergeben sich meistens dünner besetzte Matrizen, mit welchen schneller gerechnet werden kann. Von Luxburg empfiehlt in [VL07] den k-nearest-neighbours Graphen als erste Wahl für die meisten Anwendungen, unter anderem weil er bei ungünstig gewählten Parametern weniger problematisch ist als andere Graphen.

Beispiele

Die aus den Punkten in Abb. 2.4 mit diesen Methoden generierten Graphen sind in den Grafiken 2.5, und 2.6 zu dargestellt Der vollständige Graph ist aus Gründen der Übersichtlichkeit nicht mit abgebildet.

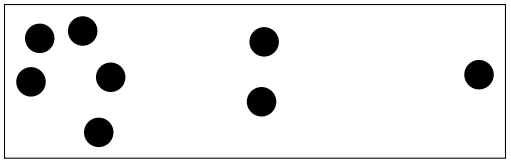


Abb. 2.4: Die Punkte, aus denen ein Graph erstellt werden soll

Bedeutung der Graphenkonstruktion

Wie man an den Beispielen erkennen kann, entstehen je nach Konstruktion Graphen, die sich stark voneinander unterscheiden. Der Einfluss der Graphenkonstruktion auf die

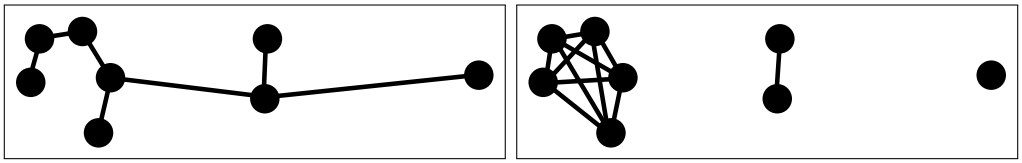


Abb. 2.5: Links: Ein minimaler Spannbaum der Punkte. Rechts: Ein r-Graph der Punkte.

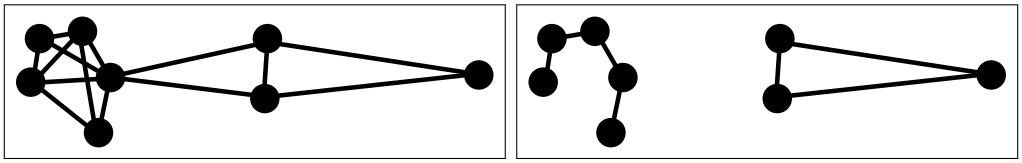


Abb. 2.6: Links: Ein kNN-Graph mit $k = 3$. Rechts: ein symmetrischer kNN-Graph mit $k = 3$.

Ergebnisse zum Beispiel von Cluster-Algorithmen wird in [VL07, Abschnitt 8.1] gezeigt und in [MVLH08] genauer untersucht. Dort wird aufgezeigt, dass die Wahl des Graphen einen großen Einfluss auf das Ergebnis haben kann.

Auch die Wahl der Parameter hat großen Einfluss auf das Ergebnis, gerade beim r-Graphen kann es schwierig sein einen Radius zu finden, welcher weder zu viele, noch zu wenige Punkte miteinander verbindet. Die kNN-Graphen sind dabei flexibler, da die Anzahl der Nachbarn nicht von der Distanz im Raum abhängt. Beim vollständigem Graphen und dem minimalem Spannbaum gibt es keine Parameter, welche die Konnektivität steuern. Dort beeinflusst nur die Berechnung der Kantengewichte das Ergebnis. Beim vollständigem Graphen sind die Kantengewichte sogar die einzige Information über die Lage der Punkte.

In den Beispielgraphen sieht man, dass völlig unterschiedliche Zusammenhangskomponenten entstehen, je nachdem mit welchem Algorithmus der Graph erzeugt wurde. Auch der Knotengrad unterscheidet sich deutlich.

2.6 Der Graph-Laplaceoperator

Der Graph-Laplace ist eine Diskretisierung des Laplaceoperators auf den Punkten eines Graphen. Das ist insbesondere interessant für Graphen, die Meshes (z.B. Triangulierungen eines Gebiets) darstellen, aber der Operator ist für alle einfachen Graphen definiert.

Während der Laplaceoperator im kontinuierlichem Raum eindeutig definiert ist, gibt es viele verschiedene Diskretisierungen des Operators, welche für verschiedene Zwecke unterschiedliche Vorteile besitzen, aber auch spezifische Nachteile haben, siehe [WMKG07].

Der zweidimensionale Finite Differenzen Laplace

Ein häufig verwendeter diskreter Laplaceoperator für reguläre Gitter ist der 5-Punkte-Stern des Finite Differenzen Laplaceoperators in zwei Dimensionen [Mit]. Dieser betrachtet Nachbarpunkte auf dem Gitter, addiert den Wert dieser vier Punkte und zieht dann

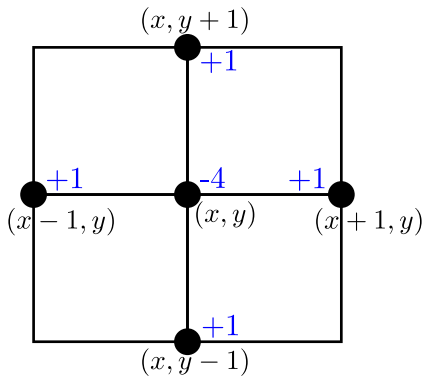


Abb. 2.7: Der 5-Punkte-Stern des finite Differenzen Laplaceoperators

den vierfachen Wert des mittleren Punkts ab.

Das heißt für eine Funktion $\phi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ gilt

$$(\Delta\phi)(x, y) \quad \mapsto \quad \phi(x-1, y) + \phi(x+1, y) + \phi(x, y-1) + \phi(x, y+1) - 4\phi(x, y)$$

Dieser diskrete Laplaceoperator arbeitet auf regulären Gittern und verwendet exakt diese 5 Punkte.

Ein allgemeiner Graph-Laplaceoperator

Wir definieren den nicht-normalisierten Graph-Laplace wie in [VL07] und verwenden auch die gleiche Notation. Die beiden normalisierten Graph-Laplaceoperatoren werden wir hier nicht betrachten, es sei aber darauf hingewiesen, dass alle drei Graph-Laplace sich sehr ähnlich sind, wenn der Grad der Knoten wenig voneinander abweicht. Sind diese Abweichungen größer, gibt es jedoch deutliche Unterschiede [VL07, 8.5].

Bemerkung: Der dort definierte Graph-Laplace L hat ein negatives Vorzeichen, im Gegensatz z.B. zum Finite Differenzen Laplaceoperator sind damit die Werte auf Diagonale positiv und die der Nachbarpunkte negativ.

Sei $G = (V, E)$ ein einfacher Graph mit nicht-negativen Kantengewichten, und $\mathbf{W}$ seine Adjazenzmatrix, welche an der Stelle i, j das Gewicht w_{ij} der Kante $e_{ij} = [i, j]$ speichert. Da der Graph ungerichtet ist, ist die Matrix $\mathbf{W}$ symmetrisch, d.h. $w_{ij} = w_{ji} \quad \forall i, j \in |V|$.

Definition 2.1. *Die Degree-Matrix* $\mathbf{D}$ *der Knotengrade ist eine Diagonalmatrix der Knotengrade:*

$$D_{ij} = \begin{cases} \text{grad}(v_i) & \text{wenn } i = j \\ 0 & \text{sonst} \end{cases} \tag{2.1}$$

Definition 2.2. *Der nicht normalisierte Graph-Laplace ist*

$$\mathbf{L} := \mathbf{D} - \mathbf{W}$$
$$\mathbf{L} \mathrel{\hat{=}} -\triangle$$

Bemerkung: Schleifen fließen nicht mit in den Graph-Laplace ein.

Dass Schleifen keinen Einfluss auf den Graph-Laplace haben ist an der Konstruktion der Einträge l_{ii} aus den w_{ii} und d_{ii} leicht zu erkennen:

$$l_{ii} := d_{ii} - w_{ii} = (\sum_{j=1}^{n} w_{ij}) - w_{ii} = (w_{i1} + \cdots + w_{ii} + \cdots + w_{in}) - w_{ii}$$

$$\Rightarrow l_{ii} = \sum_{j=1, j\neq i}^{n} w_{ij} \quad \forall i \in \{1, \ldots, n\}.$$

Daraus dass der Eintrag w_{ii} weg fällt folgt, dass eventuelle Schleifen die Matrix $\mathbf{L}$ nicht verändern.

Beispiel

In Abbildung 2.8 ist ein einfaches Beispiel mit 8 Knoten und zwei Zusammenhangskomponenten zu sehen. Die Kantengewichte für den ungewichteten Graphen nehmen wir als 1 an.

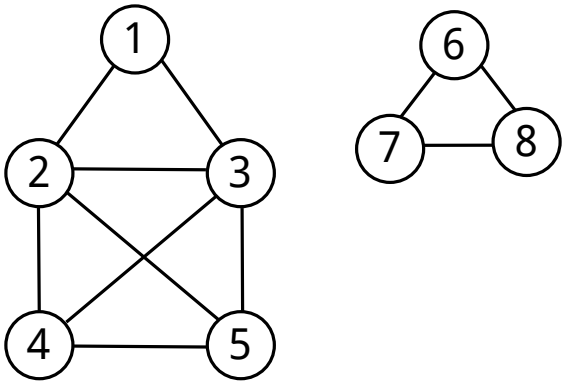


Abb. 2.8: Beispielgraph für den Graph-Laplace

Die Matrizen zum Graph in Abbildung 2.8 sind:

$$\mathbf{W} := \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \mathbf{D} := \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} \tag{2.2}$$

$$\mathbf{L} := \mathbf{D} - \mathbf{W} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix} \tag{2.3}$$

Eigenschaften des Graph-Laplace

Sei $\mathbf{L} \in \mathbb{R}^n \times \mathbb{R}^n$ eine solche Graph-Laplace Matrix. Dann gilt:

1. Alle Zeilen- und Spaltensummen der Matrix $\mathbf{L}$ sind 0.
2. Für jeden Vektor $\mathbf{f} \in \mathbb{R}^n$ gilt $\mathbf{f}^T\mathbf{L}\mathbf{f} = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$.
3. $\mathbf{L}$ ist symmetrisch und positiv semi-definit.
4. Der kleinste Eigenwert ist 0 und der zugehörige Eigenvektor ist der Eins-Vektor $\mathbb{1}$.
5. Die Eigenwerte von $\mathbf{L}$ sind $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ mit $\lambda_i \in \mathbb{R}$.

Beweis: Eigenschaften von $\mathbf{L}$
1) folgt daraus, dass D_{ii} gerade die i-te Zeilen-/Spaltensumme von $\mathbf{W}$ ist.

2) Sei $\mathbf{f} \in \mathbb{R}^n$. Dann gilt:

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \mathbf{f}^T(\mathbf{D}-\mathbf{W})\mathbf{f} = \mathbf{f}^T\mathbf{D}\mathbf{f} - \mathbf{f}^T\mathbf{W}\mathbf{f} \tag{2.4}$$

$$= \sum_{i,j=1}^{n} f_i D_{ij} f_j - \sum_{i,j=1}^{n} f_i w_{ij} f_j \tag{2.5}$$

$$= \sum_{i=1}^{n} f_i^2 D_{ii} - \sum_{i,j=1}^{n} f_i w_{ij} f_j \tag{2.6}$$

$$= \frac{1}{2}\left(2\sum_{i=1}^{n} f_i^2 D_{ii} - 2\sum_{i,j=1}^{n} f_i w_{ij} f_j\right) \tag{2.7}$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n} f_i^2 D_{ii} + \sum_{j=1}^{n} f_j^2 D_{jj} - 2\sum_{i,j=1}^{n} f_i w_{ij} f_j\right) \tag{2.8}$$

$$= \frac{1}{2}\left(\sum_{i,j=1}^{n} f_i^2 w_{ij} + \sum_{i,j=1}^{n} f_j^2 w_{ij} - 2\sum_{i,j=1}^{n} f_i w_{ij} f_j\right) \tag{2.9}$$

$$= \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \tag{2.10}$$

In 2.6 nutzen wir aus, dass $D_{ij} = 0\ \forall i \neq j$, da $\mathbf{D}$ eine Diagonalmatrix ist. Nachdem wir in 2.7 die Formel mit 2 multipliziert haben und im Ausgleich dafür außerhalb der Klammer durch 2 teilen, kann nun die Summe über D_{ii} in eine über i und eine über j aufgeteilt werden, da $\mathbf{D}$ symmetrisch ist. Im Schritt 2.9 setzen wir die Definition für D_{ii} in beide Terme ein und bekommen somit nach dem binomischem Lehrsatz gerade das Ergebnis in 2.10 heraus.

3) Die Symmetrie folgt direkt daraus, dass $\mathbf{W}$ und $\mathbf{D}$ symmetrisch sind. Aus 2) folgt $\mathbf{f}^T\mathbf{L}\mathbf{f} \geq 0$, daher ist $\mathbf{L}$ positiv semi-definit.

4) gilt, weil die Zeilensummen der Matrix 0 sind.

5) folgt daraus, dass die Matrix positiv semi-definit ist (also nur nicht-negative Eigenwerte besitzt) und wie in 4) gezeigt den Eigenwert 0 hat.

Die Eigenwerte des Graph-Laplace

Aus den Eigenwerten des Graph-Laplace lassen sich einige interessante Eigenschaften des Graphen ablesen.

Der Eigenwert 0

Definition 2.3. *Sei $A \subseteq \{1,\dots,n\}$ der Menge von Knoten-Indizes. Dann gibt der Indikatorvektor an, welche Indizes in A enthalten sind. Wir bezeichnen den i-ten Eintrag des Vektors als $\mathbb{1}_A^i$ und definieren den Indikatorvektor zu A als:*

$$\mathbb{1}_A^i := \begin{cases} 1 & \text{wenn } i \in A \\ 0 & \text{wenn } i \notin A \end{cases}$$

Satz 2.1. *Die Vielfachheit des Eigenwerts* 0 *entspricht der Anzahl der Zusammenhangskomponenten des Graphen, und die Indikatorvektoren* $\mathbb{1}_{A_i}$ *der ZHK* A_i *sind Eigenvektoren zu den Eigenwerten* 0*.*

Sei $\mathbf{f}$ ein Eigenvektor von $\mathbf{L}$ zum Eigenwert 0.
Dann gilt:

$$0 = \mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2 \tag{2.11}$$

Lemma 2.1. *In einer ZHK* A *gilt* $f_i = f_j$ *für alle Indizes* $i, j \in A$

Beweis: Für alle Indizes i, j in der gleichen ZHK gilt $f_i = f_j$

Da alle w_{ij} der Kanten größer als 0 sind, folgt für zwei direkt verbundene Knoten v_i, v_j aus 2.11, dass $f_i = f_j$ sein muss. Für einen Pfad $v_1, v_2, \ldots, v_n$ gilt für jedes Segment $[v_i, v_j]$, dass $f_i = f_j$. Nach Definition sind alle Knoten einer Zusammenhangskomponente durch Pfade miteinander verbunden, also gilt $f_i = f_j \quad \forall i, j \in A$.

□

Bemerkung: Die umgekehrte Richtung, die ZHK aus den Eigenvektoren abzulesen, ist nicht direkt möglich, da die Basis nicht unbedingt aus den Indikatorvektoren bestehen muss. Die Basisvektoren haben allerdings die Form $\sum_{i=1}^{k} a_i \mathbb{1}_{A_i}$ mit $a_i \in \mathbb{R}$, wodurch sie die Information über die ZHK trotzdem noch enthalten, siehe [VL07, Abschnitt 8.2].

Lemma 2.2. *Die Vielfachheit des Eigenwerts* 0 *entspricht der Anzahl der Zusammenhangskomponenten des Graphen.*

Beweis:
Ohne Beschränkung der Allgemeinheit können wir annehmen, dass die Zeilen des Graphen nach Zusammenhangskomponenten gruppiert sind, da die Sortierung der Knotenliste beliebig gewählt werden kann.

Durch diese Sortierung entsteht eine Blockmatrix, da es zwischen unterschiedlichen ZHK nach Definition keine Kanten gibt, und daher gilt

$$L_{ij} = 0 \quad \forall i \in A, j \notin A.$$

Die Blöcke der Matrix sind selber Graph-Laplace-Matrizen für die Teilgraphen, welche genau die entsprechende ZHK enthalten, und haben daher den Eigenwert 0 mit Eigenvektor $\mathbb{1}$.

Aus den beiden Lemmata folgt Satz 2.1.

□

In Formel 2.3 ist die Blockstruktur der Laplace-Matrix für den Graphen aus Abbildung 2.8 zu sehen.

Graph Cuts

Definition 2.4. *Das Gewicht zwischen zwei Knotenmengen A und B ist definiert als*

$$W(A, B) := \sum_{i \in A, j \in B} w_{ij}. \tag{2.12}$$

Eine Anwendung des nicht-normalisierten Graph-Laplace ist, einen Graph Cut zu finden mit welchem sich ein zusammenhängender Graph in mehrere Cluster zerlegen lässt. Die Funktion

$$\text{cut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} W(A_i, \bar{A}_i) \tag{2.13}$$

definiert dabei ein Maß für die Qualität eines Graph-Cuts, indem sie misst wie stark das Gewicht der Kanten zwischen den Teilgraphen ist.

Definition 2.5. *Ein Graph-Cut heißt* minimal*, wenn das Gewicht* $\text{cut}(A_1, \ldots, A_k)$ *minimal ist.*

Einen minimalen Graph-Cut zu finden ist in polynomieller Zeit möglich, aber wenn die Anzahl der Knoten in A und $\bar{A}$ ausgeglichen sein soll bezüglich einer Balance-Funktion, wird das Problem NP-hart[WW93]. Ohne eine solche Balance-Funktion ist die optimale Lösung für das Problem aber häufig, dass ein einzelner Knoten vom Graphen getrennt wird, was für Anwendungen wie Clustering nicht sinnvoll ist.

Einer der balancierten Graph-Cuts aus [VL07] ist RatioCut:

$$\text{RatioCut}(A_1, \ldots, A_k) := \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \bar{A}_i)}{|A_i|} \tag{2.14}$$

Bei RatioCut wird die Anzahl Knoten in den Clustern balanciert. Eine Approximation des optimalen Schnitts für zwei Knotenmengen kann mit dem Cheeger-Cut [BH09] über den zweiten Eigenwert des Graph-Laplace berechnet werden. Die Vorzeichen der Einträge des zugehörigen Eigenvektors geben dabei die Cluster an.

Beispiel

Für das Beispiel aus Abbildung 2.8 mit dem zugehörigen Graph-Laplace 2.3 haben wir:

- Eigenwerte: $0, 0, 2, 3, 3, 4, 5, 5$
- Eigenvektor zum Eigenwert 2: $(0.55, -\epsilon, -0.45, -0.28, -\epsilon, 0, 0, 0)$

- Clustering nach Vorzeichen: $(1, 2, 2, 2, 2, 1, 1, 1)$

Die mit ϵ bezeichneten Werte sind negativ und liegen nahe 0, müssen aber für das Clustering-Ergebnis, anders als die Werte die exakt 0 sind, auch als negative Werte eingeordnet werden.

In Abbildung 2.9 ist das Ergebnis des Cuts zu sehen.

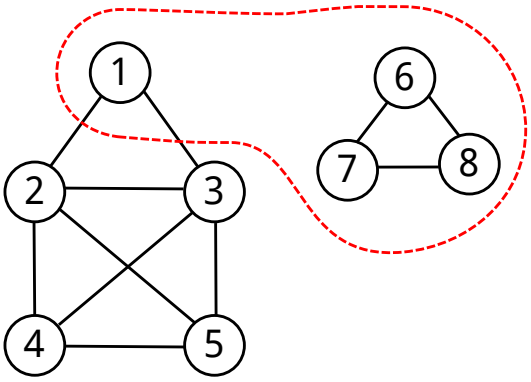


Abb. 2.9: Der mit dem zweiten Eigenvektor berechnete Graph-Cut des Beispielgraphen

Beziehung des Graph-Laplace zum Finite-Differenzen-Laplace

Berechnet man den Graph-Laplace auf einem uniformen Rechtecksgitter wie in Grafik 2.7, kommt gerade $L = -\Delta_{fd}$ heraus. Damit kann diese Definition des Graph-Laplaceoperators als eine Verallgemeinerung des Finite-Differenzen-Laplaceoperators gesehen werden.

3 Discrete Exterior Calculus (DEC)

3.1 Einleitung

Nachdem wir uns mit Graphen als allgemeinere Form von Gittern beschäftigt haben, und mit dem Graph-Laplace einen Laplaceoperator definiert haben, der auf jeder Graph-Struktur anwendbar ist, stellt sich nun die Frage, welche Graphen für verschiedene Berechnungen sinnvoll sind. Wir haben in Abschnitt 2.5 gesehen, dass sich Graphen aus einer Menge von Punkten generieren lassen, indem man in geeigneter Weise aus den geometrischen Eigenschaften der Punkte eine Kantenmenge erzeugt.

Nun betrachten wir das Problem von der anderen Seite. Für viele Anwendungen ist ein Graph bereits gegeben, da die Punkte aus einer bekannten Struktur, wie zum Beispiel einem 3D-Modell stammen. Möchte man zum Beispiel die Wärmeleitungsgleichung lösen, hat man normalerweise ein Objekt gegeben, auf dem sich die Wärme verteilen soll. Für die numerischen Berechnungen liegt dieses Objekt also schon als Mesh vor.

Da das Objekt bereits als Mesh gegeben ist sind die Kanten der Triangulierung vorhanden. Intuitiv erscheint es sinnvoll die Werte mit Hilfe dieser Kanten zu berechnen, statt ein neues Gitter zu definieren um die Werte dort zu berechnen. Die Kanten des Modells zu verwenden hat auch Vorteile gegenüber einem neuem Gitter, da sie die Oberfläche eindeutig definieren und zum Beispiel eine Ausbreitung der Wärme zwischen Punkten, die sich geometrisch nah, aber z.B. auf einer gekrümmten Oberfläche selber weit von einander entfernt sind, vermieden wird.

Konzepte wie adaptive Gitter, welche detailreichere Teile des Meshs mit einem feinerem Gitter berechnen, sind nicht mehr explizit nötig. Das Mesh selber definiert das Gitter und höher aufgelöste Teile des Meshs weisen daher auch bei der Berechnung eine höhere Auflösung auf.

Eine allgemeine Methode dies umzusetzen heißt „Discrete Exterior Calculus“ (DEC).

Was ist der Discrete Exterior Calculus?

Der „Discrete Exterior Calculus“ ist eine Methode Operatoren auf einem Mesh zu definieren, sodass dort partielle Differentialgleichungen direkt gelöst werden können. Dabei sind die konkreten Werte, wie die Koordinaten der Punkte im Mesh und die Werte auf diesen Punkten, von den Relationen der verwendeten Objekte getrennt. Dadurch kann mit den abstrakten Objekten exakt gerechnet werden, und ein Diskretisierungsfehler entsteht erst dann, wenn die berechneten Objekte auf diskretisierte Werte angewendet werden.

Eine der Stärken des DEC ist, dass physikalische Erhaltungsgrößen und Integralsätze genau erfüllt werden, da alle verwendeten Größen und Operatoren direkt auf dem Mesh definiert werden. Während bei vielen anderen Methoden die Integralsätze asymptotisch bei höherer Auflösung einen immer kleineren Fehler aufweisen, hängt im DEC nur die Genauigkeit der berechneten Größen von der Auflösung ab, nicht aber die Genauigkeit der Integralsätze.

Um dies umzusetzen sind Elemente wie Punkte, Kanten, Flächen und Tetraeder als abstrakte Objekte in einem Graphen definiert. Die numerischen Werte wie Flächeninhalt, Länge, Wert einer Funktion auf einem Punkt werden unabhängig davon gespeichert und erst bei Bedarf auf diese Objekte angewendet. Dadurch sind Operationen wie differenzieren oder die Anwendung eines Laplaceoperators als Operationen auf Graphen definiert, wodurch sie direkt auf dem verwendeten Mesh arbeiten und daher exakt sind.

Bei der konkreten Berechnung erfolgen diese Operatoren dann als Multiplikationen von Matrizen und Vektoren, welche die Relationen der Objekte abbilden, wie zum Beispiel Adjazenzmatrizen.

Grundlegende Arbeiten zum Thema

Eine der umfangreichsten Arbeiten zur Theorie des Discrete Exterior Calculus ist die Dissertation von Anil N. Hirani[Hir03] mit dem Titel „Discrete Exterior Calculus“, welcher wir im Folgenden für die theoretischen Definitionen folgen werden.

Bei der Implementierung folgen wir den Arbeiten „Building Your Own DEC at Home“ von Elcott, Schröder[ES05] und „Discrete Differential Forms for Computational Modeling“ von Desbrun, Kanso, Tong[DKT08].

3.2 Differentialformen

Da es im Discrete Exterior Calculus um eine Diskretisierung von Differentialformen handelt, wird hier zunächst das allgemeine Konzept einer Differentialform eingeführt, um diese dann später in diskreter Form im DEC zu verwenden.

Eine Differentialform ist ein Ansatz die Integration von Funktionen über (mehrdimensionale) Intervalle von Koordinaten unabhängig zu machen. So ist zum Beispiel der Wert eines Linienintegrals unabhängig davon, ob die Kurve über der integriert wird, als Ganzes im Raum verschoben wird.

In [DKT08] wird eine intuitive Definition gegeben. Dort wird $f = \frac{dF}{dx}$ als Ableitung einer Funktion F definiert und in Bezug auf die Integration

$$\int_a^b dF = \int_a^b f(x)\mathrm{dx} = F(b) - F(a) \tag{3.1}$$

ist damit der Integrand $f(x)dx$ eine differentielle 1-Form. Dabei bezieht sich der Begriff „k-Form“ auf die Anzahl der Dimensionen über die integriert wird.

Das gleiche gilt auch für eine Kurve in mehr Dimensionen wie zum Beispiel eine Funktion $G(x, y, z)$ im dreidimensionalen Raum, für welche die 1-Form

$$dG = \frac{\partial G}{\partial x}\mathrm{dx} + \frac{\partial G}{\partial y}\mathrm{dy} + \frac{\partial G}{\partial z}\mathrm{dz} \tag{3.2}$$

über jede Kurve im $\mathbb{R}^3$ integriert werden kann.

Analog ist eine 2-Form ein Integrand, welcher über jeder Fläche integriert werden kann und allgemein kann eine k-Form über einer k-dimensionalem Mannigfaltigkeit integriert werden.

Formale Definition

Die formale Definition einer k-Form ist:

Sei $\mathbb{R}^n$ der umgebende Raum und $\mathcal{M} \subset \mathbb{R}^n$ eine offene Teilmenge des Raums, eine sogenannte n-Mannigfaltigkeit. Dann wird an jedem Punkt $x \in \mathcal{M}$ der Vektorraum $T_x\mathcal{M}$ aus den Tangentenvektoren der Mannigfaltigkeit am Punkt x aufgespannt.

Eine k-Form ω^k ist dann ein anti-symmetrisches Tensorfeld auf $\mathcal{M}$, welches ausgewertet auf einem Punkt $x \in \mathcal{M}$ eine Abbildung ergibt, welche k Tangentenvektoren auf eine reelle Zahl abbildet:

$$\omega^k : \quad T_x\mathcal{M} \times \cdots \times T_x\mathcal{M} \to \mathbb{R} \tag{3.3}$$

Um mit Differentialformen auf diskreten Objekten zu rechnen, müssen die Mannigfaltigkeit und die Differentialform selber als diskrete Werte gespeichert werden. Dazu definieren wir nun eine Diskretisierung dieser Größen.

3.3 Diskretisierung

Um auf diskreten Modellen zu rechnen wird zunächst definiert, wie man eine Mannigfaltigkeit durch ein 3D-Modell diskret approximieren kann um danach Differentialformen, die auf diskretisierten Mannigfaltigkeiten arbeiten, zu definieren.

Objekte in 3D-Meshes

Die grundlegenden Objekte um eine Mannigfaltigkeit im DEC abzuspeichern sind die gleichen wie bei üblichen Mesh-Formaten von 3D-Modellen.

- Eine Menge von Punkten im Raum (Vertices), welche als eine Liste von dreidimensionalen Vektoren gespeichert wird. Ein Vertex kann eindeutig über seinen Index in der Liste identifiziert werden.

- Geordnete Listen von diskreten Objekten (Punkte, Kanten, Flächen, ...), welche durch die Vertices an ihren Eckpunkten definiert sind.

Typischen 3D-Formate speichern allerdings nur die Objekte der höchsten Dimension, da diese ausreichen um ein 3D-Modell zu beschreiben und alle weiteren Informationen implizit enthalten.
Beispiel: Ein Format, welches Dreieckmeshes speichert, besitzt eine Liste von Dreiecken, während die Kanten nur implizit aus den Dreiecken rekonstruierbar sind. Die Orientierung der Flächen ist nur durch die Reihenfolge der Vertices in der Definition des Dreiecks gegeben.

Ein solches Format ist das Wavefront „OBJ"-Format. Ein Beispiel ist in Abschnitt 3.10 zu sehen.

Objekte im DEC

Im Gegensatz zu diesen Formaten benötigt eine DEC-Implementierung Listen mit den Objekten aller Dimensionen kleiner oder gleich der Dimension der Mannigfaltigkeit, und speichert daher für jede Dimension k eine eigene Liste von k-dimensionalen Objekten ab.

Definition 3.1. *Ein* Vertex *ein d-dimensionaler Vektor im Raum $\mathbb{R}^d$, welcher als d-Tupel von reellen Zahlen gespeichert wird. Die Vertices können eindeutig mit v_i bezeichnet werden, wobei i der Index in der Vertexliste ist.*

Beispiel: $(0, 0, 1) \in \mathbb{R}^3$ ist ein Vertex im dreidimensionalem Raum.

Definition 3.2. *Ein k-Simplex ist die konvexe Hülle von $k + 1$ Vertices. Damit es k-dimensional ist, fordern wir weiterhin, dass die Vertices des Simplex linear unabhängig sind. Eine eindeutige Orientierung erhält ein Simplex durch die Reihenfolge der Vertices, welche eine Umlaufrichtung festlegt. Wir bezeichnen ein k-Simplex mit σ^k.*

Bemerkung: Ein orientiertes Simplex ist durch die in Umlaufrichtung geordnete Menge seiner Vertices eindeutig definiert, allerdings ergeben alle geraden Permutationen dieser Menge das gleiche Simplex.
Beispiel: $\sigma^2 = [v_1, v_2, v_3]$ definiert ein 2-Simplex (Dreieck) mit den Eckpunkten v_1, v_2, v_3 und der Umlaufrichtung $v_1 \to v_2 \to v_3 \to v_1$.

Wir definieren für orientierte Simplices weiterhin einen *Minus-Operator*, welcher die Orientierung umkehrt. Eine einfache Implementierung ist die Reihenfolge der Vertex-Indices im Tupel, welches ein Simplex definiert, umzukehren:

$$-\sigma^k = -[v_1, v_2, \ldots, v_{k+1}] = [v_{k+1}, v_k, \ldots, v_1] \tag{3.4}$$

Für einige Simplices verwenden wir bereits vorhandene geometrische Bezeichnungen:

- **0-Simplex:** Vertex oder Punkt
- **1-Simplex:** Kante

- **2-Simplex:** Dreieck
- **3-Simplex:** Tetraeder

Bemerkung: Der Begriff Vertex ist doppeldeutig. Häufig steht Vertex für einen Eckpunkt eines Simplex, welcher eine Koordinate im Raum ist. Ein Vertex als 0-Simplex hingegen repräsentiert einen Punkt im Raum als Objekt, welches genau einen Vertex (Eckpunkt) hat. Das 1-Tupel, welches ein 0-Simplex beschreibt, enthält dabei genau den Index, den das Vertex in der Liste der Vertices hat. Insbesondere hat die Liste der Vertices normalerweise die gleiche Kardinalität wie die Liste der 0-Simplices, aber im allgemeinen[1] nicht die gleichen Indizes.

Beispiel: Gerade Permutationen von Simplices

Als Beispiel, dass gerade Permutationen der Reihenfolge der Vertices eines Simplex die gleiche Orientierung beschreiben, betrachten wir ein Dreieck $[v_1, v_2, v_3]$. In einem Umlauf werden die Kanten $[v_1, v_2], [v_2, v_3]$ und $[v_3, v_1]$ besucht. Betrachtet man die gerade Permutation $[v_2, v_3, v_1]$, in welcher die Eckpunkte v_1, v_2 und dann in Folge v_1, v_3 getauscht wurden, besucht ein Umlauf die Kanten $[v_2, v_3]$, $[v_3, v_1]$ und dann $[v_1, v_2]$.

Man sieht, dass die gerade Permutation verändert hat an welchem Punkt des Dreiecks der Umlauf beginnt, nicht jedoch in welcher Richtung die einzelnen Kanten durchlaufen werden.

Faces

Die $(k-1)$-Simplices, welche auf dem Rand eines k-Simplex liegen werden seine *Faces* genannt. Der Ursprung des Begriffs sind die Dreiecke auf dem Rand eines Tetraeders, aber er wird hier allgemeiner verwendet für Objekte niedrigerer Dimension, welche auf dem Rand eines Objekts liegen. Ist keine explizite Dimension angegeben ist mit einem Face eines k-Simplex stets ein $(k-1)$-Simplex auf dem Rand des k-Simplex gemeint.

Da die Faces eines Simplex mit einer eindeutigen Orientierung in der Liste der $(k-1)$-Simplices abgespeichert sind, hat das Face eine Orientierung, welche sich mit der Orientierung des k-Simplex vergleichen lässt.

Die relative Orientierung der Faces eines Simplex ist dadurch gegeben, ob die Umlaufrichtung des gespeicherten $(k-1)$-Simplex mit der Umlaufrichtung des durch das k-Simplex erzeugten $(k-1)$-Faces übereinstimmt.

Dies stimmt bis auf gerade Permutationen auch mit der Definition des Minus-Operators für Simplices überein, welcher das negativ induzierte Face in eine Permutation des gespeicherten umwandelt.

$$\operatorname{sign}(\sigma^k, \sigma^{k-1}) := \begin{cases} +1 & \text{bei gleicher Orientierung} \\ -1 & \text{sonst} \end{cases} \tag{3.5}$$

[1] Tatsächlich ist die Indexierung der beiden Listen bei manchen Methoden ein Mesh einzulesen komplett unterschiedlich

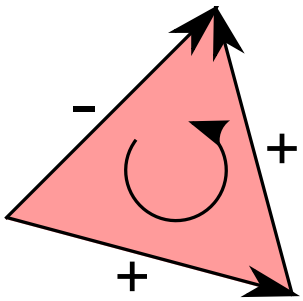


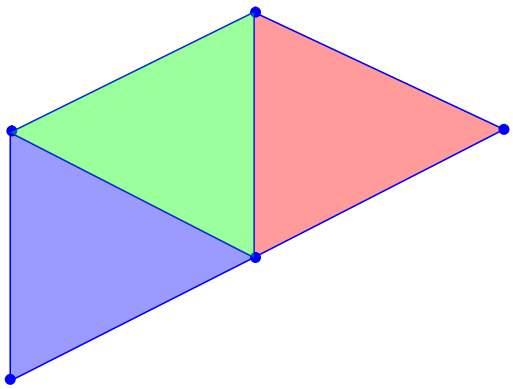
Abb. 3.1: Die Kanten haben eine negative Orientierung in Bezug auf ein Dreieck, wenn ihre Richtung nicht mit der Umlaufrichtung des Dreiecks übereinstimmt.

Wir können die Orientierung von zwei Simplices genau dann vergleichen, wenn sie entweder in der gleichen (Hyper-)Ebene liegen, oder ein gemeinsames Face haben. Im zweiten Fall sind sie genau dann gleich orientiert, wenn das gemeinsame Face von einem der Simplices positiv und vom anderem negativ orientiert wird.

Simplex-Komplex

Ein Simplex-Komplex K ist eine Menge von Simplices, für die gilt:

- Wenn ein k-Simplex (mit $k \geq 1$) im Komplex enthalten ist, sind auch alle seine $(k-1)$-Faces enthalten.
- Zwei Simplices schneiden sich entweder nicht, oder in einem gemeinsamen Face.



3 2-Simplices (Dreiecke)
7 1-Simplices (Kanten)
5 0-Simplices (Punkte)

Abb. 3.2: Beispiel für einen 2D-Simplex-Komplex

Die Menge aller Simplices im Komplex mit der Dimension k bezeichnen wir mit K^k. Die Menge der Simplices mit Dimension kleiner oder gleich k nennen wir das k-Skelett von K und bezeichnen sie mit $K^{(k)}$.

Definition 3.3. *Eine k-Chain ist eine Abbildung $c^k : K^k \to \mathbb{R}$, die jedem Simplex einen Wert zuweist. So kann eine k-Chain als Linearkombination aus k-Simplices gesehen werden (vgl. [Hir03, Def. 3.2.1]):*

$$c = \bigcup_{\sigma^k \in K^k} c(\sigma^k) \cdot \sigma^k \tag{3.6}$$

Wir bezeichnen den Raum der k-Chains in einem Komplex K mit $\mathcal{C}_k(K)$.

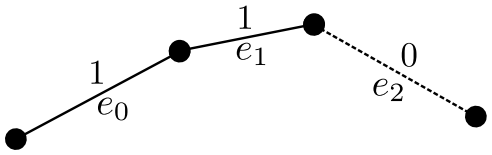


Abb. 3.3: Beispiel: Eine 1-Chain, welche aus den Kanten e_0 und e_1 Gewicht 1 zuweist, und die Kante e_2 mit 0 gewichtet.

Mit Hilfe von k-Chains ist es uns zum Beispiel möglich nur über einen Teil der diskreten Mannigfaltigkeit zu integrieren, indem der gewünschte Teil die Koeffizienten 1 hat und der Rest der Mannigfaltigkeit mit 0 gewichtet wird.

Definition 3.4. *Eine k-Co-Chain ist eine Abbildung $\omega^k : \mathcal{C}_k \to \mathbb{R}$, die jeder k-Chain eine reelle Zahl zuweist, und somit einer diskreten Differentialform entspricht. Wir bezeichnen den Raum der k-Co-Chains in einem Komplex K mit $\Omega^k(K)$*

Beispiel: Eine 1-Chain c^1 enthält die Gewichte für einen diskreten Kantenzug, welcher die Diskretisierung einer Kurve ist. Eine 1-Co-Chain bildet c^1 auf eine reelle Zahl ab, sodass die Anwendung der Co-Chain auf c^1 gerade der Integration der zugehörigen Differentialform über der Kurve entspricht.

Operatoren im DEC

Es gibt zwei Formen von Operatoren, die uns interessieren:

- Operatoren auf den Objekten, über die wir integrieren. Zum Beispiel der Randoperator, der eine Fläche auf ihren Rand abbildet.
- Operatoren auf den Differentialformen, die wir integrieren, wie z.B. der Differentialoperator.

Im DEC bildet die erste Form Operatoren k-Chains auf l-Chains ab, und die zweite k-Cochains auf l-Cochains, für $l, k \leq n$. In Abschnitt 3.5 werden einige wichtige diskrete Operatoren definiert.

3.4 Speicherung der Objekte

Im DEC gibt es drei verschiedene Kategorien von Objekten, die dauerhaft gespeichert werden, und zwei Arten, die beim Arbeiten mit dem DEC verwendet werden.

Aus einem gegebenem Mesh werden die folgenden Objekte generiert und dauerhaft gespeichert:

- **Vertices**
- **Simplices**
- **Operatoren**

Objekte, die beim Rechnen mit dem DEC verwendet werden aber nicht dauerhaft abgespeichert werden:

- **Chains** (Linearkombinationen von Simplices)
- **Co-Chains** (diskrete Differentialformen)

Vertices

Die Vertices werden als eine Liste von d-dimensionalen Vektoren gespeichert, wobei d die Dimension des Raums ist, in dem die Vertices liegen. Jeder Vertex wird durch seinen Index in der Vertexliste eindeutig identifiziert, da sich die Menge der Vertices eines Modells nicht ändert.

Simplices

Für $k = 0, \ldots, n$ werden die k-Simplices als geordnete $k+1$-Tupel von Vertex-Indices in einer Liste der Länge $|K^k|$ gespeichert. Durch die Reihenfolge ist die Orientierung des Simplex gegeben. Ein Simplex wird durch seine Dimension k und den Index in der Liste der k-Simplices eindeutig definiert, und kann z.B. als $(k, i) \in \{0, \ldots, n\} \times |K^k|$ geschrieben werden.

Da eine Änderung der Reihenfolge der Vertices nur die Orientierung des Simplex (und seiner Faces) ändert und wir die Simplices mit Vorzeichen schreiben können, ist eine einfache Art der Speicherung die Simplices der höchsten Dimension als positiv orientiert anzunehmen und dann die Faces rekursiv zu generieren. Wenn ein $(k-1)$-Face bereits der Liste der $(k-1)$-Simplices hinzugefügt wurde, wird nur gespeichert ob das Vorzeichen des generierten Face positiv oder negativ in Bezug auf das k-Simplex ist.

Chains und Co-Chains

Da eine k-(Co-)Chain eine Linearkombination von Werten zu k-Simplices ist, wird sie als Vektor $\boldsymbol{\sigma}^{\mathbf{k}}$ bzw. $\boldsymbol{\omega}^{\mathbf{k}}$ aus $\mathbb{R}^{|K^k|}$ gespeichert, wobei die Komponenten des Vektors den jeweiligen Simplices der Dimension k zugeordnet sind.

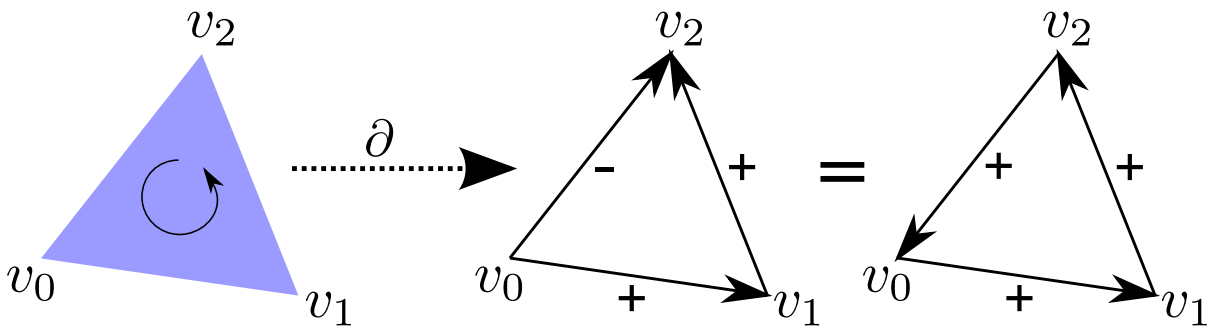


Abb. 3.4: Der Boundary-Operator ∂ bildet ein Dreieck auf seine orientierten Randkanten ab.

Die Auswertung einer k-Co-Chain auf einer k-Chain ist bei der Speicherung als Vektoren ein Skalarprodukt:

$$\omega^k(c^k) := \left\langle \boldsymbol{\omega}^{\mathbf{k}}, \mathbf{c}^{\mathbf{k}} \right\rangle = \left(\boldsymbol{\omega}^{\mathbf{k}}\right)^{\mathbf{T}} \boldsymbol{\sigma}^{\mathbf{k}} \tag{3.7}$$

Operatoren
Ein linearer Operator O, der eine k-(Co-)Chain auf eine ℓ-(Co-)Chain abbildet, wird als $\mathbb{R}^{|K^\ell|\times|K^k|}$ Matrix gespeichert, sodass die Anwendung auf eine (Co-)Chain eine Matrix-Vektor-Multiplikation ist:

$$\begin{aligned} \mathbf{O} : \mathbb{R}^{|K^k|} &\to \mathbb{R}^{|K^l|} \\ \mathbf{O}\boldsymbol{\omega}_{\mathbf{1}}^{\mathbf{k}} &= \boldsymbol{\omega}_{\mathbf{2}}^{\boldsymbol{\ell}} \text{ bzw.} \\ \mathbf{O}\boldsymbol{\sigma}_{\mathbf{1}}^{\mathbf{k}} &= \boldsymbol{\sigma}_{\mathbf{2}}^{\boldsymbol{\ell}} \end{aligned}$$

3.5 Wichtige Operatoren

Boundary-Operator für Simplices

Der Boundary-Operator für k-Simplices $\partial_k : K^k \to K^{k-1}$ ist nach [ES05, 3.1.3] definiert als:

$$\partial_k[v_0, \ldots, v_k] := \bigcup_{j=0}^{k} (-1)^k \{[v_0, \ldots, \hat{v}_j, \ldots, v_k]\} \tag{3.8}$$

wobei $\hat{v}_j$ heißt, dass gerade das j-te Vertex entfernt wurde um ein $(k-1)$-Face des Simplex zu erhalten.

In Grafik 3.4 ist ein Beispiel für die Randabbildung zu sehen. Ein Dreieck (2-Simplex) $[v_0, v_1, v_2]$ wird auf die Menge der Kanten (1-Simplices) auf seinem Rand abgebildet, indem

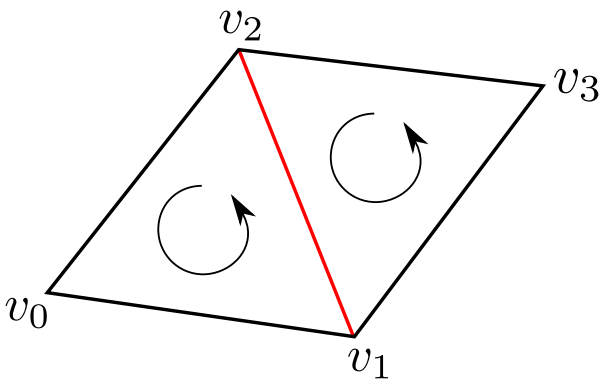


Abb. 3.5: Die Orientierung der mittleren Kante kann nur mit der Orientierung eines der beiden Dreiecke übereinstimmen

je Kante ein Vertex des Dreiecks entfernt wurde.

Da die entstehende Kante $[v_0, v_2]$ entgegen der Durchlaufrichtung des Dreiecks ($v_0 \to v_1 \to v_2 \to v_0$) orientiert ist, hat sie ein negatives Vorzeichen. Wie in der Definition eines k-Simplex beschrieben entspricht dem Simplex mit negativem Vorzeichen ein umgekehrt orientiertes Simplex mit den gleichen Punkten mit positivem Vorzeichen.

Bemerkung: Negative Orientierung lässt sich nicht vermeiden. Wenn ein Simplex σ^{k-1} das Face von zwei k-Simplices σ_1^k, σ_2^k ist, dann muss es in Bezug auf eines der beiden Simplices negativ orientiert sein. In Grafik 3.5 ist ein 2D-Beispiel zu sehen, bei dem die innere Kante nicht in Bezug auf beide Dreiecke positiv orientiert sein kann.

Bei der Implementierung gibt es zwei Möglichkeiten damit umzugehen:

- In Dreieckmeshes können Halbkanten verwendet werden, was heißt, dass die gleiche Kante in zwei verschiedenen Orientierungen gespeichert wird. Dieser Ansatz ist aber für höherdimensionale Meshes nicht sinnvoll.
- Das Vorzeichen eines k-Simplex in Bezug auf ein $k+1$-Simplex wird explizit gespeichert und das Vorzeichen zwischen Simplices mit einem größerem Dimensionsunterschied kann rekursiv berechnet werden. Dies ist im DEC durch das Vorzeichen im Randoperator gegeben.

Der Boundary-Operator für Chains

Aus der Linearität der Abbildung folgt, dass der Operator auch für k-Chains definiert ist [DKT08]:

$$\partial_k : C^k \to C^{k+1} \tag{3.9}$$

$$\partial_k(c) = \partial_k \bigcup_{\sigma^k \in K^k} c(\sigma^k) = \bigcup_{\sigma^k \in K^k} \partial_k c(\sigma^k) \tag{3.10}$$

Der Boundary-Operator als Matrix

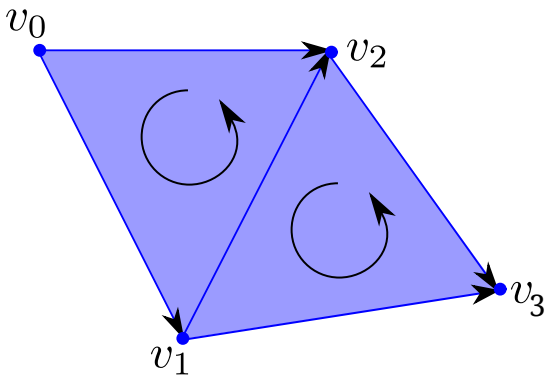
Diese lineare Abbildung von k-Chains auf $(k-1)$-Chains definiert eine $|K^{k-1}| \times |K^k|$ Matrix $\boldsymbol{\partial}_k$.

Dabei gilt für die Matrix:

- $\partial_k^{ij} = 1$, wenn σ_i^{k-1} ein Face von σ_j^k ist
- $\partial_k^{ij} = -1$, wenn $-\sigma_i^{k-1}$ ein Face von σ_j^k ist
- 0 sonst

Dabei haben die Simplices σ_i^k die Orientierung, wie sie in der Liste der k-Simplices gespeichert sind.

Beispiel



$$
\begin{aligned}
F =& \{[v_0, v_1, v_2], [v_1, v_3, v_2]\} \\
E =& \{[v_0, v_1], [v_0, v_2], [v_1, v_2], [v_1, v_3], [v_2, v_3]\} \\
V =& \{[v_0], [v_1], [v_2], [v_3]\}
\end{aligned}
$$

$$
\boldsymbol{\partial}_2 = \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}, \boldsymbol{\partial}_1 = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}
$$

Man sieht am Beispiel des Boundary-Operators $\boldsymbol{\partial}_2$, welcher Dreiecke auf Kanten abbildet, dass sich die inneren Kanten bei einer 2-Chain $(1, 1)$, die aus beiden Dreiecken besteht, gerade aufheben. Damit enthält das Ergebnis nur die Randkanten.

Der Coboundary-Operator

Analog wird der Coboundary-Operator $\mathbf{d}_k : C^k \to C^{k+1}$ definiert, der k-Chains auf die $k+1$-Chains abbildet und der adjungierte Operator zu ∂_{k+1} ist.

Da das Skalarprodukt $< \omega^k, c^k >$ eine Bilinearform ist, gilt:

$$\left\langle \mathbf{d}^k \boldsymbol{\omega}^k, \mathbf{c}^{k+1} \right\rangle = \left\langle \boldsymbol{\omega}^k, \boldsymbol{\partial}_{k+1} \mathbf{c}^{k+1} \right\rangle \tag{3.11}$$

und daher ergibt sich der Coboundary-Operator für k-Simplices als die transponierte Matrix des Boundary-Operators für $(k+1)$-Simplices: $\mathbf{d}_k = \boldsymbol{\partial}_{k+1}^T$.

Bemerkung: Dadurch, dass der Rand eines k-Simplex ein geschlossener Zyklus von $(k-1)$-Simplices ist, hat er selber keinen Rand und es ergibt sich gerade $\partial_{k-1}\partial_k = 0$.

Chain- / Cochain-Komplex

Die Boundary- und Coboundary Operatoren definieren den Chain-/Cochain-Komplex:

$$0 \longrightarrow C^n \xrightarrow{\partial_n} C^{n-1} \xrightarrow{\partial_{n-1}} \dots C^0 \longrightarrow 0$$

$$0 \longleftarrow C^n \xleftarrow[\mathbf{d}_{n-1}]{} C^{n-1} \xleftarrow[\mathbf{d}_{n-2}]{} \dots C^0 \longleftarrow 0$$

Dabei sind $\boldsymbol{\partial}_0$ und $\mathbf{d}_{n+1}$ als $\mathbf{0}$ definiert, was konsistent mit den Operatoren für Differentialformen ist. Dies lässt sich auch leicht in der Definition der Boundary-Operator-Matrix erkennen.

Der Differentialoperator

Mit Hilfe des Randoperators können wir nun den Satz von Stokes verwenden um den Differentialoperator zu definieren:

$$\mathbf{d} : \Omega^k(K) \to \Omega^{k+1}(K)$$

Sei $\boldsymbol{\partial}$ der Randoperator für diskrete k-Formen, σ eine k-Chain und ω eine k-Co-Chain. Dann gilt:

$$\int_{\partial c} \omega \mathrel{\widehat{=}} \boldsymbol{\omega}^{\mathbf{T}}(\boldsymbol{\partial}\mathbf{c}) = (\boldsymbol{\omega}^{\mathbf{T}}\boldsymbol{\partial})\mathbf{c} = (\boldsymbol{\partial}^{\mathbf{T}}\boldsymbol{\omega})^{\mathbf{T}}\mathbf{c} = (\mathbf{d}\boldsymbol{\omega})^{\mathbf{T}}\mathbf{c} \mathrel{\widehat{=}} \int_c \mathbf{d}\omega \tag{3.12}$$

Damit ergibt sich dann gerade, dass der Coboundary-Operator für k-Chains der Diffentialoperator für k-Cochains ist.

Nach Konstruktion ist der Satz von Stokes exakt erfüllt, da er zur Definition des Operators verwendet wurde.

3.6 Das Mesh

Um Riemannsche Mannigfaltigkeiten zu modellieren verwenden wir einen n-dimensionalen Simplex-Komplex, dessen Simplices die Mannigfaltigkeit diskretisieren. Nachdem wir bisher mit Definitionen gearbeitet haben, die auf beliebigen Objekten mit den entsprechenden Beziehungen arbeiten, müssen wir für ein Mesh konkrete Eigenschaften wie das Volumen der Objekte definieren.

Definition 3.5. *Wir bezeichnen das Volumen eines k-Simplex σ^k mit* $\mathrm{vol}(\sigma^k)$.

Das Volumen eines 0-Simplex (Punkt) definieren wir als 1, für alle höherdimensionalen lässt sich das Volumen über die Cayley-Menger-Determinante[Col] berechnen:

Sei $\mathbf{B}$ die $(k+1) \times (k+1)$ Matrix, in welcher die quadrierten Distanzen der Vertices des Simplex als $B_{ij} = \|v_i - v_j\|_2^2$ stehen, und $\hat{\mathbf{B}}$ die $(k+2) \times (k+2)$ Matrix, welche $\mathbf{B}$ erweitert, sodass die erste Zeile und die erste Spalte den Vektor $(0, 1, \ldots, 1)$ enthalten, dann lässt sich das Volumen des Simplex berechnen als:

$$\mathrm{vol}(\sigma^k)^2 := \frac{(-1)^{k+1}}{2^k (k!)^2} \det(\hat{\mathbf{B}}). \tag{3.13}$$

Mit dem Laplaceschen Entwicklungssatz folgt, dass sich das Volumen eines k-Simplex rekursiv aus dem Volumen eines seiner $(k-1)$-Faces und den Abständen der Vertices $v_1, \ldots, v_k$ des Faces zum Vertex v_{k+1} zusammen mit einem nur von der Dimension abhängigem Vorfaktor berechnet.

Diese Eigenschaft hat den Vorteil, dass wenn das Volumen aller 0-, $\ldots, n$-Simplices berechnet werden soll, eine rekursive Formel verwendet werden kann, welche die bereits berechneten Volumen verwendet

$$\mathrm{vol}(\sigma^k) := \frac{\mathrm{vol}(\sigma^{k-1}) h}{k} \tag{3.14}$$

mit $\sigma^{k-1} \prec \sigma^k$ und h als „Höhe“, also als minimale L_2-Distanz des neuen Punktes zur Hyperebene in der σ^{k-1} liegt.

Definition 3.6. *Das Circumcenter eines Simplex ist der Mittelpunkt der Hyperkugel, auf deren Oberfläche alle Vertices des Simplex liegen.*

Beispiel: Das Circumcenter eines Dreiecks (2-Simplex) ist der Mittelpunkt des Umkreises des Dreiecks.

Nicht bei allen Simplices liegt das Circumcenter in der konvexen Hülle der Vertices. Das Circumcenter liegt zum Beispiel außerhalb, wenn ein Dreieck einen Winkel von mehr als 90 Grad enthält. Ist das Dreieck rechtwinklig, liegt das Circumcenter auf der Hypotenuse. In Grafik 3.6 sind zwei Beispiele für das Circumcenter anhand von Dreiecken zu sehen.

Definition 3.7. *Ein Simplex heißt wohlzentriert, wenn es sein Circumcenter in seinem Innerem liegt.*

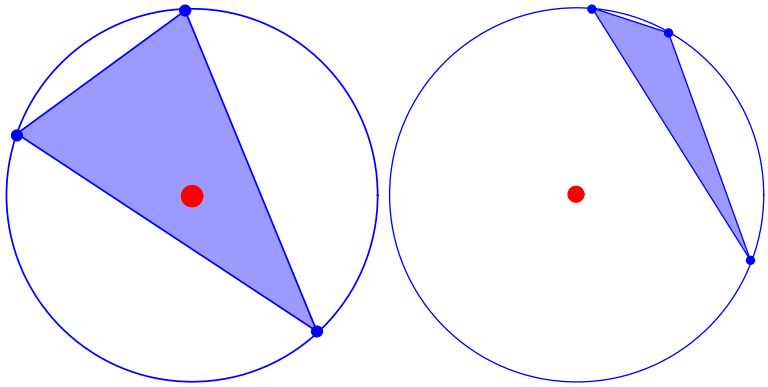


Abb. 3.6: Das Circumcenter eines Dreiecks ist der Mittelpunkt des Umkreises um das Dreieck und liegt nicht unbedingt im Inneren des Dreiecks.

Wir verwenden Im Folgenden das Circumcenter als Mittelpunkt und geben explizit an, wenn ein anderer Mittelpunkt gemeint ist.

Wir nehmen außerdem an, dass alle Simplices wohlzentriert sind.

3.7 Das duale Mesh

Zu einem gegebenem Simplex-Komplex können wir ein duales Mesh K_d berechnen. In einem dualem n-dimensionalem Mesh ist jedem k-Simplex σ^k eine duale $(n-k)$-Zelle σ_d^{n-k} zugeordnet. Diese $(n-k)$-dimensionale Zelle ist im allgemeinem kein Simplex. Ein Beispiel für einen dualen Komplex ist in Grafik 3.7 zu sehen.

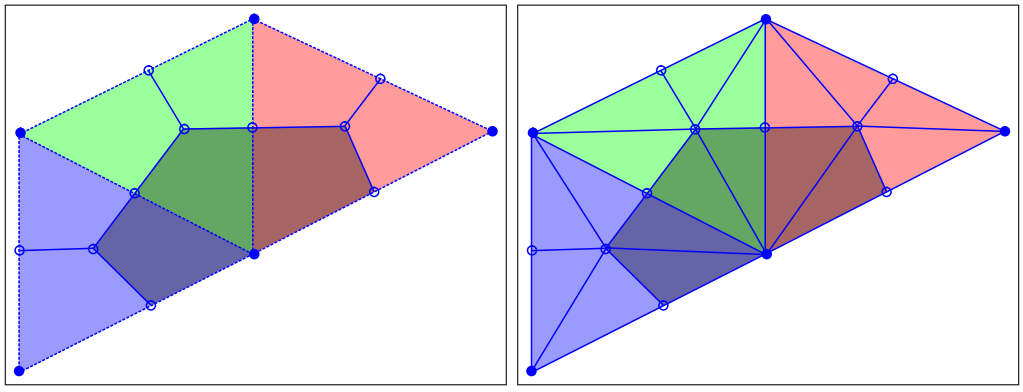


Abb. 3.7: **Links**: Das Mesh mit eingezeichnetem dualem Zellenkomplex. Eine Zelle ist dunkel hervorgehobenen. **Rechts:** Der gleiche Komplex als Subdivision-Mesh

Definition 3.8. *Eine circumcentrische duale Zelle σ_d^{n-k} eines k-Simplex ist die konvexe Hülle des Mittelpunkts des primalen Simplex und der Mittelpunkte der angrenzenden d-Simplices mit $d \leq n-1$. Die dualen Zellen aller Simplices eines Simplex-Komplex bilden den dualen Zellenkomplex des Simplex-Komplex.*

Bemerkung: Innerhalb des Meshs ist die duale Zelle die konvexe Hülle der Mittelpunkte der benachbarten n-Simplices, aber am Rand des Komplex werden duale Zellen dadurch dass es keine benachbarten n-Simplices gibt „abgeschnitten“, weswegen dort der Mittelpunkt des $n-1$ verwendet wird, denn im circumcentrischen Mesh geht eine Kante, die Mittelpunkte zweier n-Simplices verbindet gerade durch den Mittelpunkt des gemeinsamen Faces.

Ein Beispiel ist in Grafik 3.7 zu sehen. Der eingezeichnete Bereich ist eine duale 2-Zelle am Rand des Komplexes, die Kanten zwischen den Mittelpunkten der Dreiecke sind duale 1-Zellen und die Mittelpunkte selber sind duale 0-Zellen.

Analog zur Definition von primalen Chains und Co-Chains gibt es duale Chains und Co-Chains.

Definition 3.9. *Eine duale Chain $c_d^k \in \mathcal{C}_{k,d}$ ist eine Linearkombination von dualen k-Zellen.*

Definition 3.10. *Eine duale Co-Chain $\omega_d^k \in \Omega_{k,d}$ ist eine Abbildung von $\mathcal{C}_{k,d} \to \mathbb{R}$, welche jeder dualen Chain einen Wert zuweist.*

Durch die eindeutige Abbildung von k-Simplices auf $(n-k)$-Zellen ergibt sich, dass der primale Boundary-Operator gerade der duale Coboundary-Operator ist und der duale Boundary-Operator der primale Coboundary-Operator.

Damit gilt auch, dass der duale Differentialoperator $\mathbf{d}_d$ gerade der primale Boundary-Operator $\boldsymbol{\partial}$ ist.

Das Subdivision-Mesh

Mit der obigen Definition des dualen Meshs lässt sich eine neue Vertexmenge definieren, welche neben den primalen Vertices auch die Mittelpunkte der primalen Simplices enthält. Mit diesen Vertices lässt sich eine Menge dualer Simplices definieren, welche das sog. Subdivision-Mesh bilden [Hir03].

Definition 3.11. *Das Subdivision-Mesh $K_{sd}^{(0)}$ ist definiert als*

$$K_{sd}^{(0)} := \left\{ [c(\sigma^n)] | \sigma^n \in K^{(n)} \right\} \tag{3.15}$$

$$K_{sd}^{(k)} := \left\{ \sigma_d^{k-1} \cup c(\sigma^{n-k}) | \sigma^{n-k} \prec \sigma^n \right\} \tag{3.16}$$

$$\text{mit } \sigma^n \in K^{(n)}, \sigma^{n-k} \in K^{(n-k)}, \sigma_d^{k-1} \in K_{sd}^{(k-1)}. \tag{3.17}$$

Das heißt, das 0-Skelett des dualen Komplexes (die Menge der dualen Punkte) sind 0-Simplices, welche die Mittelpunkte der n-Simplices des primalen Komplexes als ihren einen Vertex haben. Ein duales k-Simplex entsteht dadurch, dass ein duales $(k-1)$-Simplex um einen Mittelpunkt eines primalen $(n-(k-1))$-Simplex, welches mit dem primalem n-Simplex benachbart ist, erweitert wird.

Ein duales k-Simplex ist also ein Simplex aus den Mittelpunkten von primalen $0, \ldots, n-k$-Simplices: $\sigma_d^k = [c(\sigma^n), c(\sigma^{n-1}), \ldots, c(\sigma^0)]$.

Ein Beispiel für ein solches Subdivision-Mesh ist rechts in Grafik 3.7 zu sehen.

Die so definierten Simplices besitzen keine konsistente Orientierung, aber diese kann durch das primale Mesh induziert werden, da die dualen Simplices in der gleichen Ebene wie die primalen Simplices liegen.

Der Hodgestar-Operator

Ein Operator um primale k-Simplices auf duale $(n-k)$-Simplices abzubilden ist der Hodgestar-Operator $\star$, in der Notation von [Hir03] auch $*$. Wir orientieren uns hier an [ES05] und [DKT08] in denen $\star$ verwendet wird, da $*$ in der Literatur teilweise verwendet wird um duale Simplices und duale Differentialformen kenntlich zu machen.

Der Operator $\star_k : \Omega^k(K) \to \Omega^{n-k}(K_d)$ soll primale Differentialformen $\boldsymbol{\omega}^k$ auf duale Differentialformen $\boldsymbol{\omega}_d^{n-k}$ abbilden, sodass die Auswertung der dualen Form auf dualen Chains das gleiche Ergebnis hat, wie die Auswertung der primalen Form auf primalen Chains.

Es muss also gelten:

$$\omega(c) = \omega_d(c_d) = \star\omega(c_d)$$

Für diskrete Differentialformen ist der Hodgestar dabei eine Matrix, welche eine primale Co-Chain als Vektor auf den Vektor einer entsprechenden dualen Co-Chain abbildet.

Für den adjungierten Operator $\star_k^{-1}$ gilt:

$$\star_k \star_k^{-1} = (-1)^{k(n-k)}\mathbb{I}$$

Der Faktor $(-1)^{k(n-k)}$ stammt aus der Antisymmetrie des Operators, ist aber in den dreidimensionalen Meshs mit denen wir arbeiten werden, für alle $k = 0, \ldots, 3$ gerade 1.

Wenn wir für ω_i^k die Volumenform $vol(\sigma_i^k)$ wählen ergibt sich der Hodgestar:

$$(\star_k \omega^k)_i = \frac{\mathrm{vol}(\sigma_{i,d}^{n-k})}{\mathrm{vol}(\sigma_i^k)} \mathrm{vol}(\sigma_i^k) = \mathrm{vol}(\sigma_{i,d}^{n-k}) = (\omega_d^{n-k})_i \tag{3.18}$$

$$(\star_k)_{ii} = \frac{\mathrm{vol}(\sigma_{i,d}^k)}{\mathrm{vol}(\sigma_i^{n-k})} \tag{3.19}$$

In Formel 3.19 ergibt sich wegen Linearität, dass der Hodgestar-Operator im circumcentrischen Mesh eine Diagonal-Matrix ist, mit den Verhältnissen zwischen dualem und primalen Volumen auf der Diagonale.

Diese $n \times n$ Matrix kann nun per Matrix-Vektor-Multiplikation auf eine diskrete Co-Chain angewendet werden, um eine duale Co-Chain zu bekommen.

Der Codifferential-Operator

Damit lässt sich nun der Codifferential-Operator δ^k definieren, welcher k-Formen auf $(k-1)$-Formen abbildet:

$$\delta_k : \Omega^k(K) \to \Omega^{k-1}(K) \tag{3.20}$$
$$\delta_k : \omega^k \mapsto \omega^{k-1} \tag{3.21}$$
$$\delta_k := \star_{k-1}^{-1} \mathbf{d}_{n-k,d} \star_k \tag{3.22}$$

Der Operator δ definiert die Abbildung damit wie folgt:

1. Der Hodgestar $\star_k$ bildet eine k-Form auf eine duale $(n-k)$-Form ab.
2. Der dualen Differentialoperator $\mathbf{d}_d$ bildet die duale $(n-k)$-Form auf eine duale $(n-k+1)$-Form ab.
3. Der zu $\star_{k-1}$ adjungierte Hodgestar $\star_{k-1}^{-1}$ bildet die duale $(n-k+1)$-Form auf eine primale $(k-1)$-Form ab.

Der De-Rham Komplex

Aus diesem Zusammenhang ergibt sich der De-Rham Komplex, welcher im Kommutativitätsdiagramm in Abbildung 3.8 abgebildet ist.

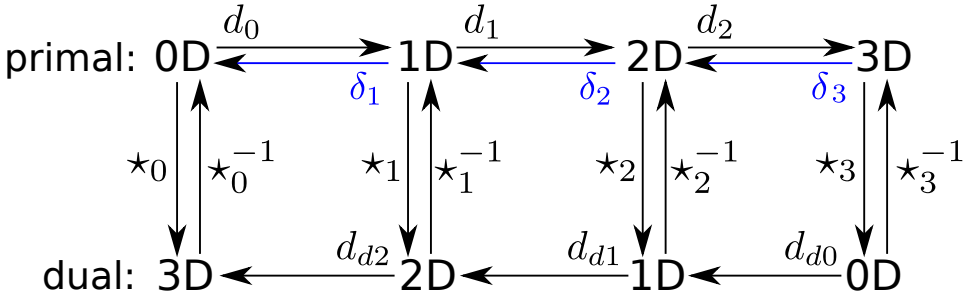


Abb. 3.8: Der De-Rham Komplex eines dreidimensionalen Meshes

Dort ist anschaulich erkennbar, wie der Operator δ mit Hilfe des Hodgestars und des Differentialoperators im dualem Mesh arbeitet. Mit einem solchem Operator, der von k-Formen auf $(k-1)$-Formen abbildet, können wir schließlich den diskreten Laplaceoperator für k-Formen definieren.

3.8 Laplaceoperator

Der diskrete Laplaceoperator ist definiert als

$$\mathbf{L_k} := \mathbf{d_{k-1}} \boldsymbol{\delta}_\mathbf{k} + \boldsymbol{\delta}_\mathbf{k+1} \mathbf{d_k}. \tag{3.23}$$

Bei $\mathbf{L_0}$ (Laplaceoperator auf den Punkten) und $\mathbf{L_n}$ (Laplaceoperator auf den Elementen mit der Dimension des Komplexes) fällt jeweils einer der Terme weg, da $\mathbf{d_n}$ und $\boldsymbol{\delta_0}$ nach der Definition des (Co-)Chain-Komplex gerade $\mathbf{0}$ sind.

Die Definition des Operators kann man sich einfach am Beispiel in Grafik 3.9 mit $\mathbf{L_0} = \boldsymbol{\delta_1} \circ \mathrm{d_0}$ veranschaulichen:

1. Gegeben eine 0-Co-Chain, die einer Funktion f entspricht (hier: Die Funktion weist einem Punkt einen Funktionswert größer 0 zu)

2. Bilde die 0-Co-Chain auf eine 1-Co-Chain ab, die auf den Kanten arbeitet, welche die Punkte mit Eintrag ungleich 0 in der 0-Co-Chain als Eckpunkt (Rand) haben. (Co-Boundary/Differential-Operator)

3. Bilde die 1-Co-Chain wieder auf eine 0-Co-Chain ab (Codifferential-Operator). Die resultierende Co-Chain angewendet auf eine 0-Chain der Punkte entspricht gerade der Anwendung von $\triangle f$ auf die Punkte.

Diese Methode $\triangle f$ zu berechnen entspricht gerade der Multiplikation des Graph-Laplace für $k = 0$, wie wir ihn in Abschnitt 2.6 definiert haben, mit einer 0-Co-Chain, welche den Punkten σ^0 den Wert $f(\sigma^0)$ zuweist.

Ein Beispiel ist in Abbildung 3.9 zu sehen. Die Orientierung dort ergibt sich daraus, dass eine positive gerichtete Kante den Eckpunkt auf den sie zeigt positiv orientiert und den anderen negativ. Ist die Kante selber negativ orientiert kehren sich die Vorzeichen an den Eckpunkten um.

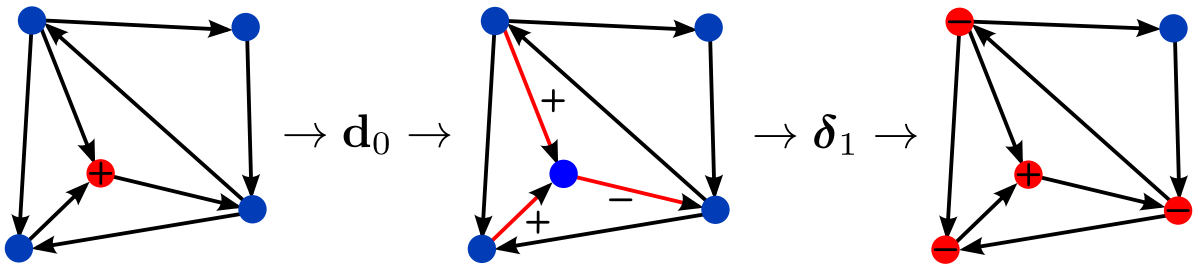


Abb. 3.9: Ein Beispiel, wie der diskrete Laplaceoperator arbeitet.

3.9 Sharp- und Flat-Operatoren

Weiter gibt es den Sharp-Operator $\sharp$ um 1-Formen auf Vektorfelder abzubilden und den Flat-Operator $\flat$ um Vektorfelder auf 1-Formen abzubilden.

Vektorfelder lassen sich auf den primalen oder auf den dualen Vertices definieren, wobei primale Vektorfelder nur für flache Meshes wohldefiniert sind, da ansonsten nicht klar ist, was der Tangentenraum ist. Bei dualen Vektorfeldern gibt es dieses Problem nicht.

Aufgrund der verschiedenen Vektorfelder und der unterschiedlichen Punkte auf denen das Vektorfeld interpoliert werden kann, ergeben sich verschiedene Operatoren, die mit **p**(primal) und **d**(dual) als Index gekennzeichnet werden, wie z.B. der primal-dual-dual Flat-Operator $\flat_{pdd}$. Dieser kann ein Vektorfeld auf jedem primalen 1-Simplex auswerten, wobei eine dual-dual Interpolation auf dem Mesh verwendet wird.

3.10 Datenstrukturen

Um mit dem DEC zu arbeiten brauchen wir Datenstrukturen, in denen die Objekte effizient gespeichert werden können. Dazu soll die Zugriffszeit auf ein Objekt möglichst klein sein, und Größen wie Volumen, Circumcenter und weitere sollen auch schnell verfügbar sein. Gerade bei Matrizen ist auch eine effiziente Speicherung für dünn besetzte Matrizen wichtig, damit eine Multiplikation schnell erfolgen kann, da Operatoren wie der Boundary-Operator, der Hodgestar und einige weitere stets dünn besetzt sind.

Datentypen

Die grundlegenden Datentypen, die für die Strukturen verwendet werden:

- Zahlen, typischerweise „float“ (Fließkommazahl) oder „int“ (ganze Zahl)
- Listen
- „dicts“ (Hashtabellen), welche eine Schlüssel→Wert Zuordnung erlauben.
- „sets“ (Mengen), die Elemente genau einmal speichern, auch wenn sie mehrfach hinzugefügt werden. Sets unterstützen außerdem die Operationen „union“ (Vereinigung von Mengen $A \cup B$), „intersect“ (Schnitt von Mengen $A \cap B$) und „difference“ (Mengen-Differenz $A \setminus B$)
- (Sparse-)Matrix-Objekte, welche Matrizen und Vektoren speichern können und Operationen wie Matrix-Vektor-Multiplikation unterstützen.

In der Python-Implementierung werden die normalen Python-Strukturen für Listen, Dicts und Sets verwendet, für die Matrix-Objekte wird die numpy/scipy-Bibliothek[2] verwendet.

Speicherung

Die Indizes aller Listen und alle Matrix-Indizes für Zeilen und Spalten beginnen mit 0.

Die Vertices $v_1, \ldots, v_n \in \mathbb{R}^d$ werden als eine Liste von d-dimensionalen Vektoren mit n Einträgen gespeichert.

[2]Verfügbar unter `http://www.numpy.org/` und `http://www.scipy.org`

```
1 vertices[0]     # erster Vertex
2 vertices[0][1] # Y-Koordinate des ersten Vertex
```

Für die Simplices wird je Dimension $k \in 0, \ldots, d$ eine Liste von k-Simplices gespeichert, welche die Simplices als Tupel $(v_0, \ldots, v_k)$ enthält. Diese Listen sind in einer Liste von Simplexlisten gespeichert. Der Zugriff funktioniert also wie folgt:

```
1 simplices[2][4]    # das 5. 2-Simplex
2 len(simplices[2]) # Anzahl 2-Simplices
```

Der Randoperator

Die Randoperatoren für k-Simplices mit $k = 1, \ldots, n$ werden als Matrix in einer Liste von Matrizen gespeichert, die allerdings nur bei Bedarf berechnet wird.

Für den Zugriff auf die Randsimplices wird ein Dict verwendet, wodurch eine aufwendige Matrixmultiplikation entfällt, wenn nur der Rand weniger Simplices abgefragt werden muss.

```
1 # Zugriff auf das Set von d-1 dimensionalen Objekten,
2 # die auf dem Rand des $d$-dimensionalen Simplex
3 # mit Index i liegen.
4 upper_lower[d][i]
5 # Test ob der Punkt s0 ein Eckpunkt der Kante s1 ist:
6 s0 in upper_lower[1][s1]
```

Genauso wird der Coboundary-Operator als Dict `lower_upper` gespeichert, welches von jedem k-dimensionalem Objekt die angrenzenden $(k+1)$-dimensionalen Objekte enthält.

Beispiel für ein Mesh-Dateiformat

Ein typisches 3D-Mesh Dateiformat ist zum Beispiel das Wavefront-.obj-Format. In diesem Format werden zunächst alle Vertices mit einer Anweisung „v x y z“ definiert, wobei x, y und z die Koordinaten des Vertex sind. Danach werden dann die Faces im Format „f i_1 i_2 i_3 [i_4 i_5 ...]“ als Liste von Indices definiert. In 3.1 ist ein Listing eines einfachen OBJ-Meshes zu sehen.

Im DEC betrachten wir als Input nur Simplexmeshes, das heißt ein n-dimensionales Objekt ist ein n-Simplex. Das stellt keine Einschränkung dar, da jedes Element das kein Simplex ist solange weiter unterteilt werden kann, bis alle Teilobjekte Simplices sind. Ein typischer Anwendungsfall ist „Quads“ in Dreiecke zu unterteilen.

Tabelle 3.1: Eine Beispiel-Datei im Wavefront .obj-Format, welche zwei Dreiecke mit einer gemeinsamen Kante beschreibt.

```
//     x   y   z
    v   0   0   0
    v   1   0   0
    v   0   1   0
    v   0  -1   0
    f   1   2   3
    f   1   2   4
```

3.11 Effiziente Berechnung

Duale Volumen

Die dualen Volumen lassen sich rekursiv von dualen 0-Simplices bis zu dualen n-Simplices berechnen. Analog zu Definition 3.5 sei das Volumen eines dualen 0-Simplex als 1 festgelegt.

Dann kann man das Volumen der dualen $1, \ldots, n$-Zellen als Summe der Produkte aus dem dualem Volumen der jeweiligen vorher berechneten $(k-1)$-Faces b und der Höhe h, welche die Strecke vom Mittelpunkt des $(k-1)$-Faces zum Mittelpunkt des k-Simplex ist, berechnen als

$$\frac{1}{k} \sum_{\sigma^{k-1} \prec \sigma^k} bh.$$

Der Algorithmus ist für die Berechnung aus primalen Punkten, Kanten, Dreiecken und Tetraedern in [ES05] beschrieben, lässt sich aber auch für höherdimensionale Simplices verallgemeinern:

$$\begin{aligned} \mathrm{vol}(\sigma^0_{d,i}) &:= 1 \\ \mathrm{vol}(\sigma^k_{d,i}) &:= \frac{1}{k} \sum_{\sigma_j^{(n-k)+1} \succ \sigma_i^{(n-k)}} \left(\mathrm{vol}(\sigma^{k+1}_{d,j}) \left\| c(\sigma_i^{n-k}) - c(\sigma_j^{(n-k)+1}) \right\| \right). \end{aligned}$$

Der Faktor von $\frac{1}{n!}$ für das Volumen eines n-Simplex ergibt sich daraus, dass duale Volumen der $(k-1)$-dimensionalen „Grundfläche“ b eines k-Simplex bereits aus den vorherigen Schritten mit dem Faktor $\frac{1}{(k-1)!}$ multipliziert sind.

Optimierungen

Cache

Mit Hilfe eines Caches können Objekte, die erst beim Zugriff generiert werden für den nächsten Zugriff gespeichert werden. Der implementierte Cache unterstützt Abhängigkeiten von anderen Objekten. Das heißt, wenn sich das Objekt, aus dem eine Größe berechnet wurde, in der Zwischenzeit geändert hat, dann werden die angefragten Daten aus dem geändertem Objekt erneut berechnet.

Der @cached-„Decorator“ speichert dazu einen Timestamp der Daten, aus denen das Ergebnis generiert wurde und erzeugt ein neues Ergebnis, wenn der Cache-Timestamp älter als der Timestamp der Daten ist, aus denen das Ergebnis erzeugt wurde.

Ein Beispiel ist im Listing in Abbildung 3.10 zu sehen.

```
1 @cached("centers", "mesh")
2 def centers(self):
3     #[...]
4     return result
```

Abb. 3.10: Beispiel für den Cache: Die Mittelpunkte der Simplices werden nur dann neu berechnet, wenn sich das Mesh geändert hat.

Boundingboxen

Boundingboxen können bei der Suche, in welchem Tetraeder ein Punkt liegt verwendet werden. Sie ermöglichen einen schnellen Test, ob der Punkt überhaupt in diesem Tetraeder liegen kann. Danach erfolgt der normale Test, ob der Punkt auch im Tetraeder selber liegt, oder nur in dessen Boundingbox.

In dieser einfachen Umsetzung können sich zwar theoretisch beliebig viele Boundingboxen überschneiden, sodass auch beliebig viele Tetraeder getestet werden müssen, aber in der Praxis sind die Überschneidungen meistens stark begrenzt (in vielen Fällen überschneidet sich die Boundingbox eines Tetraeders nur mit denen seiner direkten Nachbarn).

Diese Implementierung hat eine Komplexität von $\mathcal{O}(n)$, daher würde sich für eine Suche in wirklich vielen Tetraedern eine Datenstruktur wie in Octree anbieten, welcher einen Zugriff in $\mathcal{O}(\log n)$ ermöglicht.

3.12 Mesh-Verfeinerung

Um einen Simplex-Komplex zu verfeinern, muss die Methode wieder einen gültigen Simplex-Komplex erzeugen und außerdem sollen Eigenschaften wie die Orientierung der Simplices und Wohlzentriertheit erhalten bleiben.

Somit kann zum Beispiel kein neues Vertex auf dem Rand eines Simplex liegen, wenn nicht die angrenzenden Simplices ebenfalls entsprechend so verfeinert werden, dass das

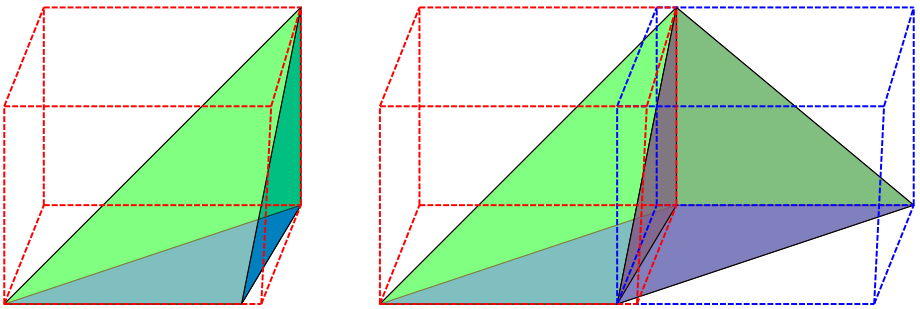


Abb. 3.11: Die Boundingbox eines Tetraeders ist der kleinste Quader, welcher den Tetraeder komplett enthält. Rechts ist ein Beispiel zu sehen, in dem sich zwei Boundingboxen überschneiden.

Vertex in den neuen Simplices enthalten ist und keine „T-Stücke“, wie sie in Grafik 3.12 zu sehen sind, entstehen. In diesem Fall wäre das Face des nicht verfeinerten Dreiecks kein Face der beiden neuen Dreiecke mehr, und die Menge der Simplices damit kein gültiger Simplex-Komplex.

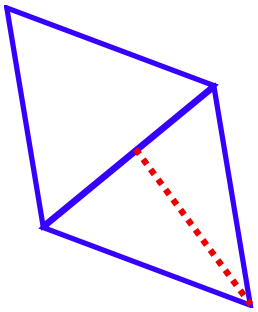


Abb. 3.12: Diese Verfeinerung ist kein gültiger Simplex-Komplex, da sie ein T-Stück erzeugt.

Eine Möglichkeit das Mesh so zu verfeinern, dass es ein Simplex-Komplex bleibt, wäre zum Beispiel im Inneren eines $(k-1)$-Faces zwischen zwei Simplices einen neuen Vertex einzufügen und aus allen Kombinationen von k Vertices, der beiden Simplices, die neuen Simplices zu bilden. Danach müssen die zwei ursprünglichen Simplices aus dem Komplex entfernt werden, damit die Simplexmenge einen neuen Simplex-Komplex ergibt.

Der Vorteil eines solchen Ansatzes ist, dass er lokal angewendet werden kann, da die äußeren Faces gleich bleiben. Daher muss an anderen Simplices nichts geändert werden.

Der Nachteil an einem solchem einfachem Ansatz ist unter anderem, dass sich die Winkel verändern und so zum Beispiel die Wohlzentriertheit verloren gehen kann.

Anstatt diesen Ansatz weiter zu verbessern, werden wir uns nun einer globalen Verfeinerung zuwenden, welche diese Eigenschaften erhält.

Verfeinerung von Dreiecksmeshs durch Einfügen eines inneren Dreiecks

Eine Methode Meshes so zu verfeinern dass diese wichtigen Eigenschaften erhalten bleiben ist das Verfeinern mit neuen Vertices, die aus den Mittelpunkten der Randfaces gebildet werden. Wir beschrieben dies hier exemplarisch für die Verfeinerung eines Dreiecksmeshs. Somit werden zum Beispiel Dreiecke um den Faktor 4 verfeinert, wie in Grafik 3.13 dargestellt ist.

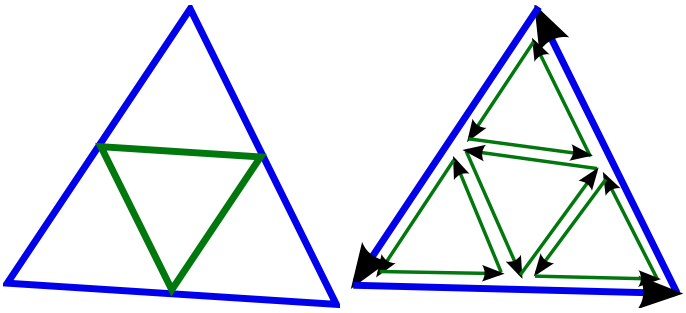


Abb. 3.13: Durch Einfügen eines inneren Dreiecks, welches die Mittelpunkte der Kanten als Vertices hat, wird das Dreieck um den Faktor 4 verfeinert. Rechts sieht man, dass die Orientierungen des Dreiecks außen die gleichen bleiben und dass das innere Dreieck in Bezug auf die äußeren Dreiecke positiv orientiert ist (d.h. die Dreiecke orientieren die gemeinsamen Kanten jeweils unterschiedlich).

Diese Verfeinerung ist nicht lokal möglich, da sie zu T-Stücken führt, wenn sie nicht auf *alle* Dreiecke des Meshes angewendet wird.

Ein Vorteil ist, dass sie für mehrere Dreiecke parallel berechnet werden, da diese Methode bei benachbarten Dreiecken auf dem gemeinsamen Face den gleichen Vertex erzeugt. Damit bietet sich zum Beispiel ein Divide-and-Conquer-Algorithmus an.

Eigenschaften

- Die neuen Vertices werden aus einem Face berechnet, und können daher aus beiden Dreiecken zu denen das Face gehört unabhängig voneinander berechnet werden, ohne das andere Dreieck zu berücksichtigen.
- Es bilden sich keine neuen „schlechten“ Winkel. Die Winkel der Dreiecke entsprechen denen des Ursprungsdreiecks, ist dieses wohlzentriert, sind die neuen Dreiecke es auch.
- Die Verfeinerung kann mehrfach hintereinander ausgeführt werden und es ist auch möglich mehrere Verfeinerungsschritte gleichzeitig zu berechnen.
- Die Orientierung der äußeren Kanten bleibt nach der Unterteilung gleich.
- Die alte Vertexmenge bleibt als Teilmenge der Neuen erhalten und kann sogar die verwendeten Indizes behalten.

Berechnung

Input: Vertexliste V, Liste von Dreiecken F mit $f = [v_i, v_j, v_k] \in F$ und $v_i, v_j, v_k \in V$.
Output: Vertexliste V', welche um $|E|$ neue Vertices erweitert wurde, Dreiecksliste F', welche $4|F|$ neue Dreiecke enthält.
Schritt 1: Füge alle Vertices aus V der Liste V' hinzu. Laufe dann über alle Kanten $[i, j]$, berechne den Mittelpunkt m_{ij}, füge ihn der Vertexliste V' hinzu und speichere eine Referenz unter der ungerichteten Kante $\{i, k\}$ ab.
Schritt 2: Laufe über alle Dreiecke $[v_i, v_j, v_k]$ und verbinde jeden Eckpunkt i des Dreiecks mit den Mittelpunkten der zwei angrenzenden Kanten in der Reihenfolge $[i, c_{ij}, c_{ki}]$, und speichere dieses Simplex in der Liste der neuen Simplices ab.

Da das Dreieck durch die Reihenfolge seiner Eckpunkte $v_i \to v_j \to v_k \to v_i$ seine Orientierung bekommt, erhält das Teilstück der äußeren Kante die gleiche Orientierung, wie die Ursprüngliche Kante. Das innere Dreieck erhält dadurch ebenfalls die Umlaufrichtung des ursprünglichen Dreiecks, wodurch seine Kanten konsistent mit den drei anderen Dreiecken orientiert sind.

Möchte man eine solche Verfeinerung nach der Berechnung des DEC anwenden, ist zu beachten, dass auch die Listen der Simplices aktualisiert werden und dadurch auch alle Operatoren neu berechnet werden müssen.
Schritt 3: Wiederhole Schritte 1 und 2, bis das gewünschte Level der Verfeinerung erreicht ist. Die Vertexliste V muss in jedem Schritt nur um die neuen Punkte erweitert werden, die Liste der Dreiecke F wird hingegen in jedem Schritt komplett neu berechnet.

3.13 Hochdimensionale Simplex-Komplexe

Die bisher betrachteten Simplex-Komplexe waren zwei- bzw. dreidimensional und aus gegebenen Meshes erzeugt, welche für viele Anwendungen die gegebene Eingabe sind. Während sich so zum Beispiel physikalische Probleme gut modellieren lassen, kann der DEC aber in beliebigen Dimensionen definiert werden, da Relationen wie Nachbarschaft von k-Simplices, $(k-1)$-Faces und andere unabhängig von der konkreten Dimension definiert sind. Auch das Circumcenter und das (Hyper-)Volumen von k-Simplices kann genauso in höheren Dimensionen berechnet werden.

Der Idee, einen Simplex-Komplex aus hochdimensionalen Daten zu verwenden, liegt die Annahme zugrunde, dass die Punktemenge zwar in einem hochdimensionalem Raum liegt, aber aus einer Verteilung mit geringerer Dimension stammt. Die Struktur in der niedrigen Dimension lässt sich in den hochdimensionalen Daten nicht direkt erkennen, lässt sich aber z.B. durch Nachbarschaftsgraphen (siehe Abschnitt 2.5) rekonstruieren. Um mit den diskreten Differentialformen des DEC zu rechnen muss ein Simplex-Komplex aus den Daten erzeugt werden.

Die verschiedenen Methoden Simplex-Komplexe zu erzeugen, verwenden jeweils Parameter mit denen sich die Nachbarschaftsbeziehungen steuern lassen (vgl. k Nachbarn beim kNN Graph, Radius beim r-Graph). Diese steuern auch, wie viele Simplices entstehen, und welche Dimension diese haben. Wird dieser Parameter passend gewählt, werden Simplices

der Dimension in welcher die Daten eigentlich liegen erzeugt. Im Folgenden Abschnitt zum Čech-Komplex ist in Grafik 3.15 ein Beispiel zu sehen, wie die höchste Dimension der Simplices im Komplex von den verwendeten Parametern abhängt.

Bei einem solchem Simplex-Komplex müssen nur die Simplices der jeweils höchsten Dimension konkret erzeugt werden, da andere Simplices dann Faces von höherdimensionalen Simplices sind und rekursiv berechnet werden können, indem jeweils einer der Punkte eines k-Simplex entfernt wird um ein $(k-1)$-Simplex zu bekommen.

Wir erinnern uns an die Bedingungen für einen Simplex-Komplex:

- Ist ein Simplex σ^k im Komplex K enthalten, dann sind auch alle seine Faces $\sigma^\ell \prec \sigma^k$ für alle $\ell < k$ im Komplex enthalten.
- Zwei Simplices σ_1^k, σ_2^k schneiden sich entweder an einem gemeinsamen Face σ^{k-1}, oder gar nicht.

Diese Eigenschaften müssen nun auch für die generierten hochdimensionalen Simplices gelten. Verschiedene Methoden um solche Simplices zu erzeugen werden in [CDS03] vorgestellt, unter anderen der Čech-Komplex, der Rips-Komplex und der α-Komplex, welche im Folgenden vorgestellt werden. Beim Čech Komplex ist die Idee der Methode sehr einfach und die Intuition warum sinnvolle Simplices erzeugt werden leicht zu verstehen, der Rips-Komplex bietet eine abgewandelte Form des Čech-Komplex, welche sich einfacher berechnen lässt, und der α-Komplex wird über die α-Shapes [EKS83] definiert, welche eine Verallgemeinerung der konvexen Hülle um eine Punktwolke sind.

Der Čech-Komplex

Der Čech-Komplex entsteht durch den Schnitt von Hyperkugeln mit einem Radius R (in der Literatur teilweise auch ϵ) um die gegebenen hochdimensionalen Vertices.

Schneiden sich $k+1$ Hyperkugeln, definieren sie ein k-Simplex mit den Vertices, welche die Mittelpunkte der Hyperkugeln sind.

Abhängig vom Radius schneidet sich eine Kugel mit unterschiedlich vielen anderen Kugeln, sodass die höchste Simplex-Dimension die erzeugt wird vom Radius R abhängt.

Wenn die Punkte auf einer k-dimensionalen Mannigfaltigkeit liegen, erzeugt der Algorithmus mit dem gleichem Radius R die gleichen Simplices, auch wenn die Punkte in einem n-dimensionalem Raum mit $n > k$ eingebettet sind.

Beispiel: Der 3D-Torus aus Grafik 3.14 kann in den zehndimensionalen Raum eingebettet werden und dort gedreht werden, ohne dass sich die Simplices ändern, obwohl die Vertices selber andere hochdimensionale Koordinaten haben.

Satz 3.1. *Die Menge der Simplices mit der höchsten Dimension im Čech-Komplex bildet einen gültigen Simplex-Komplex.*

Lemma 3.1. *Der Schnitt von $k+1$ Hyperkugeln ergibt ein im gültiges k-Simplex zusammen mit seinen $0, \ldots, (k-1)$-Faces.*

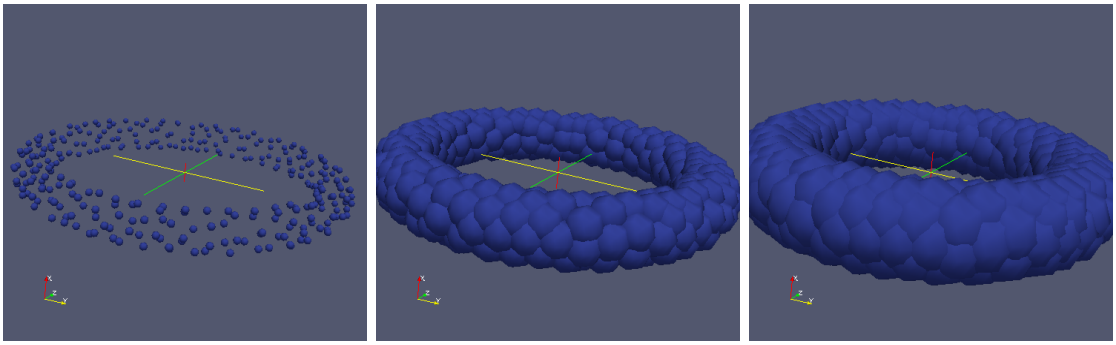


Abb. 3.14: Dreidimensionale Kugeln verschiedener Radien (von links nach rechts: 0.1, 0.5, 0.8), welche die gegebenen Vertices als Mittelpunkt haben. Mit den Kugelschnitten können Simplices aus den Punkten der Schnittmengen berechnet werden.

Beweis:
Gibt es Überschneidungen von $k+1$ Hyperkugeln, lässt sich darauf ein gültiges k-Simplex konstruieren, welches aus den Vertices der Kugeln besteht. Da sich an mindestens einem Punkt $(k+1)$ Kugeln schneiden, gibt es jeweils auch für jede Teilmenge der Kugeln mit Kardinalität $1 \dots k$ Überschneidungen, welche die $(k-1) \dots 0$-Simplices erzeugen, welche die Faces des k-Simplex bilden.

□

Durch den Existenzbeweis müssen die Schnitte niedrigerer Dimension nicht mehr explizit berechnet werden und die Faces können rekursiv aus dem k-Simplex erzeugt werden.

Lemma 3.2. *Zwei k-Simplices in einem Čech-Komplex schneiden sich nicht, oder sie schneiden sich in einem gemeinsamen Face.*

Beweis:
Angenommen zwei n-Simplices schneiden sich, aber nicht in einem gemeinsamen Face. Dann schneiden sich Faces beider Simplices. Ein k-Face wird definiert durch den Schnitt seiner $(k+1)$ Hyperkugeln um die Vertices, womit sich die Kugeln von Vertices der beiden Faces aus unterschiedlichen Simplices schneiden müssen, wenn sich die Faces schneiden. Dadurch entsteht ein neues Face, auf dessen Rand die Faces der Simplices, die sich schneiden, liegen. Da das Face damit zu beiden n-Simplices gehört, schneiden diese sich in einem gemeinsamen Face, was im Widerspruch zur Annahme steht.

□

Da zu jedem Simplex die Faces ebenfalls im Komplex enthalten sind, und zwei Simplices sich nur in einer gemeinsamen Kante schneiden können ist mit den beiden Lemmata der Satz bewiesen. □

Die Dimension der Simplices hängt vom gewähltem Radius ab. Je größer der Radius wird, desto höher wird die Dimension der Simplices, da sich mehr Hyperkugeln schneiden. Um die Mannigfaltigkeit der Daten zu rekonstruieren, muss daher der Radius entsprechend gewählt werden. Er muss groß genug sein, dass auch an Stellen mit geringer Punktdichte

Simplices entstehen, aber klein genug, dass an dichten Stellen keine Simplices zu hoher Dimension entstehen. In Grafik 3.15 ist für das Beispiel des in 10 Dimensionen eingebetteten Torus zu sehen, wie die höchste Dimension eines Simplex im erzeugtem Komplex vom Radius abhängt.

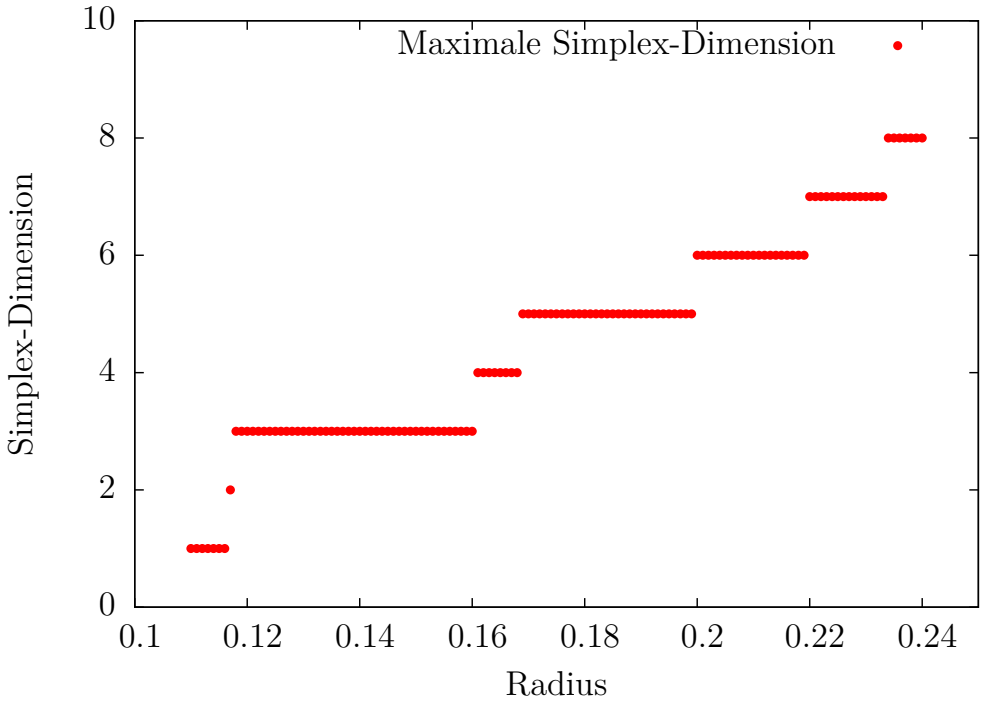


Abb. 3.15: Die maximale Dimension des Čech-Komplexes in Abhängigkeit vom verwendeten Radius zwischen 0.11 und 0.24

Der Rips-Komplex

Der Rips-Komplex ist ähnlich wie der Čech-Komplex definiert, allerdings werden nur die 1-Simplices (Kanten) durch Schnitte definiert. Die höherdimensionalen Simplices werden konstruiert, indem alle Mengen mit $k > 2$ Vertices, die paarweise durch Kanten verbunden sind, zur Simplexmenge hinzugefügt werden.

Während der Rips-Komplex der Definition dem Čech-Komplex ähnlich erscheint, gibt es je nach Lage der Punkte und Größe des Radius R deutliche Unterschiede. Dadurch, dass sich beim Rips-Komplex die Hyperkugeln der Vertices nur paarweise schneiden müssen, gibt es Konstellationen, bei denen der Rips-Komplex Simplices enthält, welche im Čech-Komplex nicht enthalten sind. Ein Beispiel ist ein gleichseitiges Dreieck, wie es in in Grafik 3.16 zu sehen ist. Dort kann es vorkommen, dass sich die Kreise um die Vertices zwar paarweise schneiden, es aber keine Stelle gibt, an der sich alle drei Kreise schneiden. Damit entsteht im Rips-Komplex ein Dreieck, während im Čech-Komplex nur die Kanten

des Dreiecks hinzugefügt werden.

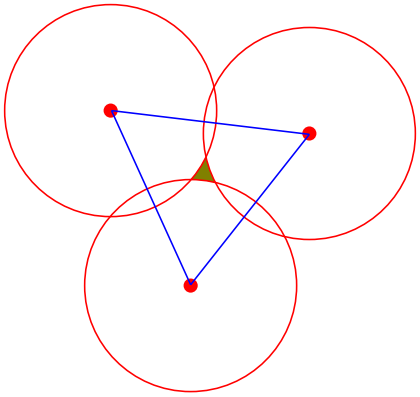


Abb. 3.16: Im Rips-Komplex entsteht hier ein 2-Simplex (Dreieck), welches im Čech-Komplex nicht enthalten ist.

Der Vorteil des Rips-Komplex ist, dass er einfacher zu berechnen ist, da nur die paarweisen Distanzen der Vertices berechnet werden müssen.

Simplices höherer Dimension können dann durch paarweise Nachbarschaft gefunden werden. Dazu wird für ein beliebiges Vertex des Simplex die Menge der Punkte in Abstand R aufgestellt, und dann für jeden der Punkte geprüft, ob der Abstand zu den anderen Vertices auch innerhalb des Radius R liegt. Wurde ein solcher Punkt gefunden, wird er dem Simplex hinzugefügt. Die Punktemenge des Startvertex kann im nächsten Schritt weiter verwendet werden, aber nun muss auch der Abstand zum hinzugefügtem Vertex mit überprüft werden.

Der α-Komplex

Der α-Komplex, basiert auf den α-Shapes, welche eine allgemeinere Definition der konvexen Hülle einer Punktemenge darstellen, wie sie von Edelsbrunner in [EKS83] definiert wurde. Dabei wird aus $k+1$ Punkten genau dann ein k-Simplex erzeugt, wenn es eine offene Hyperkugel mit Radius α gibt auf deren Rand die Punkte liegen, sodass innerhalb der Kugel kein Punkt liegt. Diese Punkte heißen dann α-exposed, und die Vereinigung aller Simplices deren Punkte α-exposed sind bilden das α-Shape.

Der Parameter α steuert, wie viele Details erfasst werden. Für $\alpha \to 0$ ist das Ergebnis dabei gerade die Punktemenge, während das Ergebnis für $\alpha \to \infty$ die konvexe Hülle der Punkte ist.

Ein einfaches Beispiel ist der Torus aus dem Beispiel in Abbildung 3.14. Ist α zu groß, wird die konvexe Hülle der Punkte gebildet und das Loch wird nicht mehr korrekt abgebildet. Wird α zu klein, kann nicht einmal die Oberfläche des Torus komplett erfasst werden. Bei nicht gleichmäßig verteilten Punkten kann es auch zu Löchern in der Oberfläche kommen.

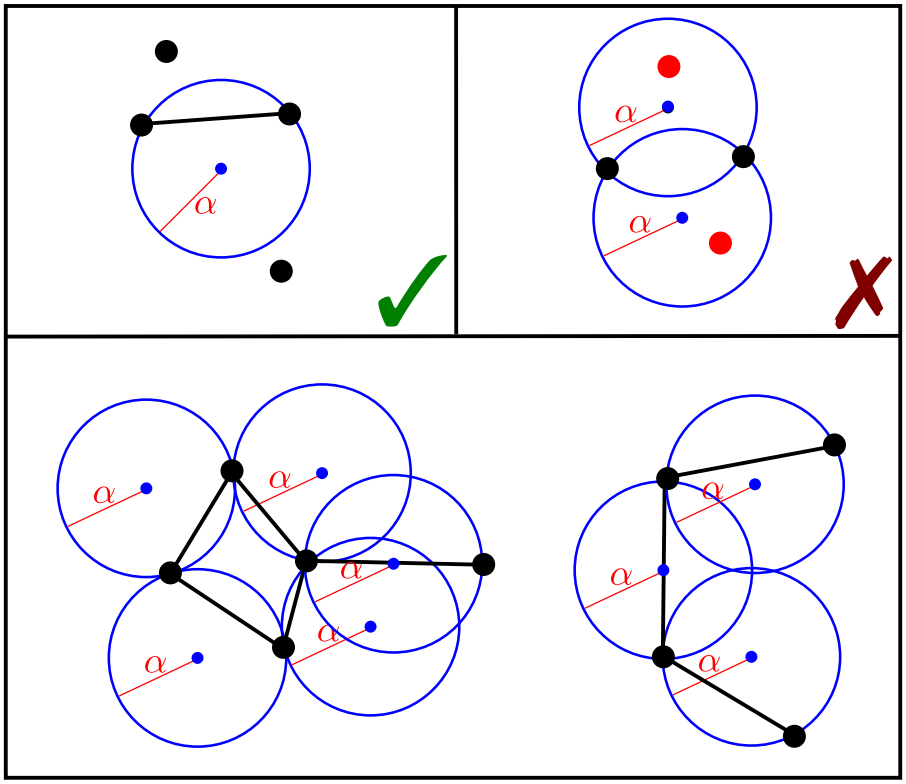


Abb. 3.17: Oben links ist ein Beispiel zu sehen, in welchem zwei Punkte α-exposed sind. Auf der rechten Seite sind diese Punkte *nicht* α-exposed, da beide mögliche α-Kreise einen anderen Punkt enthalten. Unten ist ein Beispiel für einen vollständigen α-Komplex der Dimension 1 zu sehen.

Die α-Shapes können daher ebenfalls eine Mannigfaltigkeit niedriger Dimension aus einer Menge von hochdimensionalen Punkten, die aus einer Verteilung mit geringerer intrinsischer Dimension der Punkte stammen, erzeugen.

Einen Bezug zum DEC stellt der mit den α-Shapes verbundene α-Komplex dar, welcher eng mit der Delaunay-Triangulierung der Punkte verbunden ist. Eine Definition des α-Komplexes zu einer Punktwolke findet sich in [CDS03]. Dieser ist mit dem Čech-Komplex nah verwandt, da beide mit einer Vereinigung von Hyperkugeln arbeiten. Allerdings erzeugt der α-Komplex deutlich weniger Simplices als ein Čech-Komplex.

Dadurch, dass der α-Komplex normalerweise aus einer Delaunay-Triangulation erzeugt wird unterliegt er dem „Fluch der Dimensionen“ bezüglich der Dimension des Raums in dem die Punkte liegen, wodurch er für eine effiziente Rekonstruktion einer Mannigfaltigkeit mit niedrigerer Dimension als die der Punkte im Allgemeinen nicht geeignet ist. Für geringe α fällt dies allerdings nicht so stark ins Gewicht.

3.14 Anwendungen

Mit dem DEC haben wir alle wichtigen Operatoren auf Mesh-Objekten definiert, die wir brauchen um PDEs zu lösen. Im Folgenden betrachten wir einige Probleme, die sich mit dem DEC einfach lösen lassen.

Wärmeleitung

Die Wärmeleitungsgleichung (bzw. Diffusionsgleichung) ist:

$$\frac{\partial}{\partial t}u(x,t) - \alpha \triangle u(x,t) = 0 \tag{3.24}$$

Die Temperatur u ist auf den Punkten x zu einem gegebenen Zeitpunkten t definiert. Dabei modelliert die Gleichung die Ausbreitung einer Starttemperatur u_0 auf einer Mannigfaltigkeit über eine Zeit t bei einer Temperaturleitfähigkeit von α. Mögliche Randbedingungen sind dabei Punkte mit einer festen Temperatur. Wir werden später sehen, dass auch eine Funktion $\alpha(x)$ möglich ist, falls die Wärmeleitung auf verschiedenen Teilen der Mannigfaltigkeit unterschiedlich ist.

Wenn die Zeit mit dem Euler-Vorwärts-Verfahren diskretisiert wird, lässt sich ein iterativer Zeitschritt mit den DEC-Operatoren als

$$\mathbf{u_{i+1}} := \mathbf{u_i} + \alpha \Delta_t \mathbf{L_0} \mathbf{u_i} \; \forall i = 1, \ldots, T \tag{3.25}$$

schreiben mit den folgenden Variablen:

- $\mathbf{u_i}$: Der Temperatur-Vektor mit $|V|$-Einträgen für die Temperaturen auf allen Punkten zum i-ten Zeitschritt.
- $\mathbf{L_0}$: Der diskrete Laplaceoperator für Punkte (0-Simplices).
- α: Der Temperaturleitfähigkeitskoeffizient des Materials.
- Δ_t: Die Zeitschrittweite.
- $\mathbf{i+1}$: Der Zeitschritt, welcher aus den Werten des Zeitschritts $\mathbf{i}$ berechnet werden soll.
- T: Anzahl der zu simulierenden Zeitschritte.
- $t = \Delta_t T$: Die Zeitspanne, die simuliert werden soll.

Zum Lösen geben wir einen Vektor $\mathbf{u_0}$ mit der Temperaturverteilung im ersten Zeitschritt vor und achten bei der Zeitschrittweite darauf, dass die CFL-Bedingung [CFL28]

eingehalten wird, indem wir Δ_t kleiner als das Quadrat der minimalen Kantenlänge wählen. Randbedingungen können gesetzt werden, indem nach jedem Zeitschritt im Vektor $\mathbf{u_i}$ die entsprechenden Werte wieder auf die Randbedingung gesetzt werden.

Die Verteilung der Wärme eines Punktes über die angrenzenden Kanten auf die umliegenden Punkte lässt sich anhand der Beschreibung der Schritte des diskreten Laplaceoperators in Abschnitt 3.8 nachvollziehen.

Die berechneten Ergebnisse sind in Abschnitt 4.1 zu sehen.

Krümmung berechnen

Eine weitere Anwendung ist, die Berechnung der mittleren Krümmung einer Oberfläche.

Ein einfaches Beispiel mit einer bekannten analytischen Lösung, mit der wir den Fehler der Lösung bestimmen können, ist die Krümmung einer Kugel, welche $\kappa = \frac{1}{R}$ ist.

Für die Berechnung der Krümmung gilt:

$$f_x(x,y,z) := x,\ f_y(x,y,z) := y,\ f_z(x,y,z) := z \tag{3.26}$$

$$\kappa := \|(\triangle f_x,\quad \triangle f_y,\quad \triangle f_z)\|_2 \tag{3.27}$$

Wir verwenden den diskreten Laplaceoperator $\mathbf{L_0}$ und bekommen so

$$\kappa_d := |(\mathbf{L_0} f_x, \mathbf{L_0} f_y, \mathbf{L_0} f_z)|.$$

Seien die Vektoren $\boldsymbol{\kappa}$ und $\boldsymbol{\kappa_d}$ die Vektoren mit den Werten der Krümmung an den Punkten, dann messen wir den Fehler der Krümmungsberechnung in der L_∞-Norm als $\|\boldsymbol{\kappa} - \boldsymbol{\kappa_d}\|_\infty$.

Das Ergebnis der Berechnung mit der Kugel als Beispiel ist in Abschnitt 4.4 gezeigt, wo auch die Konvergenz der Lösung betrachtet wird.

3.15 Darcy-Flow

Die Darcy-Gleichung, welche Henry Darcy 1856 aufgestellt hat um Grundwasserströmungen zu berechnen ist heute bekannt als ein Spezialfall der Navier-Stokes Gleichungen.

Ein Verfahren zur Lösung dieser Gleichung mit Hilfe des DEC ist in [HNC08] beschrieben, wovon hier die nötigen Teile um Fluss und Druck zu berechnen beschrieben und implementiert sind.

Die Darcy-Gleichung ist auf einer Mannigfaltigkeit Ω definiert als:

$$v + \frac{k}{\mu}\nabla p = \frac{k}{\mu}\rho g \qquad \text{auf } \Omega \tag{3.28}$$

$$\operatorname{div} v = \phi \qquad \text{auf } \Omega \tag{3.29}$$

$$v \cdot \hat{n} = \psi \qquad \text{auf } \partial\Omega \tag{3.30}$$

Mit k als Durchlässigkeitskoffizient, $\mu > 0$ als Viskosität der Flüssigkeit, ρ als Dichte der Flüssigkeit und g als Beschleunigung durch externe Kräfte.

Dabei ist $\phi : \Omega \to \mathbb{R}$ die vorgegebene Divergenz der Geschwindigkeit und $\psi : \partial\Omega \to \mathbb{R}$ die vorgegebene Geschwindigkeit in Normalenrichtung (gegeben durch den Einheitsnormalenvektor $\hat{n}$ nach außen) auf dem Rand von Ω.

Wir nehmen hier an, dass k konstant ist, und keine äußeren Kräfte einwirken, d.h. $g = 0$ gilt.

Wir schreiben nun die Gleichungen mit dem Discrete Exterior Calculus auf:

$$\mathbf{v}^\flat + \frac{k}{\mu}\mathbf{d_0 p} = 0 \qquad \text{auf } \Omega \tag{3.31}$$

$$\mathbf{d_{n-1}}(\star\mathbf{v}^\flat) = \boldsymbol{\phi\omega} \qquad \text{auf } \Omega \tag{3.32}$$

$$\star\mathbf{v}^\flat = \boldsymbol{\psi\gamma} \qquad \text{auf } \partial\Omega \tag{3.33}$$

Dabei ist $\omega = \star\mathbf{1}$ die Volumenform der n-Simplices in K^n und γ die Volumenform der $(n-1)$-Simplices auf dem Rand ∂K^n, welche definiert ist durch

$$\boldsymbol{\gamma}(X_1, \ldots, X_{n-1})^\flat = \boldsymbol{\omega}(\hat{n}, X_1, \ldots, X_{n-1})^\flat$$

für alle Vektorfelder $X_1, \ldots, X_{n-1}$ auf ∂K. Dabei nutzt 3.33 aus, dass für die dualen Simplices auf dem Rand $(v \cdot \hat{n})\gamma = \star v^\flat$ gilt, da sie orthogonal zu den primalen Simplices sind.

Nach Stokes' Theorem muss gelten:

$$\int_\Omega \phi\omega = \int_{\partial\Omega} \psi\gamma$$

und eine Differentialform für den Fluss über $(n-1)$-Simplices bekommen wir mit:

$$\mathbf{f} := \star(v^\flat)$$

Durch Anwendung des Hodgestar-Operators auf beiden Seiten können wir nun die diskreten Gleichungen abhängig von Fluss und Druck schreiben:

$$\mathbf{f} + \frac{k}{\mu}(\star\mathbf{d_0 p}) = 0 \qquad \text{auf } K^n \tag{3.34}$$

$$\mathbf{d_{n-1}f} = \boldsymbol{\phi\omega} \qquad \text{auf } K^n \tag{3.35}$$

$$\mathbf{f} = \boldsymbol{\psi\gamma} \qquad \text{auf } \partial K^n \tag{3.36}$$

Wir bekommen schließlich eine Matrix-Formulierung mit der man nach $\mathbf{f}$ und $\mathbf{p}$ lösen kann, wobei $\mathbf{p} \in \mathbb{R}^{|K^n|}$ den Druck in den Dreiecken (bzw. Tetraedern) und $\mathbf{f} \in \mathbb{R}^{n|K^{n-1}|}$

den Fluss über den Kanten (bzw. Dreiecke) zwischen den Volumen als Vektoren mit 2 bzw. 3 Komponenten jeweils enthält:

$$\begin{pmatrix} -(\mu/k)\star_{n-1} & \mathbf{d}_{n-1} \\ \mathbf{d}_{n-1}^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{f} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{\phi\omega} \end{pmatrix} \tag{3.37}$$

Diese Matrix ist in der Sattelpunkt-Form und dadurch effizient lösbar.

Zur Visualisierung des Flusses interpolieren wir den Fluss über die Kanten mit Whitney-1-Formen (siehe [DKT08]) und bilden diese dann mit dem $\sharp$-Operator auf ein Vektorfeld ab, welches auf den Barycentern der Dreiecke definiert ist. Dadurch dass $X^\flat = -(\star\mathbf{f}) = b\mathrm{dx} - a\mathrm{dy}$ gilt, ergibt sich der duale Vektor auf dem Barycenter gerade zu $(b, -a)$.

Die Ergebnisse der Rechnung sind in Abschnitt 4.5 gezeigt.

4 DEC-Ergebnisse

Nachdem alles notwendige definiert wurde, um mit Hilfe des Discrete Exterior Calculus PDEs mit diskreten Differentialformen und Operatoren zu lösen, werden nun einige anwendungsbezogene Gleichungen mit dem DEC diskretisiert, um numerische Ergebnisse zu erzielen, den Fehler der Diskretisierung zu messen und die Konvergenz der Ergebnisse in Bezug auf feinere Auflösungen der verwendeten Gitter zu ermitteln.

Implementierte Eingabe- und Ausgabeformate

Das im Rahmen dieser Diplomarbeit entwickelte Framework kann zwei- und dreidimensionale Meshes aus üblichen Dateiformaten direkt einlesen. Für zweidimensionale Meshes (Oberflächen im 3D-Raum) sind die Dateiformate „Wavefront OBJ“ (siehe Abschnitt 3.10) und das VTK-Dateiformat des Visualisierungsprogramms Paraview[1] implementiert. Für dreidimensionale Meshes (Tetraedermeshes) sind VTK, ELE[2] und MPHTXT, das Dateiformat des Programms Comsol[3], welches zur Modellierung von physikalischen Modellen genutzt wird, implementiert.

Damit ist es möglich 3D-Modelle aus frei verfügbaren Quellen, wie zum Beispiel dem 3D-Scan-Repository der Stanford Universität[4] einzulesen und direkt auf den Modellen zu rechnen.

Für 2D-Meshes wurde der Verfeinerungsalgorithmus aus Abschnitt 3.12 implementiert um auf verschieden fein aufgelösten Gittern rechnen zu können.

Weitere Dateiformate lassen sich einfach hinzufügen, indem die Vertices als Koordinaten und die Faces (bzw. Tetraeder) als Listen von Vertex-Indizes generiert werden. In der Praxis reicht es aber meist Objekte in anderen Formaten, wie z.B. im PLY-Format (z.B. der Stanford-Bunny) mit einem Programm wie Meshlab[5] in ein bereits implementiertes Format wie OBJ umzuwandeln.

Als Ausgabeformat hat sich VTK bewährt, welches für Meshes aus Dreiecken ein einfaches Polygon-Format unterstützt und Modelle aus Tetraedern mit Hilfe eines sog. „unstructured grid“ abbilden kann. Beide Formate sind den internen Datenstrukturen der DEC Objekte ähnlich, da sie eine Liste mit Vertices und eine Liste mit Simplices verwenden. Auch hier ist eine Erweiterung z.B. für OBJ, PLY, STL und ähnliche Formate einfach möglich, sollte sie zum Beispiel benötigt werden damit die Ausgabe von einem anderem Programm weiterverarbeitet werden kann.

[1] `http://www.paraview.org/`
[2] Eins der unterstützten Formate der Software tetgen (`http://wias-berlin.de/software/tetgen/`)
[3] `http://www.comsol.com/`
[4] `http://graphics.stanford.edu/data/3Dscanrep/`
[5] `http://meshlab.sourceforge.net/`

Tabelle 4.1: Parameter für die Wärmeleitungsgleichung auf dem Stanford-Bunny

Modell	Stanford-Bunny (2503 Punkte, 4968 Dreiecke)
Dimension	2D-Mesh der Oberfläche
Minimale Kantenlänge	0.0010582 m
Zeitschrittweite	$\Delta_t = 10^{-7}$ s
Simulierte Zeit	$T = 0.1$ s
Zeitschritte	1000000
Randbedingungen	161 Punkte $(\mathbf{u_t})_p = 1 \forall t \in [0,T], p \in \{p \vert p_y \in [0, 0.036]\}$
Startwerte	Alle Punkte p ohne Randbedingung haben den Wert $(\mathbf{u_0})_p = 0$

4.1 Wärmeleitung

Um die Wärmeleitung zu berechnen, wie sie in 3.14 diskretisiert wurde, muss man sich bei SI-Einheiten eine Zeitschrittweite in Sekunden, eine Zeit die berechnet werden soll in Sekunden und eine Temperaturleitfähigkeit in $\frac{m}{s}$, bzw. für dreidimensionale Meshs in $\frac{m^2}{s}$ vorgeben. Die Länge der Kanten wird dann in Meter gemessen. Die Temperaturleitfähigkeit nehmen wir in allen Rechnungen als 1 an.

Hier wurden zwei Beispiele gerechnet. Einmal eine Wärmeleitung von der Unterseite des Stanford-Bunnies aus, der ein Standardbeispiel für solche Berechnungen ist, und eine Diffusion auf einer Blase, die aus einer Strömungssimulation stammt, welche ein Beispiel für eine reale Anwendung ist.

Wärmeleitung auf dem Stanford-Bunny

In Abbildung 4.1 ist das Mesh des „Stanford-Bunny“ aus dem 3D-Scan-Repository der Stanford-Universität[6] und die initiale Temperaturverteilung zu sehen, bei der die Punkte auf der unteren Seite des Modells den Wert 1 und alle anderen Punkte den Wert 0 haben.

In der Simulation wurde Gleichung 3.25 für eine iterative Berechnung der Zeitschritte genutzt, ausgehend von einem Vektor $\mathbf{u_0}$ der Startwerte.

In Grafik 4.2 ist das Ergebnis der Wärmeleitungsrechnung nach zu unterschiedlichen Zeitschritten zu sehen.

Wärmeleitung auf einer Gasblase

Ein interessantes Beispiel ist die Berechnung solcher Gleichungen auf physikalisch korrekten Objekten. Dazu wurde das Mesh einer aufsteigenden Blase aus einer Strömungsrechnung (siehe [CGS09]) als Eingabe verwendet, um auf der Oberfläche die Gleichung

[6] `http://graphics.stanford.edu/data/3Dscanrep/`

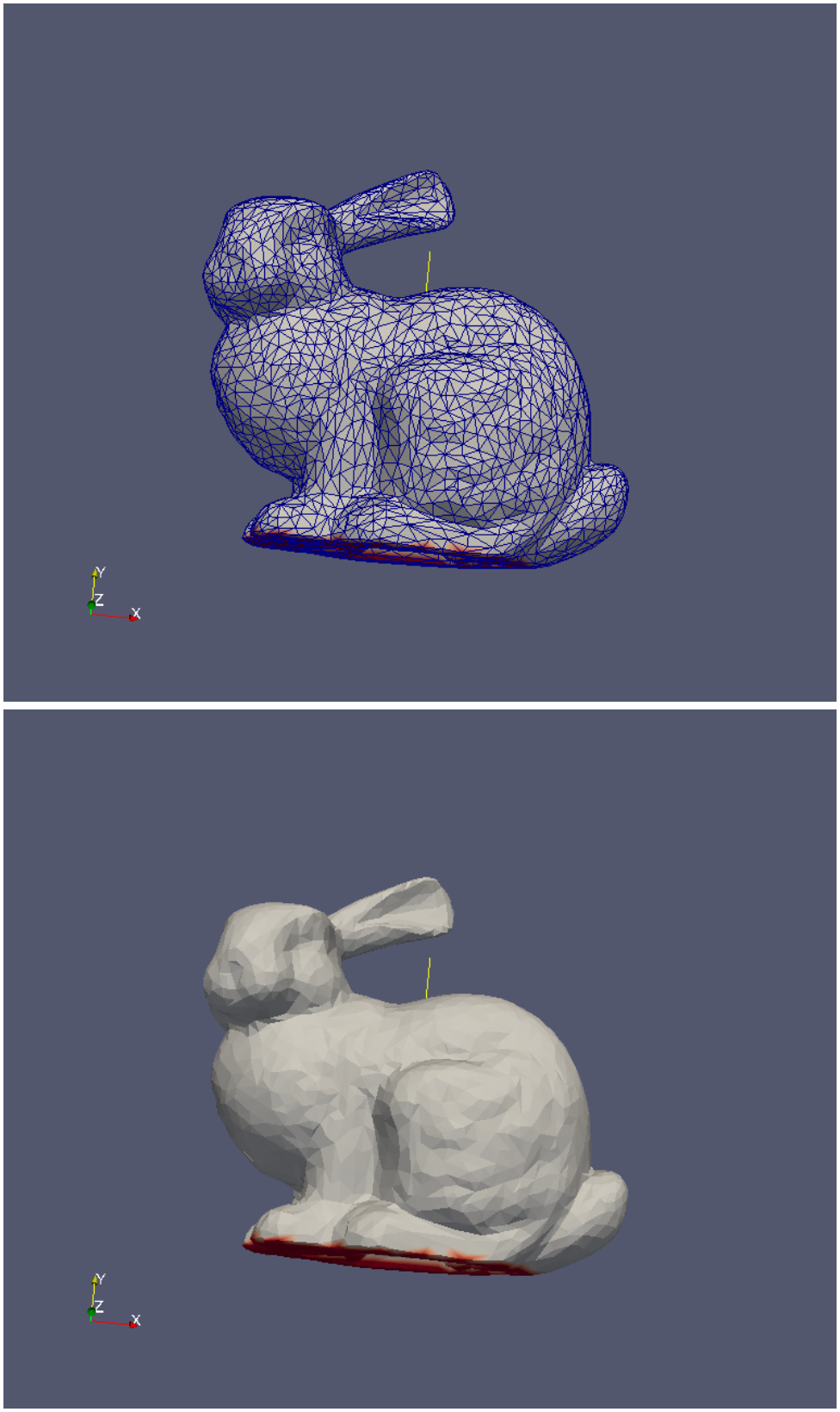


Abb. 4.1: Das Mesh des Bunnies mit der initialen Wärmeverteilung.

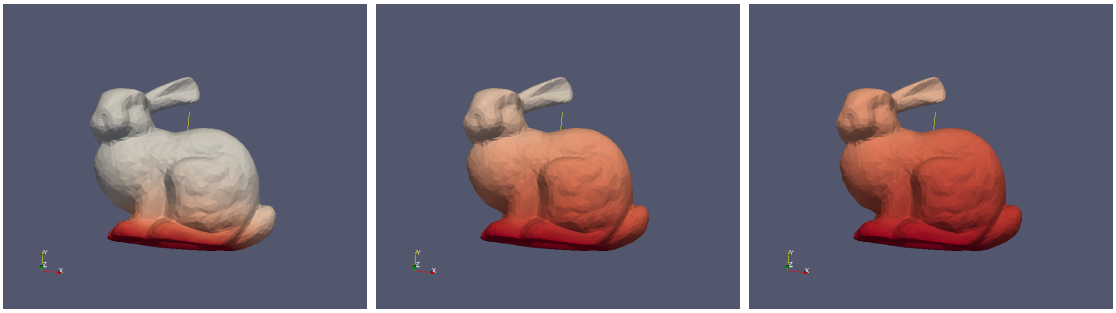


Abb. 4.2: Die Wärmeverteilung auf dem 3D-Modell des Stanford-Bunnies

Tabelle 4.2: Parameter für die Rechnung auf der Blase

Modell	Blase aus einer Strömungsrechnung (2232 Punkte, 4460 Dreiecke)
Dimension	2D-Mesh der Oberfläche
Minimale Kantenlänge	0.008531534 m
Zeitschrittweite	$\Delta_t = 10^{-5}$ s
Simulierte Zeit	$T = 0.1$ s
Zeitschritte	10000
Startwerte	1132 Punkte mit Wert 1, 1099 Punkte mit Wert 0

zu lösen. Das Ergebnis einer solchen Rechnung mit den Parametern aus Tabelle 4.2 ist in Grafik 4.3 zu sehen.

4.2 Wärmeleitung auf Tetraedermeshes

Die Wärmeleitung lässt sich auch auf Tetraedermeshes berechnen. Das Mesh in Grafik 4.4 ist ein Beispielobjekt aus dem Physik-Programm *Comsol*, welches in ein Tetraedermesh mittlerer Auflösung exportiert wurde. Die Wärmeleitung kann mit dem DEC in exakt der gleichen Weise gerechnet werden, wobei sich hier die Wärme auch durch das Innere des Tetraedergitters ausbreitet.

4.3 Nicht gleichverteilte Wärmeleitfähigkeit

Eine nicht uniforme Wärmeleitung lässt sich über eine Gewichtung der Punkte bzw. Kanten bei der Berechnen des Laplaceoperators erreichen. Im Fall der unterschiedlichen Wärmeleitung auf den Punkten, wird der Parameter α der Wärmeleitfähigkeit als Funktion der Punkte modelliert, welche im DEC als eine Co-Chain, die ein Vektor aus $\mathbb{R}^{|K^0|}$ ist, abgebildet wird.

In Grafik 4.5 ist ein Beispiel einer Berechnung der Wärmeleitung zu sehen, bei dem ein Einheitsquadrat als Randbedingung auf dem Rand einen Wert von 1 hat und auf allen

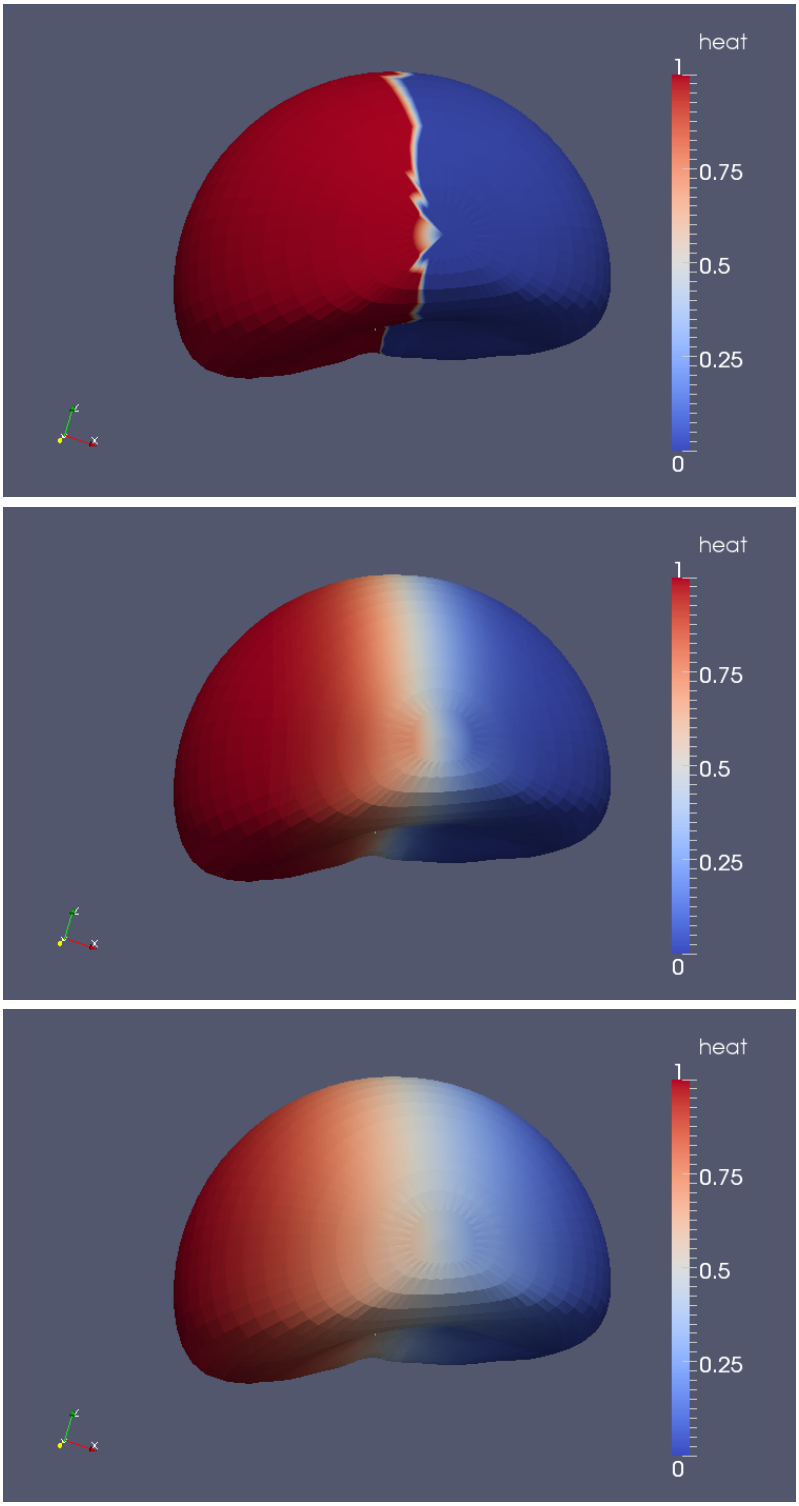


Abb. 4.3: Verschiedene Zeitschritte der simulierten Wärmeleitungsgleichung auf einem dreidimensionalem Oberflächenmodell aus dem Ergebnis einer Strömungsrechnung. Die gezeigten Bilder sind aus den Zeitschritten $0s$, $0.003s$ und 0.01.

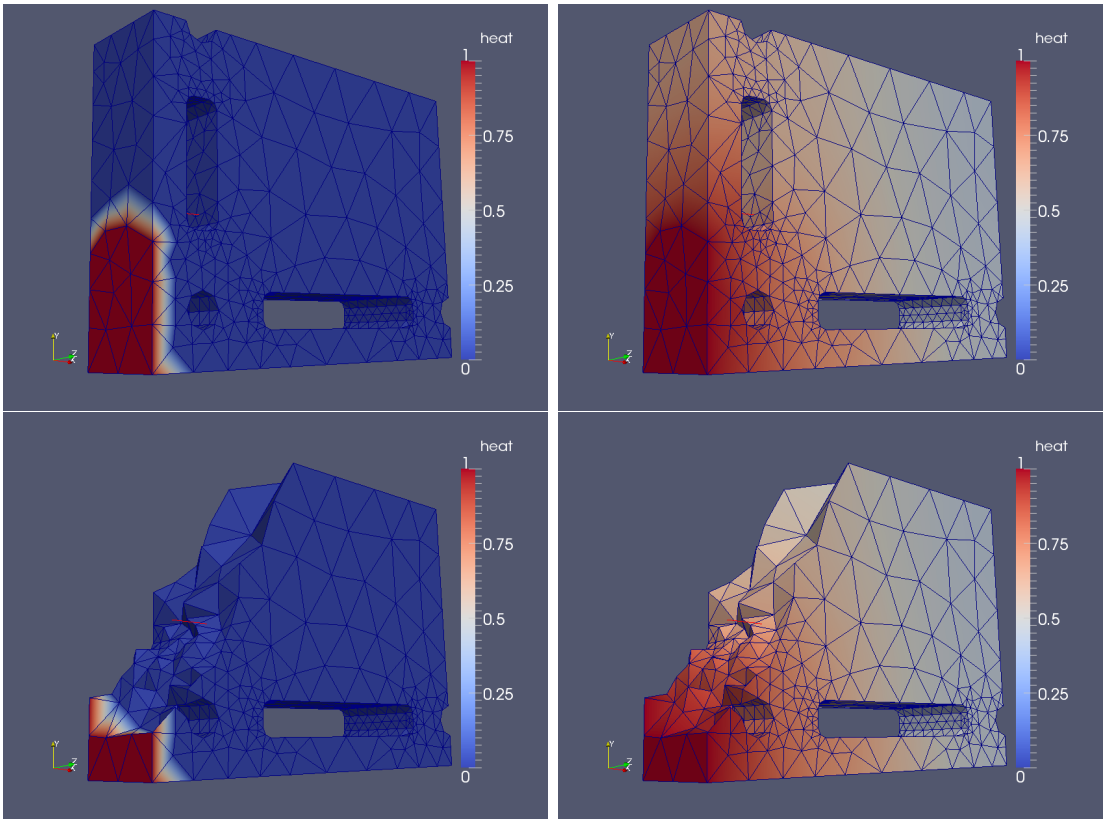


Abb. 4.4: Ein Beispielobjekt aus Comsol, auf dessen Tetraedermesh die Wärmeleitungsgleichung gerechnet wurde. Links: Zeitschritt 0 mit der Anfangsbedingung, rechts: Zeitschritt 150. In der unteren Zeile ist jeweils ein Schnitt durch das Modell gezeigt.

Tabelle 4.3: Die Parameter der Rechnung auf dem 3D-Mesh

Modell	Metallplatte mit Bohrungen (1235 Punkte, 4464 Tetraeder)
Dimension	3D-Mesh des Objekts
Minimale Kantenlänge	$1.09186 \cdot 10^{-6}$ m
Maximale Kantenlänge	$2.80756 \cdot 10^{-4}$ m
Zeitschrittweite	$\Delta_t = 10^{-6}$ s
Simulierte Zeit	$T = 0.1$ s
Zeitschritte	100000
Randbedingungen	16 Punkte $(\mathbf{u_t})_p = 1\ \forall t \in [0, T]$, $p \in \{p \vert p_x \leq \epsilon, p_y \in [0, 0.04]\}$
Startwerte	16 Punkte mit Wert 1, 1219 Punkte mit Wert 0

anderen Vertices den Startwert 0.

Nach einigen Zeitschritten wird erkennbar, dass sich die Wärme aufgrund der unterschiedlichen Wärmeleitfähigkeit nicht gleichmäßig von außen nach innen ausbreitet, da auf einigen Punkten die Wärmeleitung geringer ist als auf anderen.

Beim DEC wird unterschieden zwischen nicht-linearer Wärmeleitung, welche unterschiedliche Wärmeleitfähigkeit auf den primalen Vertices bedeutet und anisotroper Wärmeleitung, welche eine unterschiedliche Wärmeleitung entlang von Kanten bedeutet.

Im Beispiel wurde die nicht-lineare Wärmeleitung mit Hilfe eines Dual-Primal Laplaceoperator berechnet, welcher den dual-primal-primal Flat-Operator ($\sharp_{dpp}$) zusammen mit einem dual-primal Gradienten verwendet, wie er in [Hir03] definiert ist, welcher es auch ermöglicht über eine 1-Co-Chain von Kantengewichten mit einer anisotrope Wärmeleitung zu rechnen.

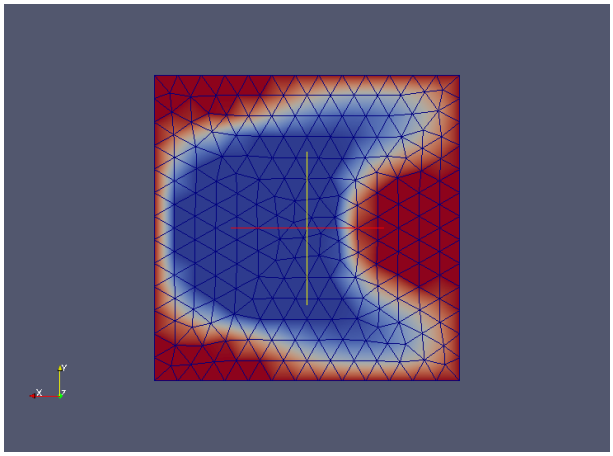


Abb. 4.5: Das Ergebnis der Wärmeleitungsgleichung mit unterschiedlicher Wärmeleitfähigkeit auf verschiedenen Punkten. Die Randpunkte haben eine konstante Temperatur, welche sich auf dem Gitter unterschiedlich schnell ausbreitet.

4.4 Krümmung der Einheitskugel

Als Test für den Fehler und die Konvergenz des numerischen Ergebnisses der Krümmungsberechnung, wie wir sie in Abschnitt 3.14 definiert haben, wurde die Krümmung der Einheitskugel mit unterschiedlicher Auflösung berechnet.

Die Modelle, die als Eingabe gedient haben, sind in Grafik 4.6 zu sehen. Sie wurden mit Comsol generiert und dann als STL-Datei abgespeichert, welche mittels Meshlab in das OBJ-Format umgewandelt wurde.

Für die Einheitskugel ist die analytische Lösung für die Krümmung $\kappa = \frac{1}{R} = 1$ auf allen Punkten der Kugel bekannt. Also können wir den Fehler der berechneten Krümmung als $\|\mathbf{1} - \boldsymbol{\kappa_d}\|_\infty$ berechnen, wobei die beiden Vektoren aus $\mathbb{R}^{|K^0|}$ den Wert der Krümmung

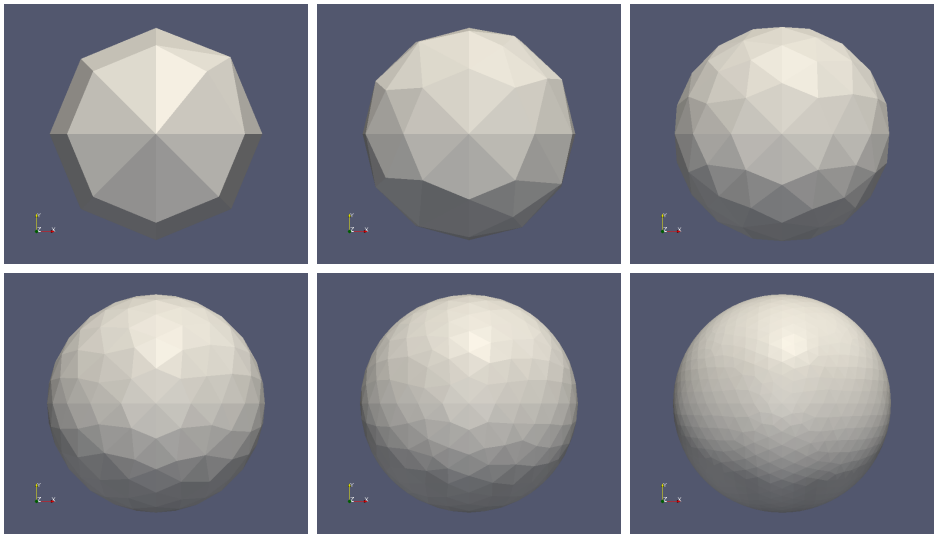


Abb. 4.6: Die Einheitskugel mit $R = 1$ in unterschiedlich feiner Auflösung

Tabelle 4.4: Die Fehler der berechneten Krümmung im Verhältnis zur Kantenlänge und dem Flächeninhalt der Faces

# Punkte	# Dreiecke	Länge der maximalen Kante [m]	Maximaler Flächeninhalt [m^2]	Fehler: $\|\boldsymbol{\kappa} - \boldsymbol{\kappa_d}\|_\infty$
28	52	1.000009	0.299707	0.008835
64	124	0.659659	0.127125	0.001212
128	252	0.392893	0.063976	0.000428
234	464	0.318179	0.036598	0.000391
428	852	0.234191	0.020655	0.000243
656	1308	0.199928	0.013559	0.000169
1408	2812	0.129933	0.006233	0.000176

auf allen Punkten des Modells enthalten. Die Konvergenz des Verfahrens wird über die Abhängigkeit des Fehlers von der Kantenlänge bzw. der Fläche der Faces gemessen.

Die Fehler der Krümmungsberechnung mit dem DEC-Laplace für unterschiedliche Auflösung des 3D-Modells der Kugel sind in Tabelle 4.4 aufgelistet und der Konvergenzplot des Fehlers bei verschiedenen Auflösungen des Modells ist in Grafik 4.7 abgebildet.

4.5 Darcy-Flow

Das Ergebnis einer Berechnung des Darcy-Flow auf einem Einheitsquadrat, welches mit 124 Dreiecken diskretisiert wurde ist in Grafik 4.8 zu sehen. Die analytische Lösung für die Quellen/Senken ist gegeben durch $\cos(\pi x)\cos(\pi y)$, das numerische Ergebnis wurde

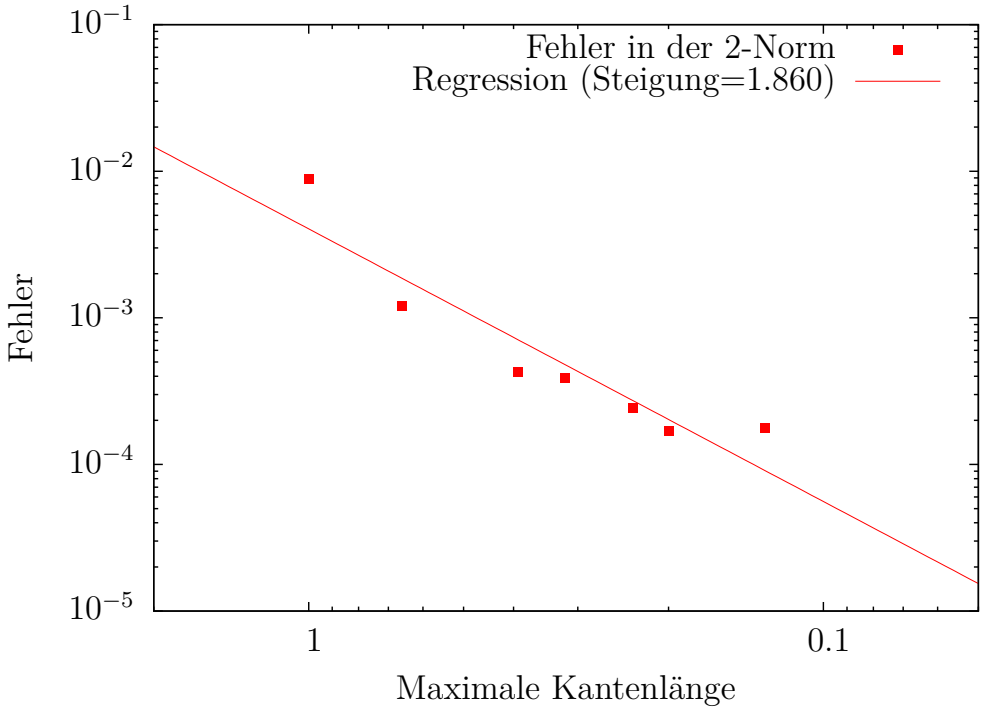


Abb. 4.7: Konvergenz der Krümmungsberechnung auf der Einheitskugel

wie in Abschnitt 3.15 beschrieben mit Hilfe des DEC berechnet. Die Ergebnisse für den Druck sind in jedem Dreieck konstant, die Flussvektoren wurden über den Fluss entlang der Normalen der Randkanten auf dem Barycenter der Dreiecke interpoliert.

Die Randbedingungen sind in diesem Beispiel:

$$p_i = \cos(\pi x_c)\cos(\pi y_c) \tag{4.1}$$

$$\nabla p_i = \begin{pmatrix} -\pi\cos(\pi x_c)\cos(\pi y_c) \\ -\pi\cos(\pi x_c)\sin(\pi y_c) \end{pmatrix} \tag{4.2}$$

$$\phi_i = 2\pi^2\cos(\pi x_c)\cos(\pi y_c) \tag{4.3}$$

$$\omega_i = \mathrm{vol}(f_i) \tag{4.4}$$

$$\gamma_i = \mathrm{vol}(e_i) \tag{4.5}$$

$$f_i = \left(-\frac{k}{\mu}\nabla p_i\right)\cdot\hat{n}\gamma_i \tag{4.6}$$

Dabei sind die Koordinaten x_c, y_c jeweils auf dem Circumcenter des Dreiecks bzw. der Kante definiert. Der Druck p_i ist auf jeder Zelle (Tetraeder bzw. Dreieck) definiert und der Flussvektor f_i auf den Randfaces der Zellen (Kanten bzw. Dreiecke).

Die Fehler des Flusses in den Beispiel-Grafiken sind in der 2-Norm angegeben, die des Drucks in der 1-Norm. In den Tabellen 4.6 und 4.5 sind die Fehler für Gitter verschiedener

Tabelle 4.5: Der Fehler des berechneten Drucks

Min. Kantenlänge	Max. Kantenlänge	1-Norm	2-Norm	∞-Norm
0.312	0.523	0.364	0.399	0.565
0.172	0.32	0.101	0.104	0.139
0.114	0.206	$7.678 \cdot 10^{-3}$	$9.698 \cdot 10^{-3}$	$2.501 \cdot 10^{-2}$
$9.776 \cdot 10^{-2}$	0.181	$6.056 \cdot 10^{-3}$	$7.593 \cdot 10^{-3}$	$1.868 \cdot 10^{-2}$
$6.172 \cdot 10^{-2}$	0.116	$5.345 \cdot 10^{-3}$	$6.164 \cdot 10^{-3}$	$1.401 \cdot 10^{-2}$
$5.335 \cdot 10^{-2}$	$9.596 \cdot 10^{-2}$	$3.239 \cdot 10^{-3}$	$3.729 \cdot 10^{-3}$	$8.798 \cdot 10^{-3}$
$3.553 \cdot 10^{-2}$	$6.753 \cdot 10^{-2}$	$3.82 \cdot 10^{-3}$	$4.105 \cdot 10^{-3}$	$7.815 \cdot 10^{-3}$
$2.389 \cdot 10^{-2}$	$4.224 \cdot 10^{-2}$	$1.251 \cdot 10^{-3}$	$1.434 \cdot 10^{-3}$	$3.513 \cdot 10^{-3}$
$1.277 \cdot 10^{-2}$	$2.568 \cdot 10^{-2}$	$8.79 \cdot 10^{-4}$	$9.6 \cdot 10^{-4}$	$2.05 \cdot 10^{-3}$

Tabelle 4.6: Der Fehler des berechneten Flusses

Min. Kantenlänge	Max. Kantenlänge	1-Norm	2-Norm	∞-Norm
0.312	0.523	0.658	0.766	1.441
0.172	0.32	0.287	0.36	0.857
0.114	0.206	0.188	0.232	0.564
$9.776 \cdot 10^{-2}$	0.181	0.16	0.198	0.417
$6.172 \cdot 10^{-2}$	0.116	0.114	0.139	0.303
$5.335 \cdot 10^{-2}$	$9.596 \cdot 10^{-2}$	$8.798 \cdot 10^{-2}$	0.109	0.225
$3.553 \cdot 10^{-2}$	$6.753 \cdot 10^{-2}$	$6.401 \cdot 10^{-2}$	$7.924 \cdot 10^{-2}$	0.184
$2.389 \cdot 10^{-2}$	$4.224 \cdot 10^{-2}$	$3.984 \cdot 10^{-2}$	$4.917 \cdot 10^{-2}$	0.125
$1.277 \cdot 10^{-2}$	$2.568 \cdot 10^{-2}$	$2.313 \cdot 10^{-2}$	$2.851 \cdot 10^{-2}$	$6.428 \cdot 10^{-2}$

Auflösung aufgetragen und die Konvergenzplots mit 1-, 2- und Maximumsnorm sind in Grafik 4.9 zu sehen.

Eine Darstellung der Ergebnisse ist in Grafik 4.8 abgebildet. Auf der linken Seite sind jeweils die Ergebnisse für den Fluss, der auf dem Barycenter des Dreiecks interpoliert wurde, zu sehen und auf der rechten die Ergebnisse für den Druck, welcher innerhalb eines Dreiecks als konstant angenommen wird. Die oberste Zeile der Grafik zeigt die analytische Lösung, in der mittleren Zeile sind die numerischen Ergebnisse zu sehen, und unten ist in schwarz-weiß der Fehler aufgetragen.

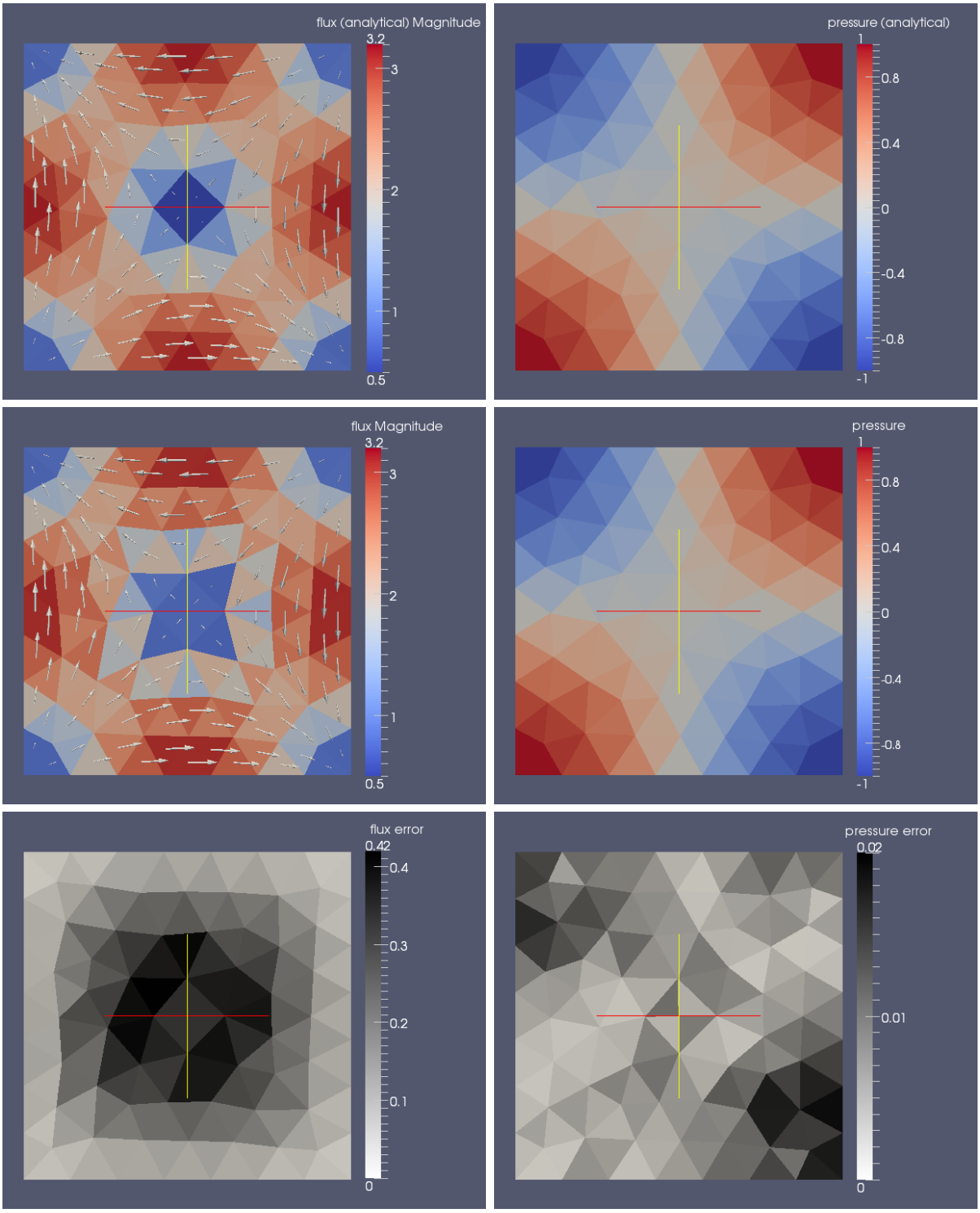


Abb. 4.8: Das Ergebnis einer numerischen Berechnung der Darcy-Flow-Gleichung auf einem Einheitsquadrat, welches mit 124 Dreiecken trianguliert wurde.

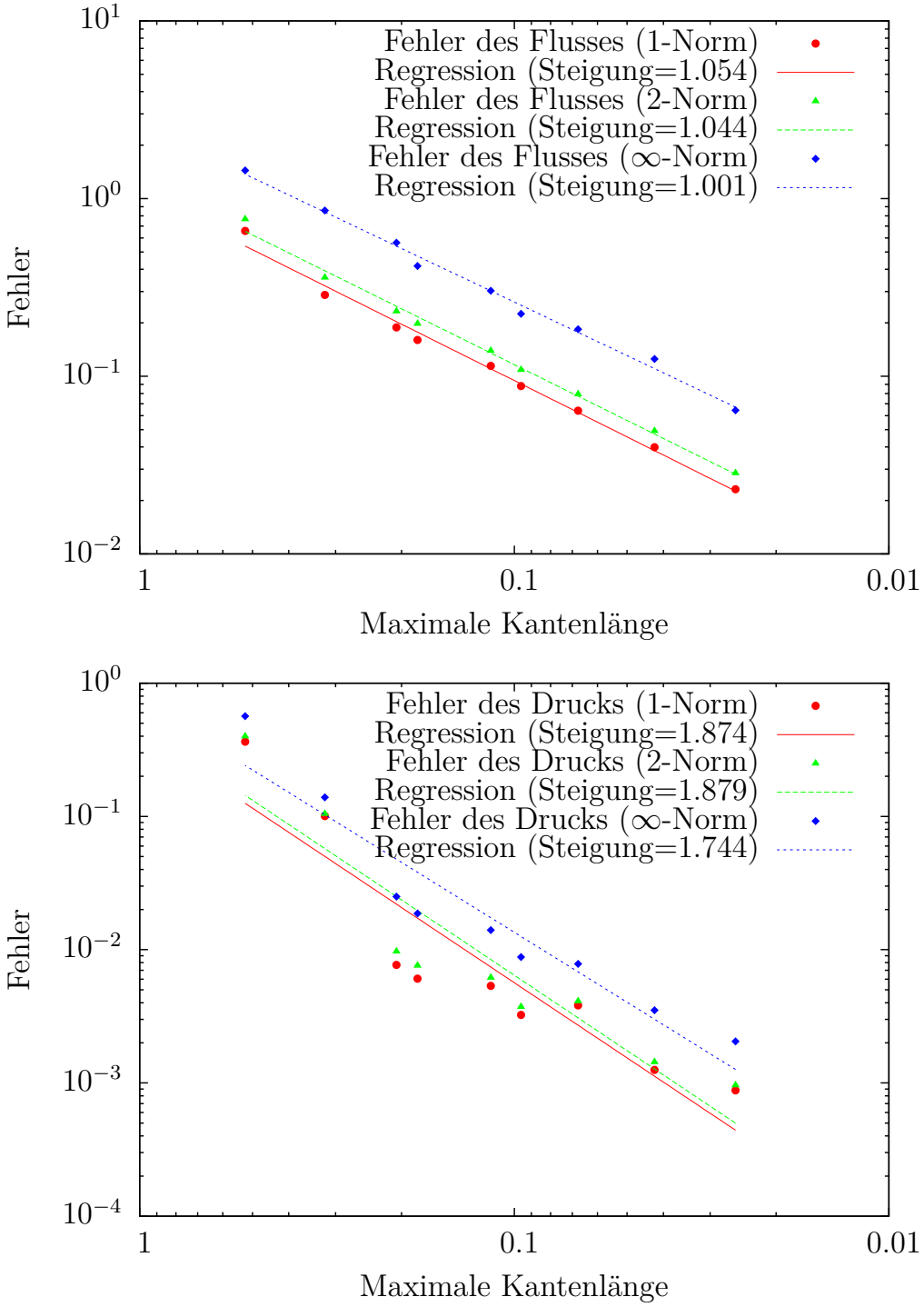


Abb. 4.9: Die Konvergenz der numerischen Berechnung des Flusses und Drucks in der numerisch berechneten Darcy-Gleichung

5 Maschinelles Lernen

5.1 Einleitung

Im maschinellem Lernen geht es darum, dass eine Funktion approximiert werden soll, indem eine Ansatzfunktion mit Hilfe von vorhandenen Trainingsdaten möglichst gut parametrisiert wird.

Dabei sollen nicht nur die bekannten Trainingsdaten gut getroffen werden, sondern die Funktion soll möglichst gut die Werte von Punkten approximieren, die während des Trainings noch nicht bekannt waren.

Zu diesem Zweck ist es oft sinnvoll die Trainingspunkte weniger genau zu approximieren, wenn die Funktion dafür einen kleineren Fehler auf den gesamten Punkten hat.

Die Idee hierbei ist, dass die Funktion hinter den Werten auf den Trainingspunkten einfach genug ist, dass sie aus den Beispielen gelernt werden kann, sodass man mit verhältnismäßig wenig bekannten Daten eine Funktion findet, die auf möglichst vielen Daten eine gute Approximation für den Funktionswert liefert.

5.2 Konzepte

Wir betrachten nun einige beim maschinellem Lernen verwendete Konzepte.

Ein häufig verwendetes Konzept ist das *supervised learning*, bei dem Punkte und die zugehörigen Funktionswerte als „Musterlösung“ gegeben sind. Mit Hilfe dieser Trainingsdaten optimiert der Algorithmus eine Ansatzfunktion, um eine Funktion zu bekommen, welche möglichst korrekte Werte auf den Punkten berechnet. Insbesondere soll sie auch für Punkte mit unbekanntem Wert eine Approximation des Funktionswertes mit möglichst geringem Fehler liefern.

Das *unsupervised learning* ist eine Methode, ohne bekannte Trainingsdaten Muster in Daten zu erkennen und zum Beispiel bei hochdimensionalen Datenpunkten die intrinsische Dimension, welche oft deutlich niedriger ist, zu erkennen. Dabei sind zum Training nur Datenpunkte ohne zugehörigen Funktionswert gegeben. Eine Beispielanwendung ist Clustering, bei dem ein Algorithmus die Punkte sinnvoll gruppieren soll. Nachdem der Algorithmus durchgelaufen ist, können die Klassen benannt werden, indem für jede Gruppe anhand einiger Punkte analysiert wird, welche gemeinsamen Eigenschaften die Punkte der Gruppe haben. Bei manchen Anwendungen ist es sogar nur nötig ähnliche Punkte zu einem gegebenen Punkt aus dem Cluster zu finden, ohne sie genau benennen zu können.

Das *semi-supervised learning* ist eine Kombination aus den ersten beiden Ansätzen. Es sind eine Reihe von Punkten mit bekannter Referenzlösung gegeben, und zusätzlich

einige Punkte, bei denen bekannt ist, dass die Funktion auf diesen Punkten definiert ist, zu denen aber keine Referenz gegeben ist. Aus diesen Punkten können Informationen über die Nachbarschaft oder Cluster der Daten gewonnen werden, was dabei helfen kann die Trainingspunkte mit bekanntem Wert besser einzuordnen. Dies ist zum Beispiel nützlich bei Daten, die in Clustern liegen aus welchen jeweils nur wenige Trainingswerte bekannt sind. Dort helfen die Punkte ohne bekannten Wert die Grenze des Clusters zu finden, was es erleichtert auch mit wenigen Punkten, die einen Trainingswert haben, die Funktion zu erlernen. Das bedeutet, dass Punkte innerhalb eines Cluster einen ähnlichen Wert annehmen sollen, während sich der Wert auf Punkten aus verschiedenen Clustern stark unterscheiden darf.

Eine weitere Methode, welche hier nicht weiter vertieft werden soll, ist das *reinforcement learning*. Hierbei wird über eine Kostenfunktion nur eine Bewertung des gelernten Ansatzes als Feedback gegeben. Der Algorithmus nutzt die „Bestärkung“ bzw. „Bestrafung“ des Ansatzes als Ausgangspunkt für den nächsten Ansatz. Damit können Paare aus Zustand und Aktion gelernt werden, zum Beispiel für einen Computergegner bei Brettspielen wie Backgammon [SB98].

Methoden

Einige der gängigsten Methoden für diese Ansätze sind:

- **supervised:** Perzeptron mit Backpropagation of Error [GD98].
- **supervised:** Radiale Basisfunktionen (RBF), z.B. lösbar mit einer Pseudoinverse [PS91].
- **unsupervised:** k-Means [HW79].
- **unsupervised:** self-organizing maps [Koh90].
- **reinforcement:** dynamische Programmierung [Bel86].
- **reinforcement:** genetische/evolutionäre Algorithmen [GH88].
- **supervised / semi-supervised:** Lernen mit Kernen [Aro50], [BN04].

Im Folgenden werden Kern-Methoden zum (semi-)supervised Lernen verwendet. Viele der anderen Methoden lassen sich mit passend gewählten Kernen abbilden, so entspricht ein RBF-Netz gerade einem Gaußkern, welcher als radiale Basisfunktion auf jedem der Datenpunkte ausgewertet wird.

5.3 Lernen mit Kernen

Um das semi-supervised learning mit Kernen zu definieren wird nun zunächst die Methode des überwachten (supervised) Lernens mit Kernen betrachtet, um diese dann zu einem semi-supervised Verfahren zu erweitern.

Einführende Literatur zu Lernverfahren mit Kernen ist unter anderem [Sch06], [HSS05], [Dau04] und [Aro50].

Die hier verwendeten Definitionen zu Kernen und die Notation stimmen weitgehend mit [HSS05] überein.

Problemstellung

Gegeben sei eine Menge $\mathcal{X}$ mit einer endlichen Teilmenge $X \subset \mathcal{X}$ an Trainingspunkten $x_i \in X$, ein normierter Vektorraum $\mathcal{Y}$ und eine Menge $Y \subset \mathcal{Y}$ mit $|X| = |Y| = n$ an Trainingswerten y_i, auf die abgebildet werden soll. Die Punkte aus den beiden Mengen sind als Paare von Punkt und Referenzwert $(x_i, y_i), i = 1, \ldots, n$ gegeben. Die Menge $\mathcal{X}$ kann dabei eine beliebige Menge sein, welche keine besonderen Anforderungen erfüllen muss.

Gesucht ist eine Funktion, die von $\mathcal{X}$ nach $\mathcal{Y}$ abbildet und eine möglichst gute Generalisierung auf ungelernten Datenpunkten zeigt.

Die Definition einer „guten“ Verallgemeinerung ist dabei nicht so einfach zu formalisieren, wenn vorher unbekannt ist was für ein Wert auf den neuen Punkten zu erwarten ist. Eine Möglichkeit die gelernte Funktion zu bewerten ist, sie auf eine Menge von zurückgehaltenen Trainingspunkten anzuwenden und den Fehler auf diesen Punkten als Kriterium zu verwenden.

Eine häufige Anwendung ist die Klassifikation von Datenpunkten in zwei Gruppen. Dazu verwenden wir als Menge der Funktionswerte $Y = \{y_i | y_i = \pm 1\} \subset \mathcal{Y} \subset \mathbb{R}$. Dabei ist die Menge $\mathcal{Y}$ die Menge der reellen Zahlen, da die Lösung im allgemeinen nicht exakt 1 oder -1 sein wird und die Klassifikation erfolgt über das Vorzeichen.

In einigen Fällen lässt der Absolutwert der Lösung eine Aussage über die Qualität zu, d.h. ein Wert nahe 0 könnte zum Beispiel heißen dass die Lösung an dieser Stelle unklar ist, während ein Wert von 1.0 bedeutet dass der Punkt eindeutig zur ersten Klasse gehört.

Merkmalsraum

Um eine Klassifikation zu ermöglichen ist es oft sinnvoll, die Punkte in einen höherdimensionalen Raum abzubilden, in dem das Problem einfacher gelöst werden kann. Dazu nutzt man eine Merkmalsabbildung $\mathbf{\Phi}$ (Featuremap), welche die Punkte in einen Merkmalsraum $\mathcal{F}$ (Featurespace) abbildet.

$$\mathbf{\Phi} : \mathcal{X} \to \mathcal{F} \tag{5.1}$$

Für eine allgemeine Menge werden teilweise die Punkte erst durch die Featuremap als

Vektoren modelliert, aber auch für Punkte aus dem $\mathbb{R}^d$ hat der Merkmalsraum $\mathcal{F}$ häufig eine deutlich höhere Dimension als der Raum $\mathcal{X}$, in dem die Datenpunkte liegen und enthält "Merkmale", die durch eine nicht-linearen Abbildung entstehen. Dadurch kann ein passend gewählter Merkmalsraum für ein nicht-lineares Problem eine lineare Trennung der Daten ermöglichen, obwohl die Punkte sich im Definitionsraum nicht linear trennen lassen.

Beispiel

Satz 5.1. *Eine Klassifikation von Punkten in verschiedene Punktemengen durch eine lineare Trennung ist nicht für alle binären Funktionen $f : \mathbb{R}^2 \to \{0, 1\}$ möglich.*

Dass es binäre Funktionen gibt, deren Ausgabe nicht linear getrennt werden kann, haben Minsky und Papert 1969 anhand der XOR-Funktion für das neuronale Netz eines Perzeptron ohne Hidden Layer bewiesen [MS69]. Mehr zum historischem Hintergrund findet sich in [Ber97], eine kurze Motivation zum Lösen des XOR-Problems mit einer Featuremap wird in [Gre13, Abschnitt 2] gegeben.

Wir werden nun beweisen, dass sich die Werte der XOR-Funktion im Raum $[0, 1]^2$ nicht linear trennen lassen, um danach zu zeigen, dass eine Abbildung in einen passend gewählten Merkmalsraum es ermöglicht dort eine lineare Trennung zu finden.

Definition 5.1. *Zwei Teilmengen $A \subseteq V$ und $B \subseteq V$ eines Vektorraums V sind genau dann linear separierbar, wenn gilt:*

$$\begin{aligned}
&\exists\ w_1, \ldots, w_{n+1} \in \mathbb{R},\ \text{sd.} \\
&\sum_{i=1}^{n} w_1 a^i \leq w_{n+1} < \sum_{i=1}^{n} w_i b^i \\
&\forall \mathbf{a} = (a^1, \ldots, a^n) \in A, \mathbf{b} = (b^1, \ldots, b^n) \in B
\end{aligned}$$

Definition 5.2. *Die binäre XOR-Funktion bildet zwei Werte, die 0 oder 1 sind auf 1 ab, wenn sie unterschiedlich sind und sonst auf 0. Es kann auch sinnvoll sein sie mit den Werten 1 und -1 zu definieren, da dann nach Vorzeichen entschieden werden kann.*

$$\begin{aligned}
\mathrm{XOR}_{\text{binär}} &: \{0, 1\}^2 \to \{0, 1\} \\
\mathrm{XOR}_{\text{binär}}(x, x') &= \begin{cases} 1 \text{ wenn } x \neq x' \\ 0 \text{ wenn } x = x' \end{cases} \quad x, x' \in \{1, 0\} \\
\mathrm{XOR}_{\pm 1}(x, x') &= \begin{cases} 1 \text{ wenn } x \neq x' \\ -1 \text{ wenn } x = x' \end{cases} \quad x, x' \in \{1, 0\}
\end{aligned}$$

Hier wird mit der Funktion $\mathrm{XOR} : \{0, 1\}^2 \to \{-1, +1\}$ gerechnet, da mit dieser die Trennung einfacher zu sehen ist. Die Funktion lässt sich leicht wieder auf die binäre Funktion abbilden:

$$\mathrm{XOR}_{\text{binär}}(x, y) = \frac{\mathrm{XOR}_{\pm 1}(x, y) + 1}{2}$$

Im Folgenden ist mit XOR stets $\mathrm{XOR}_{\pm 1}$ gemeint.

Beweis: $\mathrm{XOR} : \{0, 1\}^2 \to \{-1, +1\}$ ist nicht linear separierbar

Seien die Mengen V, A und B definiert als

$$\begin{aligned} V &:= \{(0,0), (1,1), (1,0), (0,1)\}, \\ V \supseteq A &:= \{\mathbf{a_1} = (0,0), \mathbf{a_2} = (1,1)\}, \\ V \supseteq B &:= \{\mathbf{b_1} = (1,0), \mathbf{b_2} = (0,1)\}. \end{aligned}$$

Angenommen A und B wären linear separierbar, dann gilt:

$$\sum_{i=1}^{2} w_i a_j^i \leq w_3 < \sum_{i=1}^{2} w_i b_j^i \qquad \forall a_j \in A, b_j \in B \tag{5.2}$$

$$\sum_{i=1}^{2} w_i a_1^i = 0 \leq w_3 < w_1 = \sum_{i=1}^{2} w_i b_1^i \qquad \Rightarrow 0 < w_1 \tag{5.3}$$

$$\sum_{i=1}^{2} w_i a_2^i = w_1 + w_2 \leq w_3 < w_2 = \sum_{i=1}^{2} w_i b_2^i \qquad \Rightarrow w_1 < 0 \tag{5.4}$$

$$w_1 < 0 < w_1 \Rightarrow \text{ Widerspruch} \tag{5.5}$$

□

Satz 5.2. *Nach der Anwendung einer nicht-linearen Featuremap ist die XOR-Funktion im Merkmalsraum linear separierbar.*

Sei $\mathbf{\Phi}$ eine Featuremap:

$$\begin{aligned} \mathbf{\Phi} : \mathbb{R}^2 &\to \mathbb{R}^3 \\ (x, y) &\mapsto (x, y, (x + y)^2) \end{aligned}$$

Dann sind die Mengen $A' = \{\mathbf{\Phi}(a) | a \in A\}$ und $B' = \{\mathbf{\Phi}(b) | b \in B\}$ linear separierbar.

Beweis: A' und B' sind linear separierbar
Seien

$$A' := \{\mathbf{a_1} = (0,0,0), \mathbf{a_2} = (1,1,4)\}, \quad B' := \{\mathbf{b_1} = (1,0,1), \mathbf{b_2} = (0,1,1)\}, \quad V' = A' \cup B'.$$

Für lineare Separierbarkeit muss dann gelten:

$$\begin{aligned} &\sum_{i=1}^{3} w_i a^i \leq w_4 < \sum_{i=1}^{3} w_i b^i \\ \Leftrightarrow &\sum_{i=1}^{3} w_i a^i - w_4 \leq 0 < \sum_{i=1}^{3} w_i b^i - w_4. \end{aligned}$$

Die w_i definieren eine Ebene, welche die Punkte a_i und b_i trennt:

$$\left\{ x \,\middle|\, \sum_{i=1}^{3} w_i x^i - w_4 = 0 \right\}. \tag{5.6}$$

Wir können nun mit den bekannten Punkten nach $\mathbf{w}$ lösen:

$$\begin{pmatrix} a_1 \\ a_2 \\ b_1 \\ b_2 \end{pmatrix} \mathbf{w} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 4 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \mathbf{w} = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix} \tag{5.7}$$

$\mathbf{w} = (4, 4, -2, 1)$ löst das lineare Gleichungssystem, und die XOR-Funktion kann als Skalarprodukt zwischen $\mathbf{\Phi}(x)$ und den berechneten w_i definiert werden:

$$\begin{aligned} \mathrm{XOR}(x, y) &:= \langle (w_1, w_2, w_3), \mathbf{\Phi}(x) \rangle - w_4 && (5.8) \\ &= \langle (4, 4, -2), \mathbf{\Phi}(x) \rangle - 1 \quad \forall x \in V. && (5.9) \end{aligned}$$

□

Veranschaulichung

In Grafik 5.1 ist zu sehen, dass sich im zweidimensionalem Raum keine Gerade finden lässt, welche die Punkte aus A und B von einander trennt, während sich das Problem im dreidimensionalem Raum mit einer passend gewählten Featuremap durch eine Ebene lösen lässt.

Schwierigkeit des XOR-Problems

Dass das XOR-Problem trotzdem noch eine Herausforderung ist, erkennt man leicht, wenn nicht nur die Punkte aus $\{0, 1\}^2$ getrennt werden sollen, sondern das Problem auf ganz $[0, 1]^2 \subset \mathbb{R}^2$ ausgedehnt wird. Je näher Punkte an $(0.5, 0.5)$ liegen, desto größer müssen die Gewichte in $\mathbf{w}$ werden, wodurch die Lösung numerisch instabiler wird.

Die Schwierigkeit des Problems liegt darin, dass die gesuchte Funktion Datenpunkte, die nah zusammen liegen auf Werte mit großem Unterschied abbilden muss. Weitere Varianten des XOR-Problems, wie zum Beispiel das Symmetrieproblem und das Problem binärer Addition, sind in [Fie93] beschrieben.

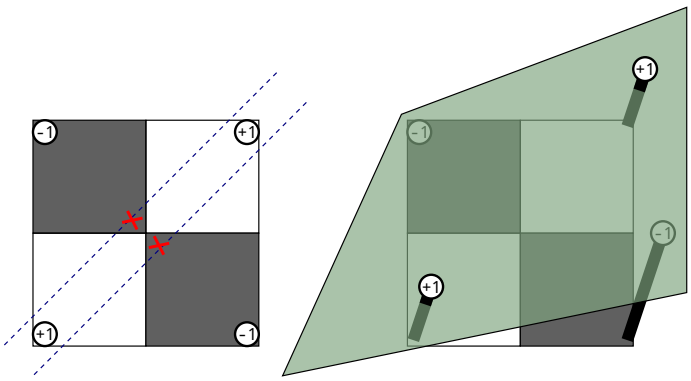


Abb. 5.1: Während das XOR-Problem im Raum der Punkte nicht durch eine lineare Trennung gelöst werden kann, ist die Lösung im Merkmalsraum möglich.

Kerne

Motivation: Der Kernel-Trick

Der sog. „Kernel-Trick“ besteht darin, dass das Skalarprodukt in einem Merkmalsraum, der häufig hochdimensional (oder sogar unendlichdimensional) ist, äquivalent zur Auswertung einer Kernfunktion ist [Dau04, Abschnitt 6.5]. Dadurch kann auf das Berechnen des Skalarprodukts im Merkmalsraum verzichtet werden und stattdessen die Kernel-Funktion ausgewertet werden, deren Berechnung wesentlich effizienter ist.
Beispiel: Der Kern $\exp(-\|x-y\|^2)$ entspricht einer Featuremap in einen unendlichdimensionalen Merkmalsraum, kann jedoch im Raum, in welchem die Punkte liegen, mit der Exponentialfunktion ausgewertet werden, sodass das Skalarprodukt im Merkmalsraum nicht berechnet werden muss.

Kernfunktionen

Wir definieren Kernfunktionen analog zu den Definitionen und die nötigen Voraussetzungen analog zu [Gre13].

Definition 5.3. *Sei V ein Vektorraum über $\mathbb{R}$. Dann ist eine Funktion $\langle\cdot,\cdot\rangle_V : V\times V \to \mathbb{R}$ ein Skalarprodukt in diesem Raum, wenn für $f, g \in V$ die folgenden Gleichungen gelten*

- $\langle f, g\rangle_V = \langle g, f\rangle_V$ *(symmetrisch)*
- $\langle f, f\rangle_V \geq 0$, *und aus* $\langle f, f\rangle_V = 0$ *folgt* $f = \mathbf{0}$ *(positiv definit)*
- $\langle af, g\rangle_V = a\langle f, g\rangle_V = \langle f, ag\rangle_V \ \forall a \in \mathbb{R}$ *(bilinear)*
- $\langle f_1 + f_2, g\rangle = \langle f_1, g\rangle + \langle f_2, g\rangle$ *und* $\langle f, g_1 + g_2\rangle = \langle f, g_1\rangle + \langle f, g_2\rangle$ *(bilinear)*
- $|\langle f, g\rangle|^2 \leq \langle f, f\rangle \cdot \langle g, g\rangle$ *(Cauchy-Schwarzsche Ungleichung).*

Mit dem Skalarprodukt $\langle f, f \rangle_V$ lässt sich die Norm $\|f\|_V = \sqrt{\langle f, f \rangle_V}$ definieren. Mit diesem Skalarprodukt und der dadurch induzierten Norm bildet der Abschluss des Raums V einen Hilbertraum $\mathcal{V}$, in dem die Grenzwerte aller Cauchy-Folgen enthalten sind.

Definition 5.4. *Eine Funktion $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ ist eine Kernfunktion, wenn es einen Hilbertraum $\mathcal{V}$ gibt, und eine Abbildung $\Phi : \mathcal{X} \to \mathcal{V}$, sodass für alle $x, x' \in \mathcal{X}$ gilt:*

$$k(x, x') := \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} . \tag{5.10}$$

Eigenschaften von Kernfunktionen

Für eine Kernfunktion $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ gilt:

- Sie ist positiv linear: αk mit $\alpha \in \mathbb{R}^+$ und $k_1 + k_2$ sind Kerne.
- Wenn $k_1(x, x) - k_2(x, x) \geq 0 \quad \forall x \in \mathcal{X}$, dann ist $k_1 - k_2$ ein Kern[1].
- Wenn $\mathcal{X}$ und $\mathcal{X}'$ zwei Mengen mit einer Abbildung $A : \mathcal{X} \to \mathcal{X}'$ sind und k ein Kern von $\mathcal{X}'$ ist, dann ist $k(A(x), A(x'))$ mit $x, x' \in \mathcal{X}$ ein Kern auf $\mathcal{X}$.
- Wenn k_1, k_2 Kerne auf $\mathcal{X}_1, \mathcal{X}_2$ sind, dann ist $k_1 \times k_2$ ein Kern auf $\mathcal{X}_1 \times \mathcal{X}_2$. Ist $\mathcal{X}_1 = \mathcal{X}_2 = \mathcal{X}$, so ist $k_1 \times k_2$ auch ein Kern. Diese Eigenschaft wird in [Gre13, Formel 3.1] bewiesen.

Kernfunktionen arbeiten häufig auf Mengen $\mathcal{X} \subseteq \mathbb{R}^n$, können aber im Allgemeinen auch für beliebige Mengen $\mathcal{X}$ definiert werden, solange sie die oben genannten Eigenschaften erfüllen.

Beispiele für Kernfunktionen

Im Folgenden seien $\mathbf{x}, \mathbf{x}' \in \mathcal{X} = \mathbb{R}^2$.

In den Beispielen halten wir den Wert $\mathbf{x} = (1, 1)$ fest und zeigen den Graphen zur Funktion $k(\cdot, x) : \mathcal{X} \to \mathbb{R}$, mit $\mathcal{X} \subset \mathbb{R}^2$.

Die Beispielkerne stammen aus der Liste verschiedener Kerne [Sou10].

Der lineare Kern

$$k(\mathbf{x}, \mathbf{x}') := \alpha \langle \mathbf{x}, \mathbf{x}' \rangle + c \quad \alpha, c \in \mathbb{R} \tag{5.11}$$

Der lineare Kern ist eine einfache Kernfunktion, welche häufig der ursprünglichen Funktion entspricht.

Mit einem solchem linearem Kern kann zum Beispiel eine trennende Hyperebene gefunden werden für ein Problem, bei dem keine besondere Featuremap zum Lösen nötig ist, wie beispielsweise bei der binären UND-Funktion.

[1] Die Bedingung folgt aus der Voraussetzung $\langle f, f \rangle_H \geq 0$ für das Skalarprodukt

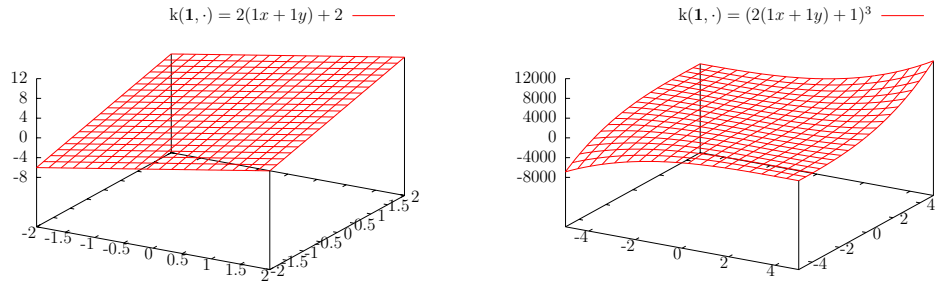


Abb. 5.2: **Links:** Der Wert eines linearen Kerns für $k(\mathbf{1}, \cdot)$ **Rechts:** Der Wert eines Polynomkerns für $k(\mathbf{1}, \cdot)$

Polynomkern

$$k(\mathbf{x}, \mathbf{x}') := \left(\alpha \langle \mathbf{x}, \mathbf{x}' \rangle + c\right)^d \quad \alpha, c \in \mathbb{R} \tag{5.12}$$

Der Polynomkern entspricht einer Featuremap, welche alle Polynome bis zum Grad d enthält. Für $c = 0$ nennt man den Kernel homogen[Sha09].

Gaußkern

$$k(\mathbf{x}, \mathbf{x}') := \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right), \quad \sigma \in \mathbb{R} \tag{5.13}$$

Der Gaußkern, welcher eine sog. Radial-Basis-Funktion verwendet, entspricht einer Gaußglocke mit Zentrum am Punkt $\mathbf{x}$, und Breite σ.

Dieser Kern funktioniert gut mit Daten, welche in mehreren zusammenhängenden Bereichen liegen.

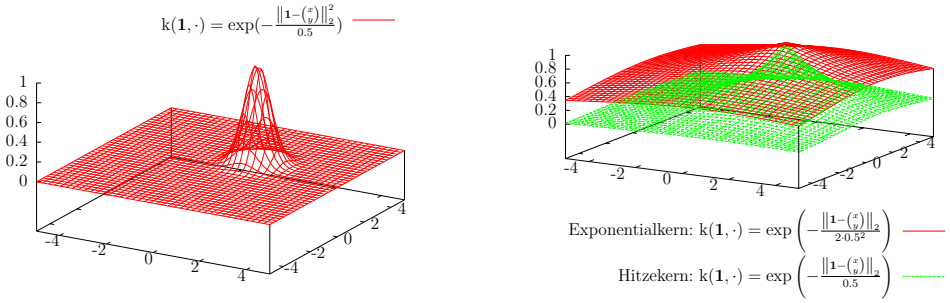


Abb. 5.3: **Links:** Der Wert des Gaußkerns für $k(\mathbf{1}, \cdot)$ **Rechts:** Der Wert des Exponential- und Hitzekerns (ohne zeitabhängigen Faktor) für $k(\mathbf{1}, \cdot)$

Exponential- und Hitzekern

Der Exponentialkern ist definiert als

$$k(\mathbf{x}, \mathbf{x}') := \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{2\sigma^2}\right) \quad \sigma \in \mathbb{R}. \tag{5.14}$$

Dabei sind Exponential- und Hitzekerne ebenfalls radiale Basisfunktionen, wobei der Exponentialkern den Abstand der Punktenorm nicht quadriert, und der Hitzekern zusätzlich dazu durch σ statt durch $2\sigma^2$ teilt. Damit sind beide Kerne dem Gaußkern ähnlich.

Der Hitzekern ist auch als Gauß-Weierstraß-Kern oder Laplacekern bekannt, welcher für einen Zeitpunkt t definiert ist als:

$$k_t(\mathbf{x}, \mathbf{x}') := \frac{1}{(4\pi t)^{n/2}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{4t}\right) \tag{5.15}$$

Dieser Kern heißt Hitzekern, da er der Fundamentallösung der Wärmeleitungsgleichung entspricht, siehe [GHL03].

Sigmoidkern

Der Sigmoid oder auch „Hyperbolic Tangent“ Kern entspricht der Aktivierungsfunktion eines Multilayer Perzeptron [Ros58]. Ein solches Perzeptron modelliert ein biologisches Neuron, welches aktiviert wird wenn die Summe der eingehende Reize einen Schwellwert überschreitet. Bei einer Aktivierung gibt es dann den Reiz weiter.

Um die dabei verwendete Stufenfunktion zu approximieren werden häufig die logistische Funktion $\frac{1}{1+e^{-x}} \in [0, 1]$ [Wei] oder der Tangens Hyperbolicus $\tanh(x) \in [-1, 1]$ verwendet.

Der Sigmoidkern,

$$k(\mathbf{x}, \mathbf{x}') := \tanh(\alpha \langle \mathbf{x}, \mathbf{x}' \rangle + c) \quad \alpha, c \in \mathbb{R}, \tag{5.16}$$

welcher die Tangens Hyperbolicus Funktion verwendet, ist dabei äquivalent zu einem zweischichtigem neuronalem Netz aus Perzeptrons.

Nach [Sou10] ist dieser Kern in der Praxis für viele Aufgaben gut geeignet.

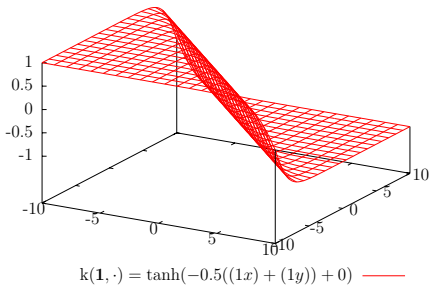


Abb. 5.4: Der Wert des Sigmoidkerns für $k(\mathbf{1}, \cdot)$

Reproduzierende Kern-Hilbert-Räume

Ein reproduzierender Kern-Hilbert-Raum (RKHS) $\mathcal{H}$ entsteht aus einer Kernfunktion $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, und man kann zeigen, dass zu jeder Kernfunktion ein Kern-Hilbert-Raum existiert.

Wir folgen bei der Konstruktion eines RKHS [HSS05].

Ein RKHS $\mathcal{H}$ ist ein Raum von Funktionen $\mathcal{X} \to \mathbb{R}$, welche durch einen Kern induziert werden. Da $\mathcal{H}$ ein Vektorraum ist, enthält er alle Linearkombinationen seiner Elemente, und eine Funktion aus $\mathcal{H}$ ist definiert als

$$f(\cdot) := \sum_{i=1}^{m} a_i k(\cdot, x_i) \in \mathcal{H}, \tag{5.17}$$

für beliebige $m \in \mathbb{N}, a_i \in \mathbb{R}, x_i \in \mathcal{X}$. Der Abschluss des Funktionenraums entsteht durch das hinzunehmen aller Funktionen $f(\cdot) = \sum_{i=1}^{\infty} a_i k(\cdot, x_i)$ mit $\|f\|_{\mathcal{H}} < \infty$.

Das Skalarprodukt von Funktionen $f(\cdot), g(\cdot) \in \mathcal{H}$ ist definiert als

$$f(\cdot) := \sum_{i=1}^{n} a_i k(\cdot, x_i) \in \mathcal{H}, \quad g(\cdot) := \sum_{j=1}^{n'} b_j k(\cdot, x'_j) \in \mathcal{H} \tag{5.18}$$

$$\langle f, g \rangle_{\mathcal{H}} := \sum_{i=1}^{n} \sum_{j=1}^{n'} a_i b_j k(x_i, x'_j) \in \mathbb{R}. \tag{5.19}$$

Aus der Definition folgt direkt, dass das Skalarprodukt symmetrisch und bilinear ist. Weiterhin ist das Skalarprodukt positiv semi-definit, da die Kernfunktion $k(\cdot,\cdot)$ positiv semi-definit ist und daher für alle f mit Koeffizienten $a_i \in \mathbb{R}$ und Datenpunkte $x_i \in \mathcal{X}$ gilt

$$\langle f, f \rangle_{\mathcal{H}} = \sum_{i,j=1}^{n} a_i a_j k(x_i, x_j) \geq 0. \tag{5.20}$$

Damit induziert das Skalarprodukt die Norm

$$\|f\|_{\mathcal{H}}^2 := \langle f, f \rangle_{\mathcal{H}}. \tag{5.21}$$

Eine Konsequenz aus Formel 5.19 ist, dass sich für $f, g \in \mathbb{R}^{\mathcal{X}}$ mit $f = k(\cdot, x)$ die Funktionsauswertung der Funktion g auf dem Punkt x schreiben lässt als

$$g(x) := \sum_{j=1}^{n'} \beta_j k(x, x'_j) = \langle k(\cdot, x), g \rangle_{\mathcal{H}} = \langle f, g \rangle_{\mathcal{H}} \tag{5.22}$$

und damit insbesondere auch gilt

$$\langle k(\cdot, x), k(\cdot, x') \rangle_{\mathcal{H}} = k(x, x'). \tag{5.23}$$

Aufgrund dieser Eigenschaften heißt der Funktionenraum „Reproduzierender Kern-Hilbert-Raum“.

Mit der Cauchy-Schwarzschen Ungleichung für das Skalarprodukt in Formel 5.3 und der reproduzierenden Eigenschaft aus Formel 5.22 gilt

$$|f(x)|^2 = |\langle k(\cdot, x), f \rangle|^2 \leq k(x, x) \cdot \langle f, f \rangle_{\mathcal{H}}. \tag{5.24}$$

Woraus folgt, dass wenn $\langle f, f \rangle_{\mathcal{H}} = 0$ ist, $f(\cdot) = 0$ gelten muss.

Zusammen mit dem Abschluss der Funktionen aus Formel 5.17 ist der Raum $\mathcal{H}$ ein Hilbertraum.

Kernfunktionen als Featuremap

Wir definieren eine Abbildung $\mathbf{\Phi}$, welche Punkte aus der Menge $\mathcal{X}$ auf Funktionen $\mathcal{X} \to \mathbb{R}$ abbildet:

$$\begin{aligned} \mathbf{\Phi} : &\mathcal{X} \to \mathcal{H} \\ &x \mapsto k(\cdot, x) \end{aligned}$$

Diese Abbildung ist eine Featuremap mit dem Funktionenraum $\mathcal{H}$ als Merkmalsraum, wie wir sie in Formel 5.1 definiert haben.

Somit definiert eine Kernfunktion $k(\cdot, \cdot)$ gerade eine Featuremap $\mathbf{\Phi}(x) := k(\cdot, x)$, die einen Punkt $x \in \mathcal{X}$ auf eine Funktion $f(\cdot) \in \mathcal{H}$ abbildet, welche jedem Punkt $x' \in \mathcal{X}$ einen Wert $f(x') = k(x', x) = \langle k(\cdot, x'), k(\cdot, x) \rangle_{\mathcal{H}} \in \mathbb{R}$ zuweist [SHS01].

Wir wollen nun einen Kern verwenden, um eine Funktion $f : \mathcal{X} \to \mathcal{Y}$ aus einer gegebenen Menge von Punkten $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ so zu rekonstruieren, dass der Fehler $\sum_i \|f(x_i) - y_i\|_2^2$ bezüglich dem gewähltem $f \in \mathcal{H}$ minimiert wird.

Das Representer-Theorem

Das Representer-Theorem [SHS01] besagt, dass die Funktion, welche den Fehler auf den Trainingsdaten $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ minimiert, in einem Unterraum von $\mathcal{H}$ liegt, welcher durch die Kernfunktionen $k(\cdot, x)$ auf den Datenpunkten $x \in \mathcal{X}$ aufgespannt wird.

Definition 5.5. *Das regularisierte Risikofunktional $J : \mathcal{H} \to \mathbb{R}$ ist die Summe aus einer Kostenfunktion c über den Trainingswerten und den von der Funktion berechneten Werten, sowie einer monotonen Funktion g, welche von der Komplexität der Funktion abhängt.*

$$J : \mathcal{H} \to \mathbb{R} \tag{5.25}$$
$$J(f) := c((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))) + g(\|f\|^2_{\mathcal{H}}) \tag{5.26}$$

Mit $c : (\mathcal{X} \times \mathcal{Y}^2)^n \to \mathbb{R}$ als Kostenfunktion des Fehlers und $g : \mathbb{R} \to \mathbb{R}$ als Funktion, welche die Stärke der Regularisierung steuert.

Definition 5.6. *Die quadratische Kostenfunktion ist definiert als*

$$c : \sum_{i=1}^{n} \frac{1}{n} \left(y_i - f(x_i)\right)^2 . \tag{5.27}$$

Satz 5.3. *Sei $\mathcal{F} = \{f \in \mathcal{H} | \ \|f\|^2_{\mathcal{H}} < \infty\}$, $g : \mathbb{R}^+ \to \mathbb{R}^+$ eine monotone Funktion, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ eine reelle Kernfunktion, $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times Y$ eine Menge an Trainingsdaten und c die quadratische Kostenfunktion.*

Dann lässt sich eine Funktion $h : \mathcal{X} \to \mathcal{Y}$, welche das regularisierte Risiko-Funktion minimiert, als Linearkombination der Werte des Kerns auf den Trainingsdaten $f(\cdot) := \sum_{i=1}^{n} a_i k(\cdot, x_i)$ darstellen.

Lemma 5.1. *Die Auswertung einer Funktion $h \in \mathcal{F}$ auf einem Trainingspunkt x_i lässt sich als Auswertung von $f + h$ mit $f \in \text{span}(k(\cdot, x_1), \ldots, k(\cdot, x_n))$ und $v \in \mathcal{F}$ schreiben, wobei f und v orthogonal sind.*

Beweis:
Eine Funktion $h \in \mathcal{F}$ lässt sich zerlegen in einen Teil f, der eine Linearkombination von Auswertungen der Kernfunktion auf den Trainingsdaten ist, und einen Teil v der dazu orthogonal ist.

Es gilt

$$h : \left(\sum_{i=1}^{n} \alpha_i \mathbf{\Phi}(x_i)\right) + v = \left(\sum_{i=1}^{n} \alpha_i k(\cdot, x_i)\right) + v = f + v \tag{5.28}$$

mit $\langle v, \mathbf{\Phi}(x_i)\rangle_{\mathcal{H}} = \langle v, k(\cdot, x_i)\rangle_{\mathcal{H}} = 0 \quad \forall x_i \in \{x_1, \ldots, x_n\}$.

Für die Auswertung der Funktion f auf einem Trainingspunkt x_j im RKHS $\mathcal{H}$ bekommt man damit:

$$h(x_j) = \langle h, k(\cdot, x_j)\rangle_{\mathcal{H}} = \langle f + v, k(\cdot, x_j)\rangle_{\mathcal{H}} \tag{5.29}$$
$$= \langle f, k(\cdot, x_j)\rangle + \underbrace{\langle v, k(\cdot, x_j)\rangle_{\mathcal{H}}}_{=0} \tag{5.30}$$
$$= \langle f, k(\cdot, x_j)\rangle_{\mathcal{H}} = f(x_j) \tag{5.31}$$

□

Lemma 5.2. *Der Wert der Funktion g ist für $\|f\|_{\mathcal{H}}$ kleiner oder gleich dem Wert für die Norm von $\|h\|_{\mathcal{H}}$ selber: $g(\|f\|_{\mathcal{H}}) \leq g(\|f+v\|_{\mathcal{H}}) = g(\|h\|_{\mathcal{H}})$.*

Beweis:
Für $\|h\|_{\mathcal{H}}^2 = \|f+v\|_{\mathcal{H}}^2$ gilt:

$$\|f+v\|_{\mathcal{H}}^2 = \langle f+v, f+v\rangle = \langle f, f+v\rangle + \langle v, f+v\rangle \tag{5.32}$$

$$= \langle f, f\rangle + \underbrace{\langle f, v\rangle}_{=0} + \underbrace{\langle v, f\rangle}_{=0} + \langle v, v\rangle \tag{5.33}$$

$$= \|f\|_{\mathcal{H}}^2 + \|v\|_{\mathcal{H}}^2 \tag{5.34}$$

$$g(\|f+v\|_{\mathcal{H}}) = g(\sqrt{\|f\|_{\mathcal{H}}^2 + \|v\|_{\mathcal{H}}^2}) \tag{5.35}$$

Da die Funktion g monoton steigend ist folgt, dass $g(\|f+v\|_{\mathcal{H}}) \geq g(\|f\|_{\mathcal{H}})$ ist für alle v und daher durch $v \equiv 0$ optimiert wird.

□

Mit diesem Lemma ist der Beweis abgeschlossen, da die Wahl von v beim Auswerten der Funktion auf einem Trainingspunkt irrelevant ist und ein v, welches nicht konstant 0 ist die Funktionsnorm vergrößert. Damit folgt, dass die Funktion, welche das Funktional J minimiert, im Raum der Linearkombinationen der Kernfunktion auf den Trainingsdaten liegt.

Die Gram-Matrix

Definition 5.7. *Die Matrix $\mathbf{G} \in \mathbb{R}^{n\times n}$, welche die Werte der Kernfunktion auf allen Kombinationen von x_i, x_j enthält, ist die Gram-Matrix von k auf X:*

$$\mathbf{G} := \begin{pmatrix} k(x_1,x_1), & \dots, & k(x_1,x_n) \\ \vdots & \ddots & \vdots \\ k(x_n,x_1), & \dots, & k(x_n,x_n) \end{pmatrix}$$

Lernen einer Funktion

Nachdem wir gezeigt haben, dass es im Raum $\mathcal{H}$ eine endliche Funktion gibt, welche die Funktionswerte auf den Trainingspunkten berechnet, möchten wir diese nun finden.

Dazu seien Messpunkte $(x_i, y_i) \in X \times \mathcal{Y}, i = 1 \dots n$ mit $n = |X| = |Y|$ gegeben, welche gegenüber der Funktion aus der sie stammen einen Fehler aufweisen. Um unser Ergebnis zu überprüfen verwenden wir eine Menge an Testpunkten, bei welchen wir die exakte Lösung ohne Fehler kennen.

Wir wissen aus Lemma 5.1, dass wir eine Funktion, welche für alle $i = 1, \dots, n$ die

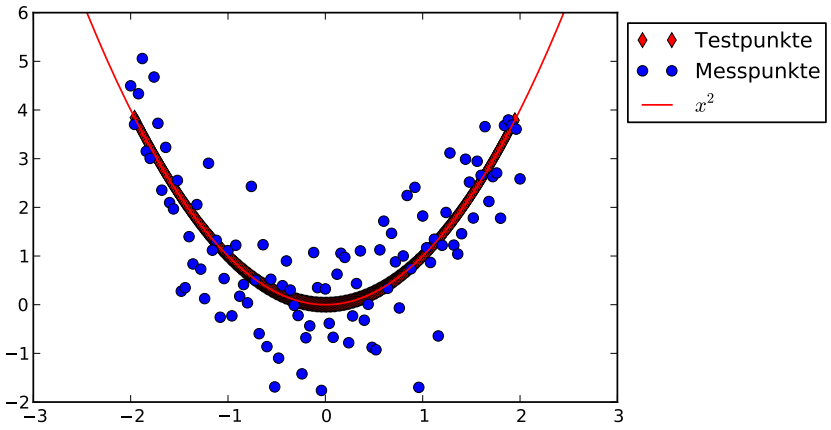


Abb. 5.5: Beispiel: ungenaue Messpunkte $(x_i, y_i) \in \mathbb{R}^2$ sind gegeben, und es soll die Funktion der Punkte rekonstruiert werden.

Punkte x_i auf y_i abbildet als Linearkombination

$$f(\cdot) := \sum_{i=1}^{n} a_i k(\cdot, x_i) \tag{5.36}$$

schreiben können, wobei die $k(\cdot, x_i)$ die Kernfunktionen auf den Punkten x_i sind.

Um nun eine Funktion zu rekonstruieren, welche die Trainingspunkte approximiert müssen die Koeffizienten a_i passend bestimmt werden.

Dazu verwenden wir das Funktional J aus Formel 5.26 mit einer Kostenfunktion $c : f \mapsto \sum_{i=1}^{n}(f(x_i) - y_i)^2$ und $g \equiv \text{const}$.

Für den Koeffizientenvektor $\mathbf{a}$, welcher eine Funktion f ergibt, die das Funktional minimiert gilt mit $\mathbf{a} = (a_1, \ldots, a_n)^T$, der Gram-Matrix $\mathbf{G}$ und $\mathbf{y} = (y_1, \ldots, y_n)^T$ als Vektor der Trainingswerte

$$\mathbf{Ga} = (f(x_1), \ldots, f(x_n))^T \stackrel{!}{=} \mathbf{y}. \tag{5.37}$$

Mit einem Lösungsvektor $\mathbf{a}$, welcher die Gleichung erfüllt, rekonstruiert die Funktion $f(\cdot)$, wie sie in Formel 5.36 definiert ist, auf den Trainingspunkten x_i gerade die Trainingswerte y_i, da es nach dem Representer-Theorem für die Gleichung eine exakte Lösung gibt.

Der Fehler auf einer Menge von Testpunkten mit bekanntem Funktionswert ist $\|\mathbf{f} - \mathbf{y}\|_2^2$. Wenn die Menge der Testpunkte mit der Menge der Trainingspunkte übereinstimmt, dann ist der Fehler gerade 0.

Während die Funktion auf den Trainingsdaten mit dieser Lösung keinen Fehler aufweist, liefert sie allerdings im Allgemeinen keine gute Generalisierung auf Punkten $x \in \mathcal{X} \setminus X$, die nicht zum Lernen verwendet wurden. Da es immer eine Funktion gibt, welche die

Trainingspunkte korrekt rekonstruiert, misst man den Fehler der Lösung auf Mengen von Testpunkten, welche mit der Menge der Trainingspunkte disjunkt sind.

Um eine bessere Generalisierung zu erhalten, welche nach Möglichkeit auch auf den ungelernten Punkten einen kleinen Fehler hat, verwendet man den von $\|f\|_{\mathcal{H}}$ abhängigen Term um die Komplexität der Funktion zu beeinflussen. Damit lässt sich verhindern, dass die Funktion die Messdaten nur "auswendig lernt".

Regularisierung

Der Effekt, dass die Trainingsdaten sehr genau getroffen werden, aber ungelernte Daten einen großen Fehler aufweisen, heißt Overfitting. Ein Ansatz um dem Problem zu begegnen ist die Funktion möglichst einfach zu halten in der Annahme, dass die Funktion aus der die Daten stammen glatter ist, als die Funktion, welche die fehlerbehafteten Testdaten exakt trifft. Sollte die Funktion diese Glattheitsannahme nicht erfüllen, ist es im Allgemeinem schwierig sie mit Hilfe der Trainingsdaten zu lernen.

Wir verwenden daher den Term $g(\|f\|_{\mathcal{H}})$ um die Komplexität der Funktionen gering zu halten, wobei g eine monotone Funktion $\mathbb{R} \to \mathbb{R}$ ist, durch welche die Stärke der Regularisierung gesteuert wird.

Wenn die Funktionsnorm stärker gewichtet wird vergrößert sich zwar der Fehler auf den Trainingsdaten (x_i, y_i), aber häufig werden die nicht gelernten Daten durch eine weniger komplexe Funktion besser approximiert.

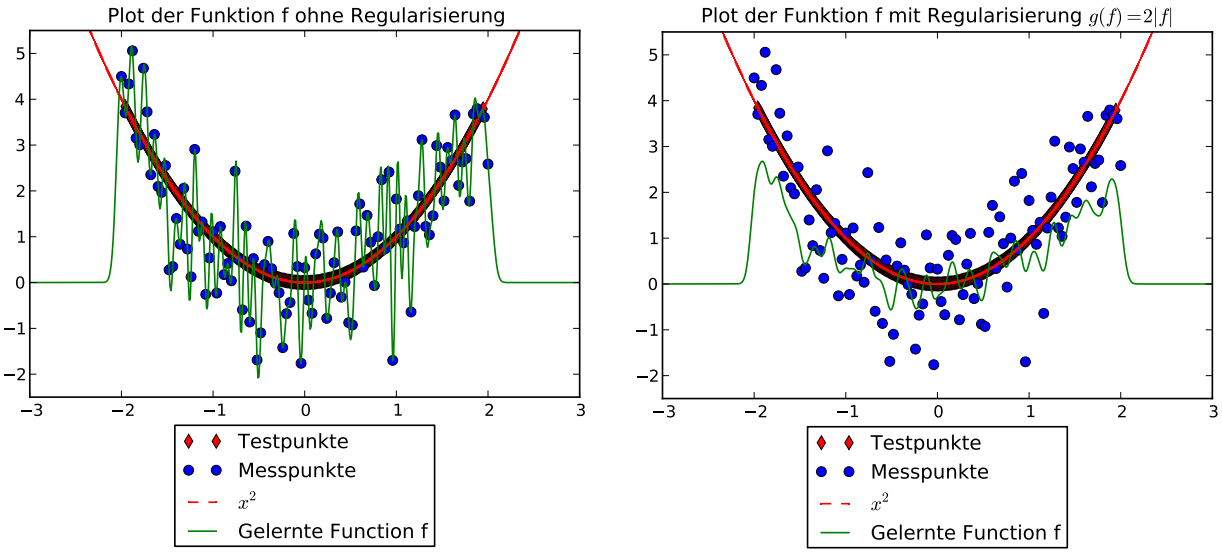


Abb. 5.6: Eine Funktion, die mit Hilfe eines Gaußkerns $k : x, y \mapsto \exp(-\frac{\|x-y\|_2^2}{2(0.05^2)})$ aus den Trainingspunkten mit Regularisierung über die Funktionsnorm gelernt wurde.

In Abbildung 5.6 ist die Rekonstruktion einer Funktion aus den Messdaten in Grafik 5.5 gezeigt, einmal ohne Regularisierung (d.h. $g(\cdot) = 0$) und einmal mit einer Funktion

$g(\|\cdot\|_2^2) \mapsto \gamma\|\cdot\|_{\mathcal{H}}^2$ als Regularisierungsterm.

Dabei ist zu beachten, dass die Messpunkte auch in Bezug auf die Funktion der Testpunkte schon einen Fehler aufweisen, welcher durch die rekonstruierte Funktion gerade ausgeglichen werden soll.

Man sieht, dass die regulierte Funktion glatter ist, und auf den exakten Testpunkten einen kleineren Fehler hat, während der Fehler auf den ungenauen Messpunkten durch die Regulierung größer wird. Es ist daher auch zu erwarten, dass die Interpolation auf weiteren Punkten ebenfalls ein besseres Ergebnis liefert, als die Funktion ohne Regularisierung.

Wenn allerdings die Gewichtung der Regularisierung gegen unendlich geht, dann geht die Funktionsapproximation gegen $f(\cdot) = 0$, da $a_i = 0\ \forall i$ die Norm $\|f\|_{\mathcal{H}}$ minimiert und der Fehlerterm sich mit steigender Gewichtung der Funktionsnorm immer weniger auf das Ergebnis auswirkt. Ein Beispiel für ein zu großes Gewicht ist in Grafik 5.7 zu sehen.

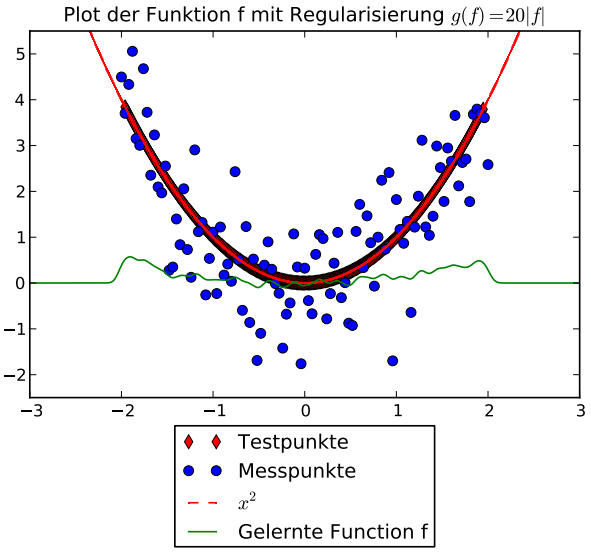


Abb. 5.7: Das Ergebnis zu starker Regularisierung ist, dass die Funktion sich $f(\cdot) = 0$ annähert. Daher führt zu starke Regularisierung ebenfalls zu einem schlechtem Ergebnis.

Mit der Funktionsnorm $\|f\|_{\mathcal{H}} = \langle f, f\rangle$ von f aus Formel 5.21, dem Koeffizientenvektor $\mathbf{a}$, dem Vektor der Funktionswerte auf den Trainingspunkten $\mathbf{f} = (f(\mathbf{x_1}), \ldots, f(\mathbf{x_n}))$, der Gram-Matrix $\mathbf{G}$ und der monotonen Funktion $g : x \mapsto \gamma x$ mit $\gamma \geq 0$ lässt sich das Optimierungsproblem wie folgt beschreiben.

$$
\begin{aligned}
J(f) &:= \sum_{i=1}^{n} \|f(x_i) - y_i\|_2^2 + \gamma\|f\|_{\mathcal{H}} && (5.38)\\
&= \|\mathbf{Ga} - \mathbf{y}\|_2^2 + \gamma(\mathbf{a^T Ga}) && (5.39)\\
&= \mathbf{a^T G^T Ga} - 2\mathbf{a^T G^T y} + \mathbf{y^T y} + \gamma(\mathbf{a^T Ga}) && (5.40)\\
&= \mathbf{a^T}(\mathbf{G^T} + \gamma\mathbb{I})\mathbf{Ga} - 2\mathbf{a^T G^T y} + \mathbf{y^T y} && (5.41)
\end{aligned}
$$

Die notwendige Bedingung für ein $\mathbf{a}$, welches J minimiert ist:

$$\nabla_{\mathbf{a}} J(\mathbf{a}) = 2(\mathbf{G^T} + \gamma\mathbb{I})\mathbf{G}\mathbf{a} - 2\mathbf{G^T}\mathbf{y} = \mathbf{0}$$

Dass es sich bei der Extremstelle um ein Minimum handelt kann leicht festgestellt werden, da $\nabla^2_{\mathbf{a}} J(\mathbf{a}) = 2(\mathbf{G^T} + \gamma\mathbb{I})$ gilt und $\mathbf{G}$ nach Definition positiv semi-definit ist.

Damit muss

$$2\mathbf{G^T}\mathbf{G}\mathbf{a} + 2\gamma\mathbb{I}\mathbf{G}\mathbf{a} = 2\mathbf{G^T}\mathbf{y} \tag{5.42}$$

gelöst werden. Der Fehler der Lösung auf den Trainingspunken enthält den konstanten Term und ist gerade die Fehlernorm

$$\|\mathbf{G}\mathbf{a} - \mathbf{y}\|_2^2 = \mathbf{a^T}\mathbf{G^T}\mathbf{G}\mathbf{a} - 2\mathbf{y^T}\mathbf{G}\mathbf{a} + \mathbf{y^T}\mathbf{y} \geq 0, \tag{5.43}$$

welche der erste Term des Funktionals ist.

5.4 Unsupervised Learning

Unsupervised learning Verfahren erkennen Muster in Daten, zu denen nur die Datenpunkte, aber keine Trainingswerte gegeben sind.

Eine typische Anwendung ist zum Beispiel das Clustering von Punkten in eine bestimmte Anzahl Cluster, innerhalb derer sich die Punkte wenig unterscheiden, während die Unterschiede zwischen Punkten unterschiedlicher Cluster groß sind.

Nachdem ein solches Verfahren ein „Muster“ in den Daten gelernt hat, können zum Beispiel Stichproben aus verschiedenen Clustern verwendet werden, um die gemeinsame Eigenschaft einer Gruppe zu bestimmen.

So kann zum Beispiel ein unsupervised Verfahren aus einer Menge von handschriftlichen Ziffern[2] Gruppierungen finden, welche jeweils einer Ziffer entsprechen. Die Zuordnung, welcher Cluster welcher Ziffer entspricht kann dann am Ergebnis vorgenommen werden.

Beispiele für unsupervised learning Algorithmen sind unter anderem einfache Cluster-Algorithmen wie k-Means, welcher k Schwerpunkte einer Punktemenge findet und self-organizing maps [Koh90], welche die Datenpunkte als Stimulus verwenden um die Punkte eines Gitters zu verschieben, wobei sich die im Gitter benachbarten Punkte gegenseitig beeinflussen.

Unüberwachtes Lernen mit dem Laplaceoperator

Der Laplaceoperator misst, wie stark sich der Mittelwert einer Funktion in der Umgebung eines Punktes mit wachsendem Radius der Umgebung verändert.

[2] Die MNIST-Datenbank [LC98] ist ein häufig verwendeter Test für Klassifikationsalgorithmen

Bei diskreten Punkten heißt das, dass der Graph-Laplace eine Aussage über den Einfluss eines Punktes auf seine Nachbarpunkte auf dem Graphen ermöglicht.

Diese Eigenschaft ist für Clustering interessant, da sich innerhalb eines Clusters der Wert der Punkte nur wenig ändern soll, während sich der Wert für Punkte aus unterschiedlichen Clustern deutlich unterscheiden darf. Damit lässt sich mit dem Laplaceoperator eine Aussage über die Qualität eines Clustering treffen und er kann sogar genutzt werden, um ein eigenes Clustering zu definieren.

Einfache Beispiele

In Abschnitt 2.6 haben wir bereits gezeigt, dass sich aus den Eigenvektoren des Graph-Laplace zum Eigenwert 0 die Zusammenhangskomponenten des Graphen $G = (V, E)$ ablesen lassen. Damit ist bereits erkennbar, welche Punkte keine Ähnlichkeit (bezüglich des verwendeten Graphen) zueinander aufweisen.

Dieser triviale Ansatz zeigt aber nur vorhandene ZHK auf und ermöglicht noch kein Clustering in k Cluster mit gegebenem $k \in \mathbb{N}, k \leq |V|$.

Graph-Cuts, wie der Cheeger-Cut, welchen wir in Abschnitt 2.6 beschrieben haben, sind ebenfalls eine Möglichkeit um Knoten in unterschiedliche Klassen einzuteilen, welche einem Balance-Kriterium entsprechen und das Gewicht der Kanten zwischen zwei Clustern klein halten.

Spektrales Clustering

Eine Methode des unsupervised learning mit Hilfe des (Graph-)Laplace ist das *spektrale Clustering*, welches in Algorithmus 1 (aus [VL07]) definiert ist.

Dabei werden die i-ten Einträge der ersten k Eigenvektoren zu Vektoren y_i zusammengefasst, welche dann mit Hilfe eines einfachen k-Means Algorithmus in k Cluster von Eigenvektoren unterteilt werden können. Die Cluster der Eingabepunkte enthalten dann

die Punkte, welche zu den jeweiligen Zeilen in den Vektoren y_i korrespondieren.

Algorithmus 1: Algorithmus: Spektrales Clustering mit dem unnormalisiertem Graph-Laplace

Input : Graph $G = (V, E), V = \{v_1, \ldots, v_n\}$, Graph-Laplace $\mathbf{L}$, Anzahl $k \leq n$ der Cluster.

- Berechne die ersten k Eigenvektoren $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_k$ des Graph-Laplace $\mathbf{L}$.
- Stelle die Matrix $\mathbf{U}$ auf, welche als Spaltenvektoren die k Eigenvektoren $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_k$ enthält.
- Berechne mit k-Means die Cluster $C_1, \ldots, C_k$ der Zeilenvektoren $\mathbf{y}_1, \ldots, \mathbf{y}_n$ der Matrix $\mathbf{U}$.
- Die Cluster A_i sind die Mengen der Punkte zu den Zeilenvektoren in den Clustern C_i: $A_i := \{v_j | y_j \in C_i\}$ $\forall i = 1, \ldots, k$ mit $j \in \{1, \ldots, n\}$. Es gilt $A_i \cap A_j = \emptyset$ $\forall i, j$ und $v_j \in \bigcup_i A_i$ $\forall j = 1 \ \ldots, k$

Output : Cluster $A_1, \ldots, A_k$

5.5 Semi-Supervised Learning

Einführung

Das *semi-supervised learning* (halbüberwachtes Lernen) kombiniert den Ansatz des überwachten Lernens aus bekannten Daten mit dem Ansatz des unüberwachten Lernens, bei dem nur Datenpunkte, aber keine Werte dazu bekannt sind.

Nachdem wir in Abschnitt 5.3 eine Methode zum überwachtem Lernen mit Trainingsdaten vorgestellt haben und in Abschnitt 5.4 gesehen haben, dass der Laplaceoperator ein Ähnlichkeitsmaß zum unüberwachtem Lernen ist, werden nun beim semi-supervised learning die beiden Ansätze kombiniert um bessere Ergebnisse zu erzielen. Dazu wird die Methode, mit einem Kern eine Funktion aus Beispieldaten zu erlernen, verbessert, indem Datenpunkte mit unbekanntem Wert hinzugenommen werden.

Verwendet man den Laplaceoperator als Regularisierung beim überwachtem Lernen, wird die Glattheit der gelernten Funktion mit einbezogen und damit auf Funktionen hin optimiert, welche auf Punkten, die auf der Mannigfaltigkeit benachbart sind, ähnliche Werte liefern.

Bei diesem Ansatz folgen wir u.a. Belkin, Niyogi [BN04], [BNS06] und Minh, Sindhwani, [MS11].

Motivation

Als Motivation sei ein einfaches Klassifikationsproblem gegeben. Dazu sollen zweidimensionale Punkte in einem 100×100 Quadrat in zwei Klassen eingeteilt werden. In Abbil-

Tabelle 5.1: Farben in den Eingabegrafiken

hellrot	Trainingspunkt aus der ersten Klasse
hellgrün	Trainingspunkt aus der zweiten Klasse
dunkelrot	Testpunkt aus der ersten Klasse
dunkelgrün	Testpunkt aus der zweiten Klasse
schwarz	Punkt auf der Mannigfaltigkeit ohne Trainingswert
weiß	Unbekannter Punkt

Tabelle 5.2: Farben in den Ergebnis-Grafiken

hell	Klassifikation eines unbekannten Punktes
mittel	Klassifikation eines gegebenen Punktes ohne bekannten Trainingswert
dunkel	Klassifikation eines Trainingspunktes
sehr dunkel	Klassifikation eines Testpunktes

dung 5.8 sind die Eingabedaten für ein Problem mit wenig bekannten Trainingspunkten, mit vielen bekannten Trainingspunkten und mit wenig bekannten Trainingspunkten, aber zusätzlichen Punkten ohne bekannten Trainingswert zu sehen.

Die Farben in der Eingaben in Abbildung 5.8 sind in Tabelle 5.1 definiert und Tabelle 5.2 beschreibt die Farben in den Ergebnisgrafiken.

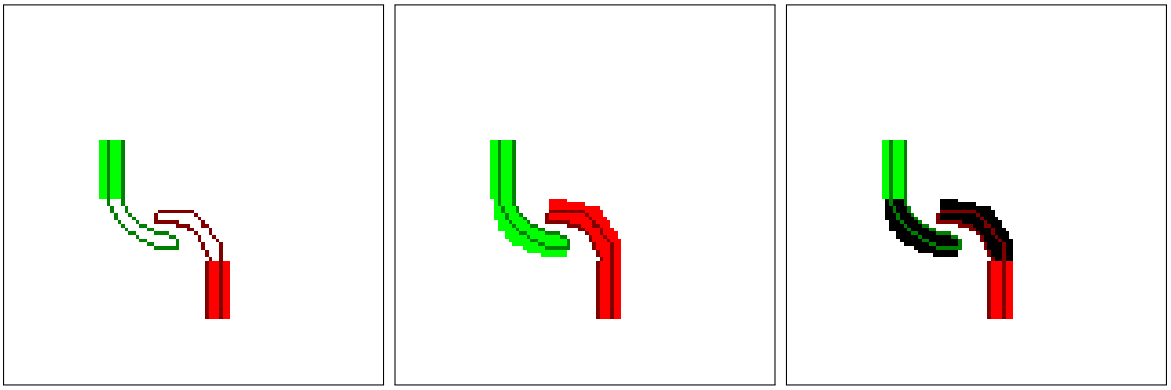


Abb. 5.8: Links: Eine Eingabe mit $l = 160$ Lernpunkten (hell) und $t = 153$ Testpunkten (dunkel) aus $\{-1, +1\}$. Mitte: Eine Eingabe mit $l = 422$ bekannten Punkten und $t = 153$ Testpunkten. Rechts: Eine Eingabe bei der die Mannigfaltigkeit mit $l+u = 422$ Punkten bekannt ist, aber nur bei $l = 160$ Punkten ein Trainingswert bekannt ist, welche ebenfalls $t = 153$ Testpunkte enthält.

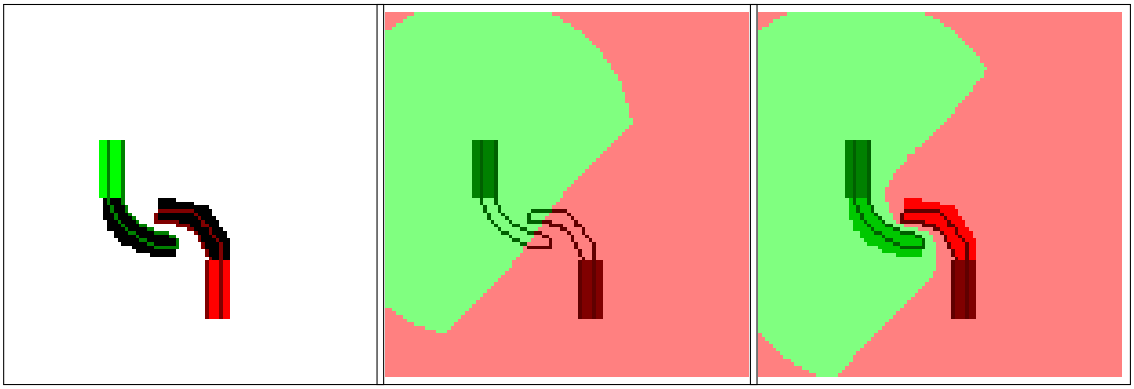
Problem des supervised learning Verfahrens

Wenn man in diesem Beispiel ein supervised Verfahren zusammen mit einer Regularisierung über die Funktionsnorm verwendet, ist die typische Lösung eine Funktion, welche alle

Trainingsdaten trifft und dabei wenig Komplexität hat, dafür aber schlecht generalisiert. Dadurch liefert sie auf den Testpunkten einen verhältnismäßig großen Fehler.

Eine solche Lösung mit supervised learning ist in Grafik 5.9 in der Mitte zu sehen. Da die Punkte ohne Trainingswert nicht genutzt werden, ergibt die Lösung eine Trennung, welche nur die Trainingsdaten berücksichtigt, diese aber exakt trifft.

Semi-Supervised Learning mit Kernen



Eingabe Supervised Semi-Supervised

Abb. 5.9: Ergebnisse bei der Klassifikation mit supervised und semi-supervised Kern-Verfahren mit Gaußkern

Um den Laplaceoperator in die Methode des Lernens mit Kernen mit einzubeziehen und so eine Regularisierung zu bekommen, welche besser an die Daten angepasst ist, verwendet man eine Erweiterung des Kostenfunktionals aus Formel 5.26. Dazu ergänzt man einen Term, welcher das Funktional mit Hilfe des Laplaceoperators regularisiert.

Die Laplace-Seminorm

Die Laplace-Seminorm $\|\cdot\|_L^2$ für Funktionen aus $\mathcal{H}$ ist definiert als

$$\|f\|_L^2 = \int_\Omega f(x)(\triangle f)(x)\mathrm{dx} \tag{5.44}$$

$$\text{diskret: } \|f\|_L^2 = \sum_{i,j=0}^{n} f(x_i)L_{ji}f(x_j) = \mathbf{f^T L f} \tag{5.45}$$

Lemma 5.3. *$\|\cdot\|_L$ ist eine Seminorm, das heißt es gilt:*

$$\|\lambda f\|_L = |\lambda| \ \|f\|_L$$
$$\|f_1 + f_2\|_L \leq \|f_1\|_L + \|f_2\|_L$$

Im Gegensatz zu einer Norm ist eine Seminorm nicht zwingend positiv definit. Das

heißt, dass $(\|f\| = 0) \Rightarrow (f \equiv 0)$ nicht zwingend gilt. Ein Beispiel bei dem die Laplace-Seminorm diese Eigenschaft nicht erfüllt, ist eine Funktion, welche einen Sattelpunkt hat.

Das erweiterte Kostenfunktional

Das um die Laplace-Seminorm als Regularisierung erweitere Kostenfunktional ist definiert als:

$$J : \mathcal{H} \to \mathbb{R} \tag{5.46}$$

$$F(f) := c((x_1, y_1, f(x_1)), \ldots, (x_l, y_l, f(x_l))) + g_1(\|f\|_{\mathcal{H}}^2) + g_2(\|f\|_L^2) \tag{5.47}$$

Dabei ist l die Anzahl Punkte mit Trainingswert (Label), u die Anzahl Punkte ohne Trainingswert (unlabeled), $g_1, g_2 : \mathbb{R} \to \mathbb{R}$ sind zwei monotone Funktionen, $\|f\|_{\mathcal{H}}$ ist die Funktionsnorm und $\|f\|_L^2$ die Laplace-Seminorm der Funktion.

Dabei legen wir uns auf $g_1(x) := \gamma_A x$ und $g_2(x) := \gamma_I x$ fest mit den Indizes A für "Ambient" und I für "Intrinsic", wie sie in [MS11] verwendet werden. Die Kostenfunktion c sei $\|f(\mathbf{x}) - \mathbf{y}\|_2^2$, und die Funktionen in $\mathcal{H}$ der Form $f(\cdot) := \sum_i^{(l+u)} a_i k(\cdot, x_i)$, also Linearkombinationen aller gegebenen Punkte. Damit ergibt sich das Funktional:

$$F(f) := \|\mathbf{Ga} - \mathbf{y}\|_2^2 + \gamma_A \|f\|_{\mathcal{H}}^2 + \gamma_I \|f\|_L^2 \tag{5.48}$$

Dabei ist der Fehler auf den Trainingsdaten auf den Punkten $x_1, \ldots, x_l$ definiert, die Funktions- und Laplace-Norm aber auf allen Punkten $x_1, \ldots, x_{l+u}$.

Sei $\mathbf{A} \in \mathbb{R}^{(l+u)\times(l+u)}$ die Gram-Matrix über alle gegebenen Punkte, und $\mathbf{L} \in \mathbb{R}^{(l+u)\times(l+u)}$ ein Graph-Laplace der Punkte.

Definition 5.8. *Eine Ausschneidematrix $\mathbf{J}_{l\times(l+u)}$ ist definiert als:*

$$\mathbf{J}_{l\times(l+u)} := \begin{cases} 1 & \text{für } J_{ii} \quad \text{mit } i \leq l \\ 0 & \text{sonst.} \end{cases} \tag{5.49}$$

Eine Anwendung der Matrix als $\mathbf{JA}$ ergibt eine Matrix aus $\mathbb{R}^{l\times(l+u)}$, welche die ersten l Zeilen der Matrix A enthält. Analog ergibt $\mathbf{AJ^T}$ eine Matrix mit den ersten l Spalten der Matrix $\mathbf{A}$.

Damit gilt:

- $\mathbf{A}_{(l+u)\times(l+u)}$ ist die Gram-Matrix aller gegebenen Punkte.
- $\mathbf{G}_{l\times l} = \mathbf{JAJ^T}$ ist die Gram-Matrix der Trainingspunkte.
- $\mathbf{a} \in \mathbb{R}^{(l+u)}$ ist der Lösungsvektor für das Optimierungsproblem.
- $\mathbf{f}_{l+u} = \mathbf{Aa}$ ist der Vektor der Funktionswerte $f(x_i), i = 1, \ldots, l+u$.
- $\mathbf{f}_l = \mathbf{Jf}_{(l+u)} = \mathbf{Ga} = \mathbf{JAa}$ ist der Vektor der Funktionswerte auf den Punkten mit Trainingswert.

- $\mathbf{y}_l$ ist der Vektor der Trainingswerte.

Damit können wir das Funktional mit Matrizen schreiben als:

$$F(f) := \mathbf{a^T A^T J^T J A a} - 2\mathbf{a^T A^T J^T y} + \mathbf{y^T y} + \gamma_A \mathbf{a^T A a} + \gamma_I \mathbf{a^T A^T L A a} \tag{5.50}$$

$$\nabla_{\mathbf{a}} F(f) := 2\mathbf{A^T J^T J A a} - 2\mathbf{A^T J^T y} + \gamma_A 2\mathbf{A a} + \gamma_I 2\mathbf{A^T L A a} \tag{5.51}$$

$$= 2(\mathbf{A^T J^T J} + \gamma_A \mathbb{I} + \gamma_I \mathbf{A^T L})\mathbf{A a} - 2\mathbf{A^T J^T y} \tag{5.52}$$

Der Fehler auf den Trainingspunkten ist damit:

$$\text{Error}(\mathbf{f}_l, \mathbf{y}_l) = \mathbf{a^T A^T J^T J A a} - 2\mathbf{a^T A^T J^T y} + \mathbf{y^T y} \tag{5.53}$$

Der Laplaceoperator als Regularisierungsterm bewirkt dabei, dass Funktionen bei denen sich der Funktionswert zwischen benachbarten Punkten wenig ändert bevorzugt werden, da dort der Laplace-Term klein bleibt.

Diesen Effekt kann man für das Beispiel in Grafik 5.9 sehen, wo eine Lösung mit $\gamma_A = 10$ für das supervised-Verfahren und $\gamma_I = 10$ für das semi-supervised Verfahren gezeigt ist.

In diesem Beispiel erkennt man, dass die Information aus den Punkten ohne bekannten Funktionswert hilft eine optimale Lösung für das Klassifikationsproblem zu finden, obwohl gerade für die Punkte an der „schwierigen Stelle“, wo sich die zwei Klassen nahe kommen, keine Funktionswerte zum Training vorhanden sind.

5.6 Unterschiede der Laplaceoperatoren

Beim semi-supervised Learning wird zum Beispiel ein einfacher Graph-Laplace wie er in Abschnitt 2.6 beschrieben ist verwendet, da die Daten nur als hochdimensionale Punkte vorliegen. Durch den so erzeugen Laplaceoperator wird eine Dimensionsreduktion erreicht, die es ermöglicht eine Funktion zu finden, welche die hochdimensionalen Datenpunkte in einer wesentlich kleineren Dimension beschreiben kann.

Dies unterscheidet sich zunächst deutlich vom DEC-Laplace, wie wir ihn in Abschnitt 3.8 beschrieben haben, da dieser *nicht* zur Dimensionsreduktion definiert wurde, sondern nur um die Geometrie der Mannigfaltigkeit beim Rechnen zu verwenden, während die Daten der Mannigfaltigkeit normalerweise in der (geringen) Dimension des umgebenden Raums liegen.

Beide Ansätze können sich aber auch überlappen, wenn zum Beispiel wie in Abschnitt 3.13 beschrieben aus hochdimensionalen Datenpunkten ein Simplex-Komplex erzeugt wird und der daraus berechnete DEC-Laplaceoperator verwendet wird.

Wie im Beispiel dort ist es möglich, dass die Daten eine geringere Dimension haben als die Dimension des Raumes in dem die Datenpunkte liegen. Die Überschneidung der beiden Ansätze entsteht dadurch, dass das Berechnen eines DEC-Laplace aus Simplices, die aus den hochdimensionalen Daten generiert wurden, dem Ansatz aus der Punktwolke direkt einen Graph-Laplace zu erzeugen sehr ähnlich ist.

In beiden Fällen wird aus den hochdimensionalen Punkten ein Graph erzeugt, welcher beim DEC-Laplace strengeren Regeln unterliegt, da der Graph zusammen mit einer Metrik zwischen den Punkten einen gültigen Simplex-Komplex ergeben muss.

Wenn man vermutet, dass die Punkte eine einfache Geometrie beschreiben, kann ein Simplex-Komplex die Geometrie vermutlich besser erfassen als ein allgemeiner Graph.

Weitere diskrete Laplaceoperatoren werden in [WMKG07] verglichen. Dort wird auch aufgezeigt warum es keine allgemeingültige beste Wahl für einen diskreten Laplaceoperator gibt.

6 Ergebnisse beim maschinellem Lernen

6.1 Lernen von Funktionen

Die Beispiele aus dem Abschnitt zur Funktionsrekonstruktion wurden mit einem für diese Arbeit entwickeltem Python-Programm gerechnet. Nachdem wir in Kapitel 5 im Abschnitt zum supervised learning bereits die Ergebnisse gesehen haben, sind die resultierenden Funktionen und die Fehler auf den Trainings- und Testdaten in Grafik 6.1 aufgetragen.

Man sieht, dass ohne Regularisierung der Fehler auf den Trainingsdaten exakt 0 ist, während der Fehler auf den Testdaten groß ist. Die Fehler auf den einzelnen Datenpunkten sind gleichmäßig verteilt, da der Fehler auf den Trainingspunkten aus einer Normalverteilung erzeugt wurde.

Mit Regularisierung steigt der Fehler auf den Trainingsdaten, bleibt aber noch unter dem Fehler auf den Testdaten, welcher ohne Regularisierung auftritt. Der Fehler auf den Testdaten hingegen sinkt, da die erzeugte Funktion glatter ist und dadurch die korrekte Funktion besser approximiert indem der Fehler in den Trainingsdaten durch eine Mittlung kompensiert wird.

Ist die Funktion überregularisiert, geht der Fehler gerade gegen den Absolutwert der Funktionsdaten, da die Funktion sich $f \equiv 0$ annähert.

6.2 Klassifikation von 2D-Daten

Die Beispiele aus Grafik 5.8 im Abschnitt 5.5 sind ebenfalls mit dem selbst entwickeltem Programm zum maschinellen Lernen berechnet worden, wobei dieses beliebig erzeugte Bilder im PNG-Format als Eingabe akzeptiert. Die Farben für die Eingabegrafiken sind in Tabelle 5.1 definiert, die für die Ausgabe in 5.2.

Der Fehler der Klassifikation wird nur auf den Testpunkten gemessen, da zum Beispiel ein supervised Learning ohne Regularisierung die Trainingspunkte immer exakt trifft. Für die Mannigfaltigkeitspunkte ohne Label und alle unbekannten Punkte ist kein Wert bekannt, daher kann der Fehler dort auch nicht bestimmt werden.

Es wäre möglich, dass (einige) Mannigfaltigkeitspunkte ebenfalls Testpunkte sind, hier sind die verschiedenen Punktemengen jedoch paarweise disjunkt.

In Grafik 6.2 ist die Rechnung mit dem supervised-Verfahren zu sehen, bei der nur das Ergebnis, welches für alle Punkte auf der Mannigfaltigkeit eine Trainingslösung hat, es schafft eine Funktion zu lernen, die keinen Fehler auf den Testpunkten hat. In der rechten Abbildung sieht man, dass die Mannigfaltigkeitspunkte ohne Trainingswert in die Lösung nicht mit einfließen.

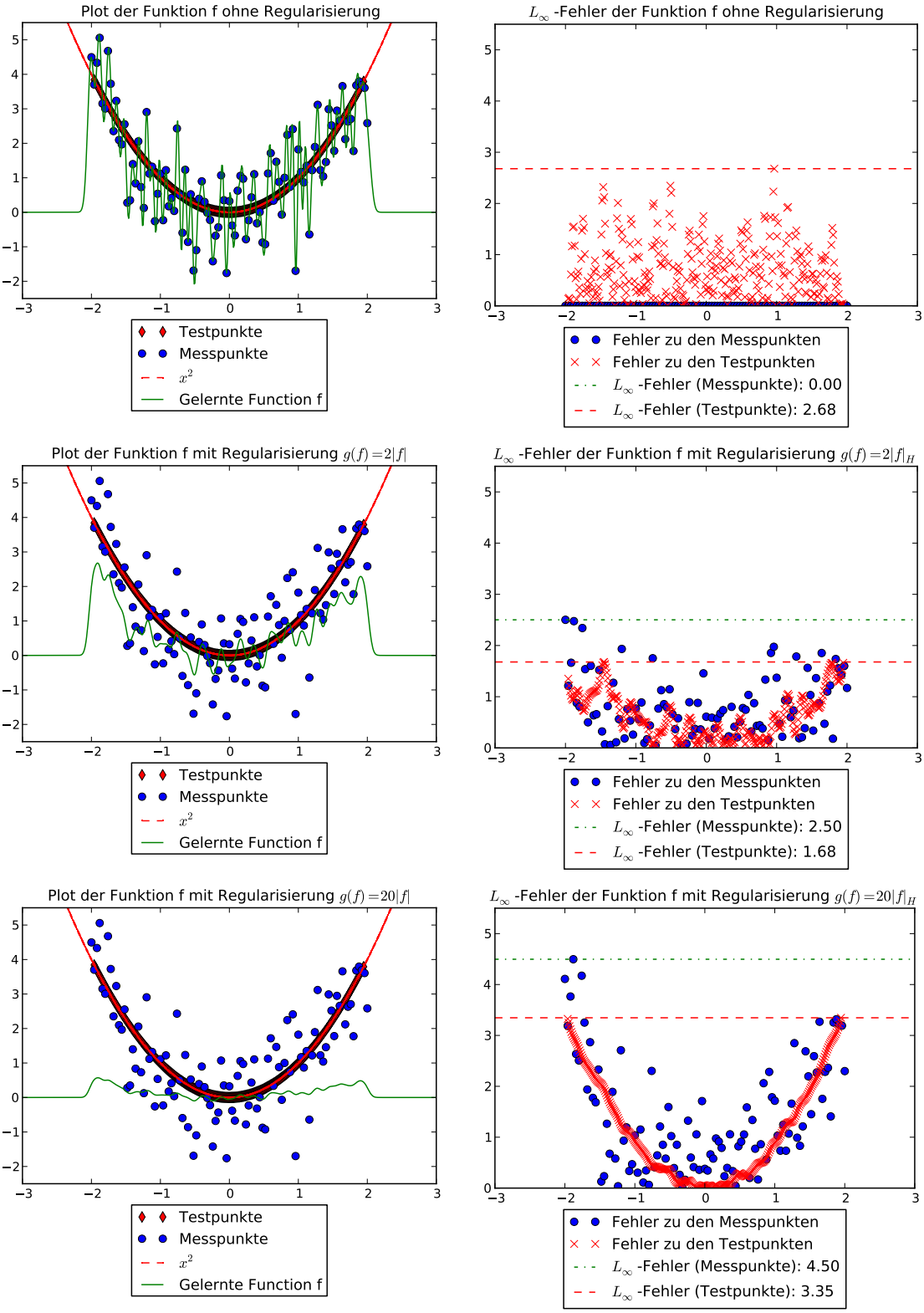
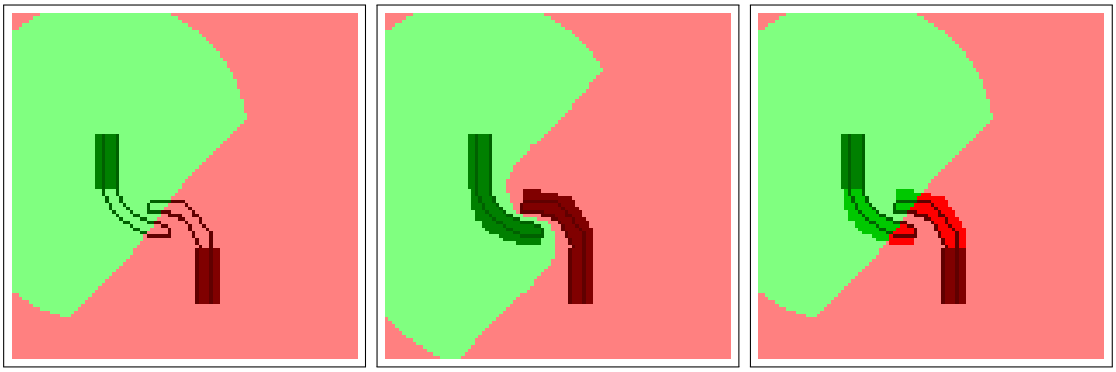


Abb. 6.1: Eine Funktion, die mit Hilfe eines Gaußkerns $k : x, y \mapsto \exp(-\frac{\|x-y\|_2^2}{2(0.05^2)})$ aus den Trainingspunkten mit Regularisierung über die Funktionsnorm gelernt wurde



$\gamma = 10$	$\gamma = 10$	$\gamma = 10$
$\|\nabla_{\mathbf{a}} J\|_2 = 0.00049$	$\|\nabla_{\mathbf{a}} J\|_2 = 0.56362$	$\|\nabla_{\mathbf{a}} J\|_2 = 0.00336$
$\|\mathbf{e}\|_1 = 0.16340$	$\|\mathbf{e}\|_1 = 0.0$	$\|\mathbf{e}\|_1 = 0.16340$

Abb. 6.2: Eine supervised Lösung des Klassifikationsproblems für die Beispiele aus Grafik 5.8 berechnet mit einem Gaußkern mit $\sigma = 1$ und dem Fehlervektor $\mathbf{e} = \frac{1}{t} \sum_i^t |(\text{sgn}(f_i) - \text{sgn}(y_i))|$

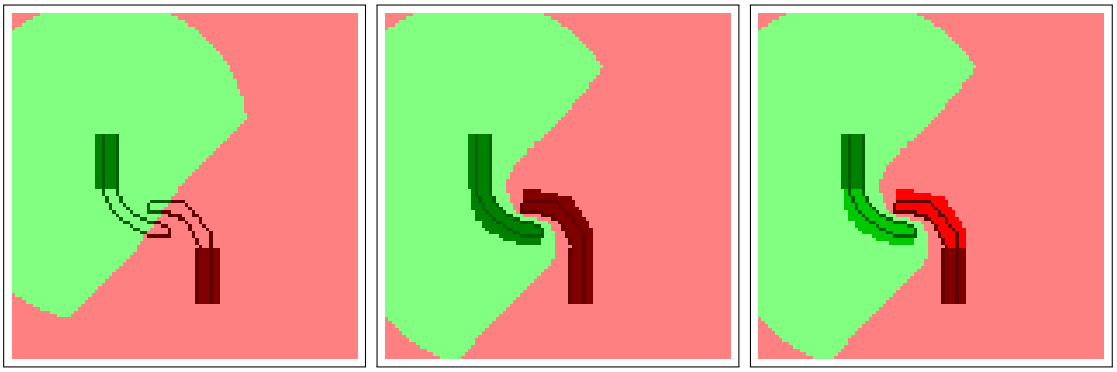
In Grafik 6.3 ist das gleiche Ergebnis mit einem semi-supervised Verfahren berechnet worden. Die Lösungen für den ersten und zweiten Eingaben bleiben gleich, da dort keine zusätzlichen Mannigfaltigkeitspunkte gegeben sind, aber das Ergebnis auf der dritten Eingabe erzielt dank der Laplace-Regularisierung ein Ergebnis, welches mit deutlich weniger Kernvektoren die Klassen korrekt trennt.

6.3 Klassifikation von Ziffern aus dem MNIST-Datensatz

Eine Anwendung, bei der Klassifikation eine große Rolle spielt, ist die Handschrifterkennung. Diese wird zum Beispiel zur Erkennung von Postleitzahlen auf Briefen und in OCR-Programmen verwendet. Da jede Handschrift unterschiedlich ist, müssen Ziffern mit einer Toleranz erkannt werden, die es ermöglicht sie richtig zuzuordnen, auch wenn sie undeutlich geschrieben sind.

Da dieses Beispiel praxisrelevant ist und von Klassifikationsalgorithmen als einfacher Test verwendet werden kann, ist der MNIST-Datensatz von handschriftlichen Ziffern([LC98]) ein häufig verwendetes Beispiel. Dieser Datensatz enthält die Ziffern als 28×28 Pixel Graustufen-Bilder, welche beim semi-supervised Lernen mit Kernen jeweils einen 784-dimensionalen Datenpunkt ergeben.

Mit dem entwickeltem Programm wurden die Ziffern 1 und 0 aus den Trainingsdaten des Datensatzes klassifiziert und mit Hilfe einer Menge von ungelernten Testziffern der Fehler der Klassifikation gemessen. Außerdem ist die Möglichkeit implementiert eine Ziffer, die in einem Grafikprogramm als 28×28 Schwarzweiß-Bild gezeichnet wurde, von dem Programm klassifizieren zu lassen.



$\gamma_I = 10, \gamma_A = 0$	$\gamma_I = 10, \gamma_A = 0$	$\gamma_I = 10, \gamma_A = 0$
$\|\nabla_{\mathbf{a}} J\|_2 = 0.46098$	$\|\nabla_{\mathbf{a}} J\|_2 = 0.56362$	$\|\nabla_{\mathbf{a}} J\|_2 = 1.18532$
$\|\mathbf{e}\|_1 = 0.16334$	$\|\mathbf{e}\|_1 = 0$	$\|\mathbf{e}\|_1 = 0$

Abb. 6.3: Eine Lösung des Klassifikationsproblems mit einem Gaußkern mit $\sigma = 1$, Laplace-Regularisierung $\gamma_I = 10$ und dem Fehlervektor $\mathbf{e} = \frac{1}{t} \sum_i^t |(\text{sgn}(f_i) - \text{sgn}(y_i))|$

Beispiele für die Ziffern, die beim Lernen verwendet wurden, sind in Grafik 6.4 zu sehen.

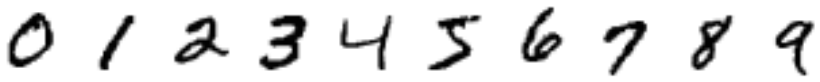


Abb. 6.4: Einige Beispiele für die Ziffern aus dem MNIST Datensatz.

Die Ergebnisse des Tests sind in Abschnitt 6.2 gezeigt.

6.4 Erkennung handschriftlicher Ziffern

Aus dem MNIST Training-Datensatz wurden 422 Ziffern zum Lernen verwendet und 153 Ziffern zum Testen der Ergebnisse zurückgehalten. Mit Hilfe des Algorithmus zur Klassifikation wurden die Ziffern 1 und 0 aus dem MNIST-Datensatz klassifiziert.

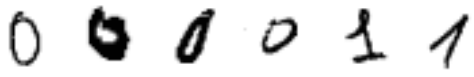


Abb. 6.5: Die 6 falsch klassifizierten Ziffern

Das Ergebnis ist, dass von 1278 Ziffern nur 6 Ziffern inkorrekt erkannt wurden, und diese auch nur weiter als der Schwellenwert ϵ von der korrekten Klasse entfernt waren und nicht der falschen Klasse zugeordnet wurden. Die nicht exakt erkannten Ziffern sind in Abbildung 6.5 zu sehen.

Tabelle 6.1: Parameter beim Lernen der Ziffern aus dem MNIST-Datensatz

Kern	Gaußkern
Graph	Generierter r-Graph mit Kantengewicht 1 für Punkte mit Distanz kleiner oder gleich 1.5 und Kantengewicht 0 sonst.
Eingabe	Die Ziffern sind 28×28 Pixel in Graustufen, d.h. jede Ziffer ist ein 784-dimensionaler Punkt.
Trainingsdaten	1278 Punkte, 635 mit Label, davon 297 mit Label „0“ und 338 mit Label „1“.
γ_A	0.0001
γ_I	0.0001
σ	50
ϵ	0.3 (Abweichung um mehr als ϵ von 0/1 gilt als „nicht erkannt“)

Tabelle 6.2: Die Anzahl nicht korrekt erkannter Ziffern

Klasse	Anzahl	Fehlerrate
Ziffer 0	4	1.35%
Ziffer 1	2	0.59%

6.5 Weitere Anwendungen

Weitere Anwendungen sind mit dem Framework des semi-supervised Learnings einfach umzusetzen, wenn sich die Eingabe als Menge von hochdimensionalen Punkten modellieren lässt, zwischen denen es eine Ähnlichkeitsfunktion gibt und bei denen eine Interpolation Sinn ergibt.

Das Verfahren wurde hier für Funktionen beschrieben, welche hochdimensionale Punkte auf skalare Werte abbilden, aber es lässt sich auch auf vektorwertige Funktionen übertragen, siehe z.B. [MS11].

7 DEC und maschinelles Lernen

7.1 Einleitung

Das Ziel ist nun im semi-supervised learning das DEC-Framework zu verwenden, um Funktionen auf geometrischen Objekten besser zu approximieren. Dazu kann als Laplaceoperator, welcher beim Lernen bisher als Graph-Laplace generiert wurde nun der DEC-Laplaceoperator dienen, welcher besser an die Daten angepasst ist als ein Operator aus einem generiertem Graph, der ungünstige Parameter haben kann. Auch enthält das Mesh Informationen, welche aus einer reinen Punktemenge nicht extrahiert werden können, vorausgesetzt das Mesh wurde nicht selber aus einer Punktemenge erzeugt.

7.2 Anwendungen

Sinnvolle Anwendungen des maschinellen Lernens beim Discrete Exterior Calculus entstehen dadurch, dass ein Lernverfahren aus Ergebnissen des DEC lernen kann, aber auch der DEC mit dem Ergebnis einer gelernten Funktion als Startwert oder Randbedingung eine Berechnung durchführen kann. Im DEC kann ein solcher Startwert gesetzt werden, indem das Ergebnis einer Funktionsapproximation auf allen k-Simplices ausgewertet und dann als k-Form verwendet wird.

Wir betrachten nun die umgekehrte Richtung, nämlich das Ergebnis einer DEC-Berechnung mit Hilfe eines Kernverfahrens als Eingabe zum Lernen zu verwenden. Damit kann zum Beispiel die Berechnung verkürzt werden, indem Werte auf Punkten, die nicht explizit berechnet wurden, interpoliert werden können.

7.3 Der Laplacekern

Nachdem wir in Kapitel 5 verschiedene Kerne zum Lernen verwendet haben und dann zum semi-supervised Learning die Regularisierung mit der durch den Laplaceoperator induzierten Norm eingeführt haben, möchten wir nun einen eigenen Kern aus dem Laplaceoperator definieren.

Die Idee ist, dass ein solcher Kern mit einem Laplaceoperator, welcher die Struktur der Daten gut abbildet, es auch erleichtert eine Funktion auf diesen Daten zu lernen.

Wir definieren nun einen Laplacekern wie in [ZKLG06] beschrieben.

Definition 7.1. *Der Laplacekern ist über die Gram-Matrix zu $k : X \times X \to \mathbb{R}$ definiert, wobei X eine diskrete Menge von Punkten ist. Sei $G = (X, E)$ ein Graph dieser Punkte, und L ein diskreter Graph-Laplace von G. Dann ist die Gram-Matrix K des Kerns k mit einer Regularisierungsfunktion r definiert als $K_r := \sum_{i=1}^{n} r(\lambda_i)\phi_i\phi_i^T$, mit λ_i, ϕ_i als i-ter Eigenwert bzw. Eigenvektor der Matrix L.*

Als $r(\lambda_i)$ kommen mehrere Funktionen in Frage, unter anderen

- $r(\lambda_i) = \frac{1}{\lambda_i + \epsilon}$ (regularisierter Laplace),
- $r(\lambda_i) = \exp(-\frac{\sigma^2}{2}\lambda_i)$ (Diffusionskern),
- und $r(\lambda_i) = (\alpha - \lambda)^p$ mit $\alpha \geq 2$ (Random-Walk mit p Schritten).

Weitere Funktionen finden sich in [ZKLG06]. Im Folgenden wird $r(\lambda_i) = \frac{1}{\lambda_i^k}$ verwendet.

7.4 Äquivalenz der Funktionsnorm und der Laplace Seminorm

Für den Laplacekern wie er in Def. 7.1 definiert wurde, ist die Laplace-Seminorm wie sie in Formel 5.45 definiert ist äquivalent zur Funktionsnorm im RKHS.

Beweis: Für den Laplacekern sind $\|f\|_{\mathcal{H}}$ und $\|f\|_L$ äquivalent.

Sei $\mathbf{G} = \mathbf{G^T} = \sum_k^n \lambda_k \boldsymbol{\phi}_\mathbf{k} \boldsymbol{\phi}_\mathbf{k}^\mathbf{T}$ die Gram-Matrix des Laplacekerns, $\mathbf{a}$ der Koeffizientenvektor der Funktion und $\mathbf{f} = \mathbf{Ga}$ der Vektor der Funktionswerte $f(x_i)$ für alle $x_i \in X$. Weiterhin seien $\mathbf{L}$ die Laplace-Matrix und $(\lambda_i, \boldsymbol{\phi}_\mathbf{i})$ die Eigenwerte und Eigenvektoren von $\mathbf{L}$.

Dann gilt:

$$\begin{aligned}
\|f\|_L^2 &= \mathbf{a^T G^T L G a} && (7.1)\\
&= \mathbf{a^T G L G a} && (7.2)\\
&= \mathbf{a^T} \left(\sum_i^n \frac{1}{\lambda_i} \boldsymbol{\phi}_\mathbf{i} \boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \right) \mathbf{L} \left(\sum_j^n \frac{1}{\lambda_j} \boldsymbol{\phi}_\mathbf{j} \boldsymbol{\phi}_\mathbf{j}^\mathbf{T} \right) \mathbf{a} && (7.3)\\
&= \mathbf{a^T} \left(\sum_{i,j}^n \frac{1}{\lambda_i \lambda_j} \boldsymbol{\phi}_\mathbf{i} \boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \mathbf{L} \boldsymbol{\phi}_\mathbf{j} \boldsymbol{\phi}_\mathbf{j}^\mathbf{T} \right) \mathbf{a} && (7.4)\\
&= \mathbf{a^T} \left(\sum_{i,j}^n \frac{1}{\lambda_i \lambda_j} \boldsymbol{\phi}_\mathbf{i} \boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \lambda_j \boldsymbol{\phi}_\mathbf{j} \boldsymbol{\phi}_\mathbf{j}^\mathbf{T} \right) \mathbf{a} && (7.5)\\
&= \mathbf{a^T} \left(\sum_{i,j}^n \frac{1}{\lambda_i} \boldsymbol{\phi}_\mathbf{i} (\boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \boldsymbol{\phi}_\mathbf{j}) \boldsymbol{\phi}_\mathbf{j}^\mathbf{T} \right) \mathbf{a} && (7.6)\\
&= \mathbf{a^T} \left(\sum_{i,j}^n \frac{1}{\lambda_i} \boldsymbol{\phi}_\mathbf{i} \boldsymbol{\phi}_\mathbf{j}^\mathbf{T} \right) \mathbf{a} && (7.7)\\
&= \mathbf{a^T} \left(\sum_{i}^n \frac{1}{\lambda_i} \boldsymbol{\phi}_\mathbf{i} \boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \right) \mathbf{a} && (7.8)\\
&= \mathbf{a^T G a} = \|\mathbf{f}\|_\mathcal{H}^2 && (7.9)
\end{aligned}$$

□

In Schritt 7.5 wurde eingesetzt, dass $\mathbf{L}\boldsymbol{\phi}_\mathbf{k} = \lambda_k \boldsymbol{\phi}_\mathbf{k}$ gilt und in den Schritten 7.7 und 7.8 wurde verwendet, dass die Vektoren $\boldsymbol{\phi}$ eine Orthonormalbasis bilden und daher gilt:

$$\boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \boldsymbol{\phi}_\mathbf{j} = \begin{cases} 1 & \text{wenn } i = j \\ 0 & \text{wenn } i \neq j \end{cases}$$

$$\sum_i^n \boldsymbol{\phi}_\mathbf{i}^\mathbf{T} \boldsymbol{\phi}_\mathbf{j} = 1 \; \forall j$$

Andere Funktionen als $r(\lambda_i) \mapsto 1/\lambda_i$ entsprechen dabei anderen Varianten der Laplace-Seminorm. Verwendet man zum Beispiel $r(\lambda_i) \mapsto 1/\lambda_i^k$, entspricht die Norm im RKHS der Funktionsnorm $\|\cdot\|_{L^k} := \mathbf{a^T G^T L^k G a}$. Es ist leicht zu sehen, dass $(1/\lambda_i^k)\mathbf{L^k}\boldsymbol{\phi}_\mathbf{i} = \boldsymbol{\phi}_\mathbf{i}$ gilt und damit die gleiche Herleitung wie für $\|\cdot\|_L$ möglich ist.

7.5 DEC-Laplace vs. Graph-Laplace

Der Unterschied der verschiedenen Laplaceoperatoren wird beim Laplacekern besonders deutlich. Der Graph-Laplaceoperator sorgt dafür, dass der Kern nur zwischen benach-

barten Punkten wirkt, wodurch die Qualität des Kerns stark vom verwendeten Graphen abhängt. Ein generierter Graph kann gute Eigenschaften aufweisen und zum Beispiel hochdimensionale Punkte so erfassen, dass der Kern auf der Mannigfaltigkeit arbeitet, auf welcher die hochdimensionalen Punkte liegen. Eine ungünstige Wahl des Graphen kann aber auch zu schlechten Ergebnissen führen und es ist nicht immer klar, welche Wahl der Parameter sinnvoll ist.

Wir haben in Kapitel 3 den DEC-Laplaceoperator definiert, welcher ein vorhandenes Mesh verwendet. Wenn ein solches vorliegt hat man den entscheidenden Vorteil, dass das vorhandene Mesh die Informationen über die Mannigfaltigkeit auf der gerechnet werden soll bereits enthält und die Beziehung zwischen den Punkten nicht erst über Distanz, nächste Nachbarn oder ähnliche Methoden approximiert werden muss.

Zum Beispiel bei der Wärmeleitungsgleichung ist offensichtlich, dass die Wärme, die auf einem Objekt, das z.B. aus Metall besteht, sich vor allem auf der Oberfläche ausbreitet und selbst wenn sich zwei Stellen der Oberfläche sehr nahe kommen nur wenig Wärme durch die Luft dazwischen übertragen wird.

Zum Beispiel bei einem r-Graphen oder kNN-Graphen kann es passieren, dass nicht (direkt) verbundene Teile des Modells, welche nah beieinander liegen trotzdem verbunden werden und sich gegenseitig stark beeinflussen.

7.6 Ein Vergleich der Laplacekerne

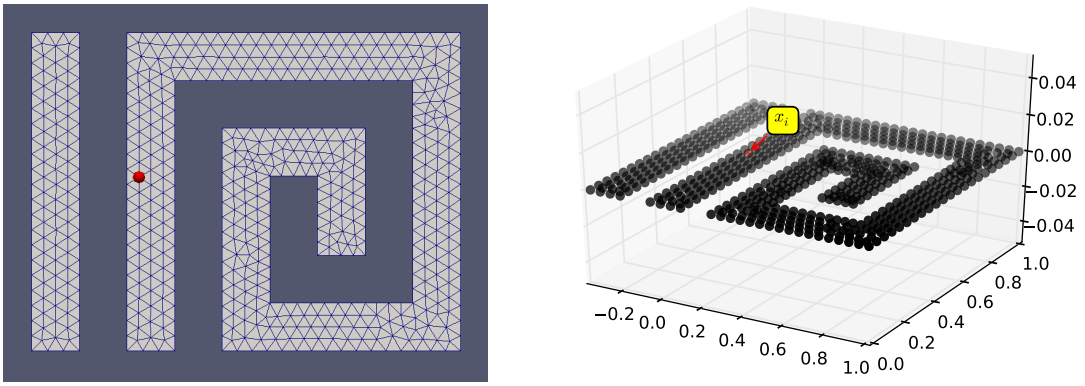


Abb. 7.1: Die Testdaten, auf denen der Laplacekern verwendet werden soll

Um den Effekt zu demonstrieren verwenden wir das Beispiel-Mesh in Abbildung 7.1, welches einerseits zwei getrennte Zusammenhangskomponenten enthält und andererseits durch die Spirale eine Geometrie enthält, bei welcher die intrinsischen Abstände auf dem Mesh mit der Distanz im Raum wenig zu tun haben. Der rote Punkt ist die Stelle x_i, welche wir festhalten um den Laplacekern $k_i(\cdot) = k(x_i, \cdot)$ auf allen Punkten $x_1, \dots, x_n$ des Meshes auszuwerten.

Ein passend gewählter Graph-Laplace kann die Information der Mannigfaltigkeit zwar oft gut erkennen, braucht dazu aber passend gewählte Parameter. Dies führt häufig zu einem „Tradeoff“ zwischen zu viel oder zu wenig verbundenen Punkten, siehe auch [MVLH08].

Hat man jedoch ein Mesh als Input gegeben, kann man den DEC-Laplace verwenden. Die Kantenmenge des Meshes definiert den Graphen und die Gewichtung der Kanten ist im DEC durch den metrikabhängigen Hodgestar-Operator gegeben. Dadurch, dass der DEC die vorhandenen Informationen über das Mesh komplett ausnutzt, schafft er es auch die Geometrie der Objekte exakt abzubilden.

Damit werden zum Beispiel verschiedene Zusammenhangskomponenten durch den DEC-Laplaceoperator *immer* korrekt getrennt, da der Operator sich aus dem Boundary- und Co-Boundary-Operator des Meshes zusammensetzt. Da diese gerade die Adjazenzmatrizen der Punkte sind und damit per Definition 0, wenn zwischen zwei Punkten keine Kante existiert, garantiert der DEC-Laplace, dass ein Funktionswert eines Punktes innerhalb einer ZHK keine Auswirkung auf Punkte einer anderen ZHK haben kann.

7.7 Ergebnis des Vergleichs

In einem gutem Ergebnis auf dem Mesh sollen sich Punkte, die auf der Mannigfaltigkeit weit voneinander entfernt liegen, unabhängig von ihrer Distanz im Raum wenig beeinflussen. Punkte aus unterschiedlichen Zusammenhangskomponenten sollten sich daher gar nicht beeinflussen, da die Distanz von zwei Punkten, zwischen denen es im Graphen keinen Pfad gibt, als unendlich definiert ist.

In Grafik 7.2 ist ein Kern aus einem Graph-Laplace mit einem r-Graphen dargestellt, und ein Laplacekern, welcher den DEC-Laplace als Operator verwendet.

Man kann gut erkennen, dass der Kern des Graph-Laplaceoperators zum r-Graphen die Struktur des Modells erfasst und sich dadurch Punkte, die geometrisch nah zusammen liegen, aber auf der Mannigfaltigkeit eine größere Distanz haben, weniger beeinflussen. Dennoch wird deutlich, dass einige Punkte mit geringer euklidischer Distanz sich trotzdem zu stark gegenseitig beeinflussen. Auch auf dem Teil des Meshes, welcher keine Verbindung zum Punkt x_i besitzt, hat der Kern überall einen Wert ungleich 0.

In dem Plot des Kerns zum DEC-Laplace hingegen ist zu erkennen, dass die ZHK, welche keine Verbindung zum Testpunkt hat, konstant 0 ist und auf dem Rest des Meshs der Wert des Kerns mit der Distanz entlang des Gitters abfällt.

Der Kern ist über eine Summe der Eigenvektoren definiert, welche aber nicht zwangsläufig über alle n Eigenvektoren läuft. Da $1/\lambda_i$ mit einfließt, fällt das Gewicht der Summanden ab, wenn die Summe über die Eigenwerte und Eigenvektoren in aufsteigender Reihenfolge der Eigenwerte läuft. Das Verhalten des Kerns hängt stark davon ab, wie viele Eigenvektoren für den Kern verwendet wurden.

In Grafik 7.3 sieht man, dass der DEC-Laplacekern unabhängig von der Zahl der Eigenvektoren die Komponenten des Modells trennt und zum Inneren der Spirale hin einen deutlichen Abfall des Funktionswertes zeigt. Der Abfall in der direkten Nähe des Punktes,

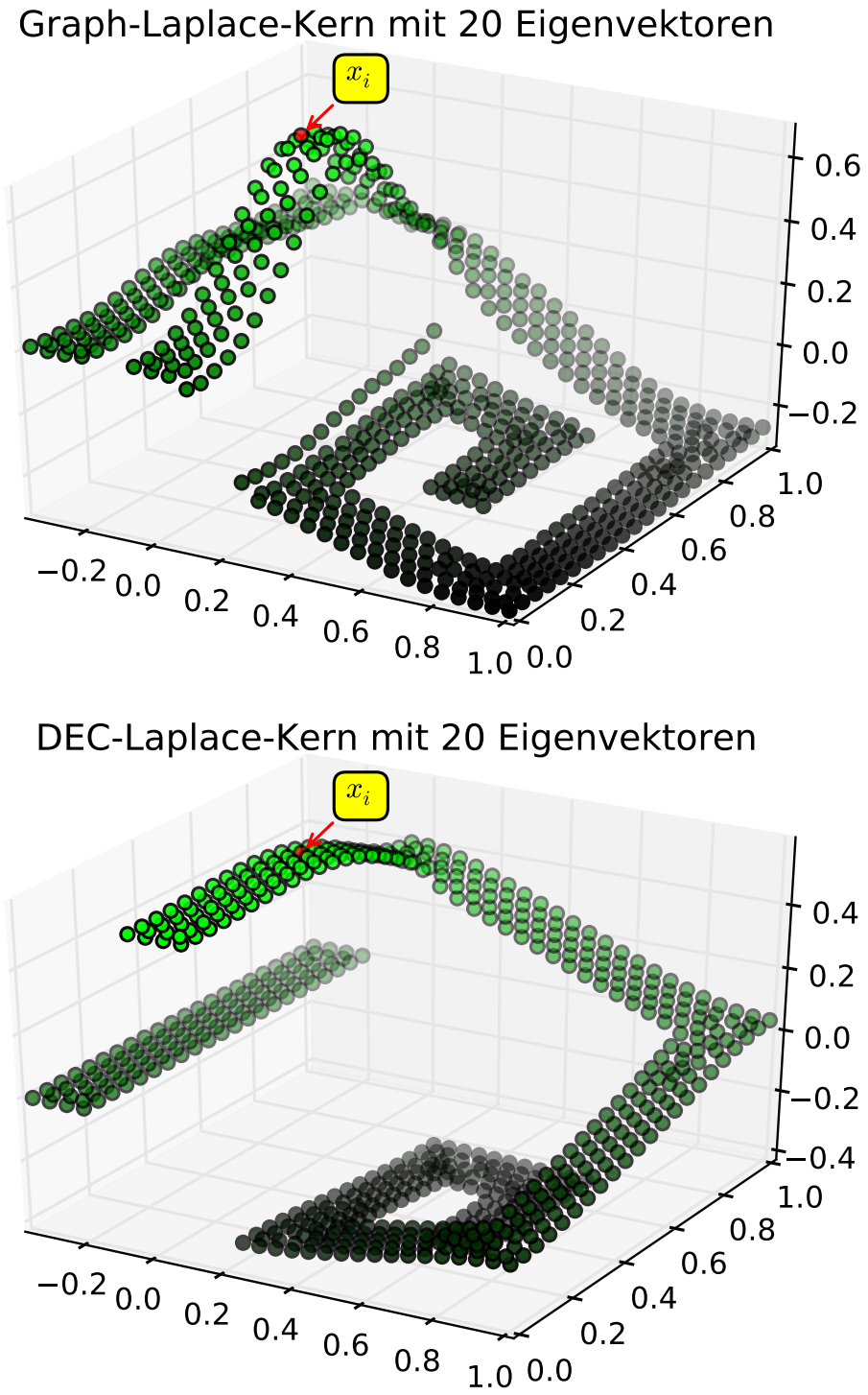


Abb. 7.2: Oben ist der Laplacekern zu einem Graph-Laplace mit r-Graphen und binärer Abstandsfunktion für $r = 0.16$ dargestellt und unten ein Laplacekern, der aus dem DEC-Laplaceoperator des Meshs erzeugt wurde. Für beide Kerne ist der Wert von $k(x_i, \cdot)$ auf allen Punkten dargestellt.

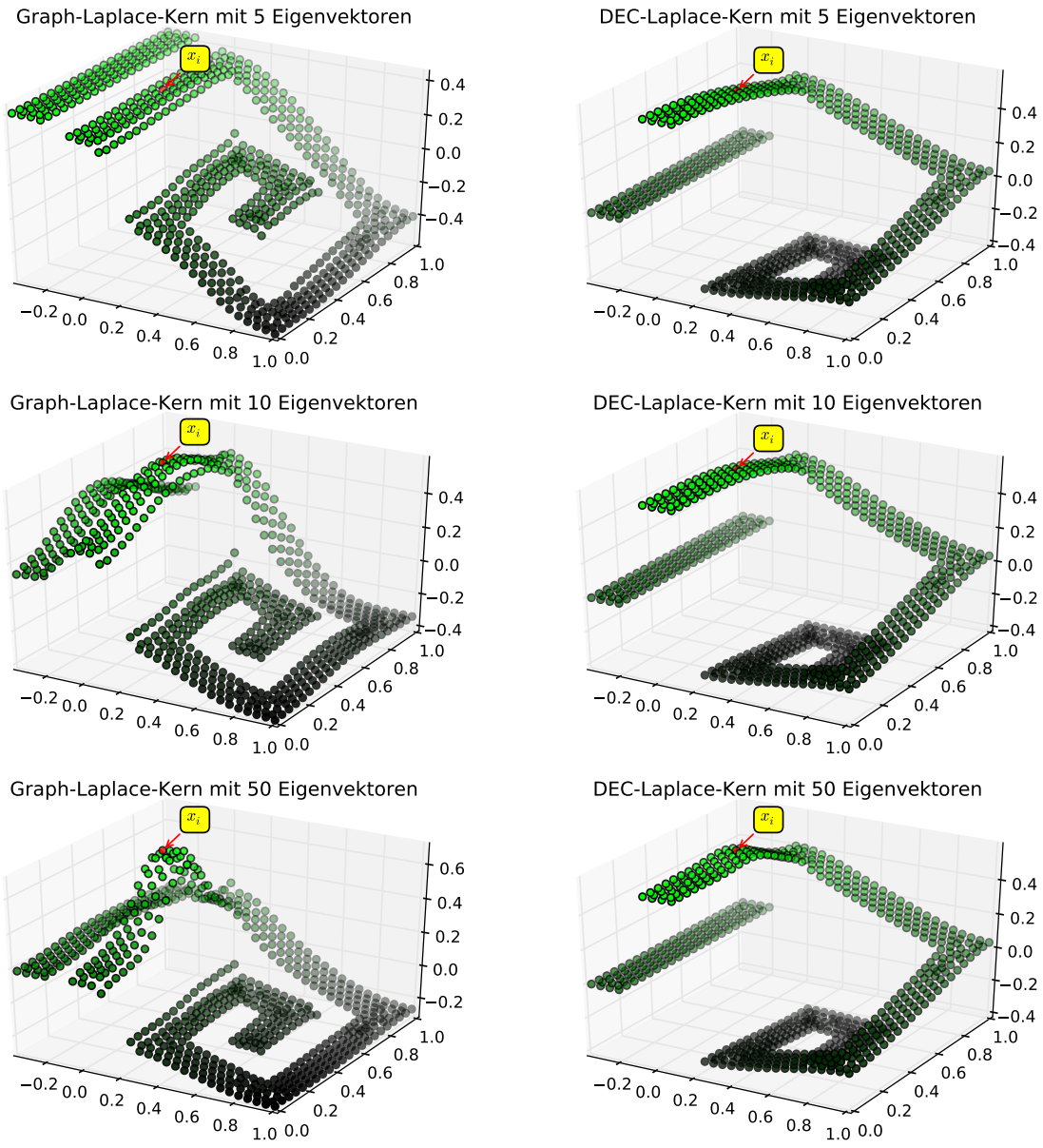


Abb. 7.3: Das Verhalten der beiden Laplacekerne in Abhängigkeit von der Anzahl verwendeter Eigenvektoren

an dem ausgewertet wird, hingegen hängt deutlich von der Zahl der verwendeten Vektoren ab.

Der Graph-Laplacekern für den r-Graphen hingegen benötigt mehr Eigenvektoren für ein gutes Ergebnis, welches erzielt wird wenn die Form der Funktion spitzer wird und dadurch weiter entfernte Punkte weniger stark beeinflusst werden.

Die Qualität des Graph-Laplacekerns lässt sich durch eine Anpassung des verwendeten Graphen und Parameter für das Problem optimieren. Eine gute Wahl der Parameter wird in [MVLH08] und [VL07] diskutiert.

Im zweiten Artikel wird als allgemeine Empfehlung der kNN-Graph genannt, da dieser unabhängig von der Skalierung der Daten funktioniert und in vielen Fällen die Punkte, welche tatsächlich auf der Mannigfaltigkeit benachbart sind, eher verbunden werden als weiter entfernte Punkte. Damit ist er auch robuster gegenüber ungünstig gewählten Parametern.

Im Beispiel aus Abbildung 7.1 würde ein kleinerer Radius des r-Graphen oder die Wahl eines kNN-Graphen das Ergebnis deutlich verbessern, allerdings versagen beide Ansätze bei Punkten, die auf der Mannigfaltigkeit weit voneinander entfernt sind, aber geometrisch nah zusammen liegen.

Ist zum Beispiel die Auflösung des Meshs gröber als die Distanz zwischen solchen Punkten, versagen viele Ansätze:

- Beim r-Graphen gibt es keinen Radius, sodass die Punkte auf dem grobem Mesh verbunden werden, aber die Punkte die sich an der engen Stelle nahe kommen nicht verbunden werden.
- Beim kNN-Graphen werden ebenfalls aufgrund der Distanz die falschen Punkte zuerst verbunden.
- Beim vollständigem Graphen ist die Gewichtung zwischen den nahen Punkten deutlich größer, als die Gewichtung zwischen den Punkten im Mesh.

Ein Beispiel, welches mit einem r-Graphen und einem kNN-Graphen nicht korrekt erfasst werden kann ist in Grafik 7.4 zu sehen, wobei die roten Kanten die des erzeugten Graphen sind. Ein Beispielpunkt wurde jeweils grün markiert. Beim kNN-Graphen sind die drei Nachbarn gezeigt und beim r-Graphen ist der Radius um den Beispielpunkt herum hervorgehoben.

Beim r-Graphen muss der Radius groß werden bevor die Punkte, welche auf dem Mesh benachbart sind überhaupt verbunden werden, aber schon mit relativ kleinem Radius entstehen Kanten zwischen den beiden ZHK.

Der kNN-Graph mit 3 Nachbarn weist etwas bessere Eigenschaften auf, da die ZHK zusammenhängend bleiben, aber auch hier gilt für alle $k > 0$, dass als erstes die Kanten zwischen den verschiedenen ZHK zum Graphen hinzugefügt werden, bevor Punkte innerhalb des Meshs der ZHK verbunden werden.

Gerade in solchen Fällen ist es sehr hilfreich, wenn die zusätzlichen Informationen über die Konnektivität in Form der Kanten des Meshes mit in die Lösung des Lernproblems

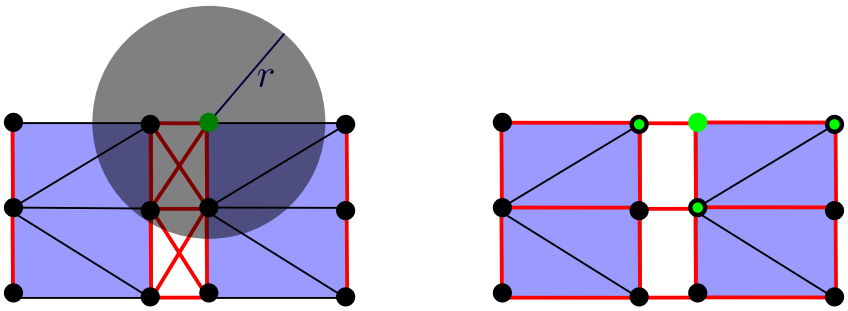


Abb. 7.4: Links ist ein r-Graph der Punkte des Meshs zu sehen, rechts ein kNN-Graph mit $k = 3$. Für jeweils einen der Punkte sind die zugehörigen Kanten hervorgehoben.

einfließen, wie es bei einem Kern aus dem Laplaceoperator des Discrete Exterior Calculus der Fall ist.

Lernen mit dem Laplacekern

Bei der Verwendung des so definierten Kerns mit $r(\lambda_i) = 1/\lambda_i^k$ für verschiedene k und verschiedene Regularisierungsparameter λ_A ließen sich bei den verwendeten Testdaten keine Parameter finden, bei denen die gelernte Funktion eine gute Generalisierung zeigt. Das Lernen mit dem Kern funktionierte auf synthetischen Testdaten etwas besser als auf den Eingaben aus DEC-Berechnungen, sodass zu vermuten ist, dass sorgfältig an das Problem angepasste Parameter zu besseren Ergebnissen führen, wozu jedoch erst einmal bekannt sein muss welche Parameter ein bestimmtes Problem erfordert. Die Trainingsdaten wurden bei den meisten Kombinationen der Parameter gut getroffen, aber eine stärkere Regularisierung führte nur zu geringeren Funktionswerten (und damit größerem Fehler) auf den Trainingspunkten, nicht aber zu einer besseren Generalisierung auf ungelernten Testpunkten. In der gesamten Testreihe wurde $\lambda_I = 0$ verwendet, da wir in Abschnitt 7.4 gesehen haben, dass beim Laplacekern die Regularisierung durch die Funktionsnorm äquivalent zu einer Regularisierung mit der Laplace-Seminorm ist.

7.8 Lernen aus DEC-Berechnungen

Mit dem Discrete Exterior Calculus als Werkzeug um PDEs zu lösen und der Methode des Lernens mit Kernfunktionen, welche durch den Laplaceoperator, welcher mittels DEC für das Mesh erzeugt wird und im Lernprozess genutzt werden kann um eine an die Daten angepasste Kernfunktion zu definieren, kann man nun Daten die auf Punkten des Meshs liegen lernen.

Dabei interessieren insbesondere zwei Problemstellungen:

- Interpolation von Werten auf einem feinen Mesh aus einer Anzahl gegebener Werte ohne Fehler.

- Interpolation von Werten auf einem feinen Mesh aus Werten, die auf einem grobem Mesh berechnet wurden.

Bei der ersten Problemstellung ist die Anwendung, dass reale Daten z.B. aus einer Messung gegeben sind und der Fehler als vernachlässigbar klein angenommen werden kann. Nun möchte man aus möglichst wenig Messpunkten eine Lösung mit möglichst kleinem Fehler auf dem Mesh berechnen.

Die zweite Problemstellung verwendet auch für die Eingabedaten nur eine Simulation, spart aber Rechenzeit dadurch, dass diese nur auf einem grobem Gitter berechnet werden muss und die fehlenden Punkte auf dem feinem Gitter interpoliert werden. Hier ist der Fehler nach unten beschränkt durch die Genauigkeit der Rechnung auf dem grobem Gitter, weswegen es nicht möglich ist, den absoluten Fehler für eine optimale Lösung zu minimieren, sondern nur den Fehler daraufhin zu optimieren, dass er nicht schlechter wird als der Fehler der Daten auf dem grobem Gitter.

Lernen auf einem feinem Mesh

Mit einem Mesh, das eine „U-Form“ hat, bei welcher ein Ende den Wert eins als Randbedingung hat ist es für Verfahren, die nur auf den Punkten arbeiten schwierig die Funktion korrekt zu approximieren.

Ein solches Modell der Größe $1 \times 1 \times 0.1$ wurde in verschiedenen Auflösungen trianguliert und die Trainingsdaten durch eine Simulation mit Hilfe des DEC-Framework aus Kapitel 3 berechnet. Auf dem Modell, wie es in Abbildung 7.5 zu sehen ist wurde mit einer Zeitschrittweite von $10^{-5}\,\mathrm{s}$ die Wärmeleitung über eine Zeit von $1.0\,\mathrm{s}$ hinweg berechnet um dann das Ergebnis als Trainingsdaten zum Lernen zu verwenden.

Beim Lernen wurden alle Punkte der jeweiligen Triangulation für den Laplaceoperator verwendet. Die Punkte mit Trainingslösung werden gewählt, indem solange Punkte mit der größten Hausdorffdistanz (d.h. der minimale Abstand des neuen Punkts zu allen bisher gewählten ist maximal) zu den bisher gewählten Punkten hinzugefügt werden, bis die gewünschte Menge an Trainingspunkten erreicht ist. Damit ist es möglich eine gute Abdeckung der Punkte des groben Meshs zu erreichen. Die Parameter beim Lernen sind in Tabelle 7.1 aufgelistet.

Der Fehler der Interpolation auf den ungelernten Punkten ist in Tabelle 7.2 zu sehen, wobei die Fehler bei verschiedenen Auflösungen sind nicht direkt vergleichbar sind, da jeweils die Anzahl an Testpunkten auf den Meshes unterschiedlich ist, weil Trainingspunkte und Testpunkte aus dem selbem Mesh gewählt wurden.

Die Ausreißer bei einem großem Anteil Trainingspunkte sind dadurch zu erklären, dass die Abdeckung durch Punkte mit großer Hausdorffdistanz bei vielen Testpunkten auf dem gleichem Mesh weniger gleichmäßig wird, da die Bereiche mit bisher geringer Punktdichte nacheinander aufgefüllt werden. Da bei dieser Wahl der Punkte gerade in diesen Bereichen dann mehr Testpunkte als Trainingspunkte liegen, ergibt sich dort ein größer Fehler.

Es sind nur die Fehler für 20% bis 90% aufgetragen, da die Daten bei weniger Trainingspunkten nicht mehr aussagekräftig sind und bei 100% Trainingspunkten der Fehler

Tabelle 7.1: Die Parameter beim semi-supervised Lernen auf dem Mesh

Kern	Gaußkern
σ	10.2
Graph-Laplace	Der DEC-Laplaceoperator des Meshs
Eingabe	Die dreidimensionalen Punkte des Meshs mit ihrem durch den DEC berechnetem Wert nach 10^5 Zeitschritten.
γ_I (Laplace-Regularisierung)	0.001
γ_A (RKHS-Regularisierung)	0.0

Exakte Anzahl Lernpunkte und Testpunkte bei einem Mesh mit 2188 Punkten:

Prozent	20	30	40	50	60	70	80	90
Lernpunkte	437	656	875	1094	1312	1531	1750	1969
Testpunkte	1751	1532	1313	1094	876	657	438	219

Exakte Anzahl Lernpunkte und Testpunkte bei einem Mesh mit 506 Punkten:

Prozent	20	30	40	50	60	70	80	90
Lernpunkte	101	151	202	253	303	354	404	455
Testpunkte	305	355	304	253	203	152	102	51

Tabelle 7.2: Die Fehler der Interpolation auf Triangulierungen verschiedener Auflösung mit unterschiedlichem Anteil an Trainingspunkten. Horizontal: Anzahl Punkte des Meshs (Pkt). Vertikal: Anteil der Punkte, welche als Trainingspunkte gewählt wurden (%).

% \ Pkt	308	506	933	2188	6314
90	0.133	0.112	0.148	0.050	0.147
80	0.138	0.134	0.176	0.067	0.321
70	0.201	0.184	0.231	0.223	0.051
60	0.180	0.213	0.141	0.155	0.353
50	0.233	0.242	0.221	0.241	0.241
40	0.254	0.307	0.248	0.275	0.450
30	0.261	0.284	0.271	0.260	0.261
20	0.247	0.276	0.258	0.258	0.576

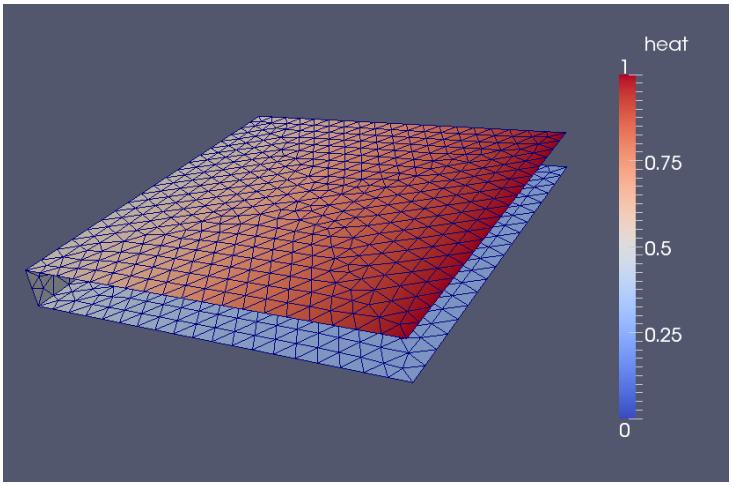


Abb. 7.5: Auf der Kante am oberen Ende des gebogenen Modells mit 933 Punkten ist der Wert 1 als Randbedingung gegeben. Die Wärme breitet sich entlang des Meshes aus, sodass die Kante am unterem Ende der Form den geringsten Wert aufweist.

mangels Testpunkten nicht messbar ist.

Die Konvergenzraten der Lösung bezüglich des Anteils der Punkte, welche eine Trainingslösung haben, sind in Abb. 7.6 abgebildet.

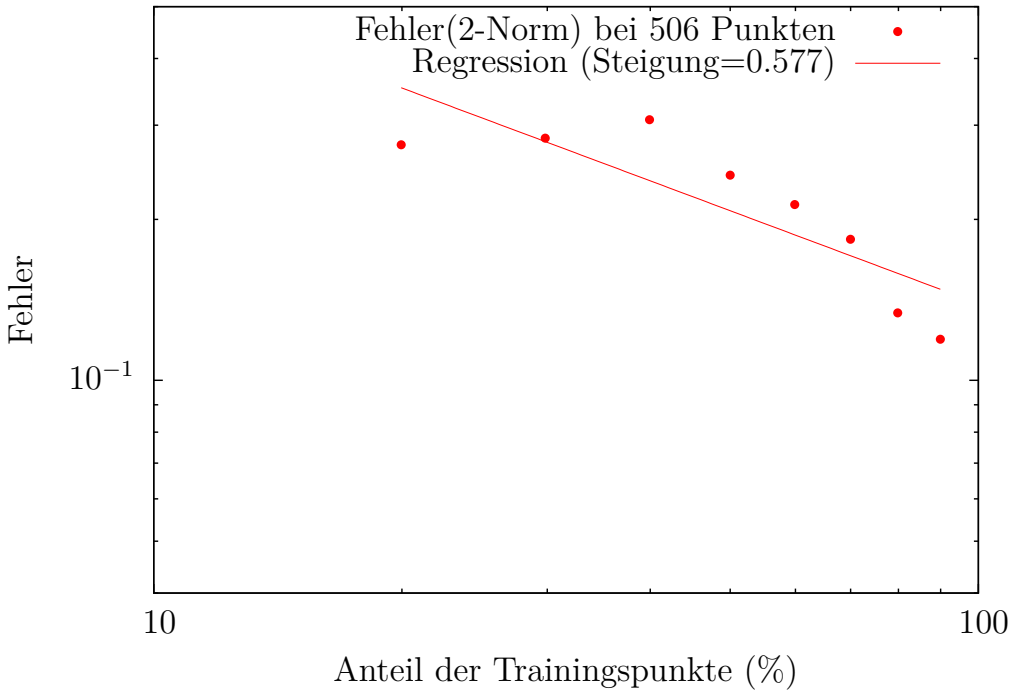
Lernen auf einem grobem Mesh

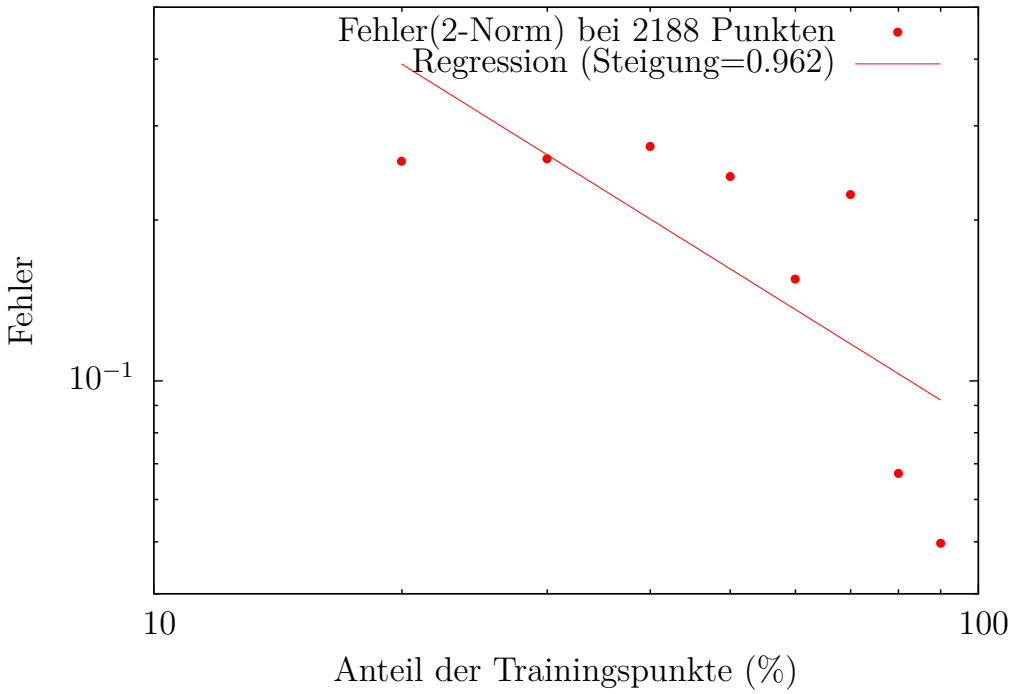
In der zweiten Variante sind die Werte auf den Punkten eines groben Meshs aus einer Simulation bekannt und die Werte auf den Punkten eines feinen Meshs sollen interpoliert werden, sodass auf dem feinen Mesh keine Simulation ausgeführt werden muss. Der Fehler nicht absolut minimiert werden, da die Trainingspunkte selber bereits einen Fehler aus der Simulation aufweisen, welcher den Fehler der berechneten Lösung nach unten beschränkt. Daher ist das Ziel, dass der Interpolationsfehler möglichst nah am Fehler der Trainingsdaten bleibt.

Zum Testen des Verfahrens wurde die Rechnung auf Meshes mit verschieden vielen Punkten ausgeführt, damit der Fehler der Trainingspunkte in Bezug auf das feinste Mesh ermittelt werden kann, indem die Punkte des feinen Meshs als Testpunkte verwendet werden um den Interpolationsfehler zu messen.

Es wird das gleiche Modell wie in der vorherigen Rechnung verwendet, nur dass es beim Lernen in zwei verschiedenen Triangulierungen vorliegt, eine mit geringer Auflösung, auf der das Lernverfahren benutzt wird und eine mit feiner Auflösung, auf der die Funktion zum messen des Fehlers ausgewertet wird.

Damit die Trainingspunkte zwischen grobem Gitter und feinem Gitter vergleichbar sind, verwenden wir die Verfeinerung aus Abschnitt 3.12, wodurch vermieden werden kann schon beim Vergleich der Trainingswerte zwischen den beiden Gittern interpolieren





Anzahl Punkte	308	506	933	2188	6314
Mittlerer Fehler (2-Norm)	0.231	0.230	0.217	0.209	0.338
Konvergenzrate (2-Norm)	0.438	0.577	0.368	0.962	0.854

Abb. 7.6: Konvergenzraten der Lösung bezüglich des Anteils an Trainingspunkten.

Tabelle 7.3: Die Parameter beim semi-supervised Lernen auf den verschiedenen groben Meshes

Kern	Gaußkern
σ	1.0
Graph-Laplace	Der DEC-Laplaceoperator des groben Meshs
Eingabe	Die dreidimensionalen Punkte des groben Meshs mit ihrem durch den DEC berechnetem Wert nach 10^5 Zeitschritten.
Ausgabe	Der mittlere Fehler in der 2-Norm auf den Punkten des feinen Meshs, die nicht im groben Mesh enthalten sind.
γ_I (Laplace-Regularisierung)	0.00001
γ_A (RKHS-Regularisierung)	0.0

Exakte Anzahl Lernpunkte und Testpunkte bei einem Mesh mit 4541 Punkten:

Prozent	10	20	30	40	50	60	70	80	90	100
Lernpunkte	483	940	1352	1863	2296	2759	3204	3664	4139	4541
Testpunkte	4058	3601	3189	2678	2245	1782	1337	877	402	0

Exakte Anzahl Lernpunkte und Testpunkte bei einem Mesh mit 1167 Punkten:

Prozent	10	20	30	40	50	60	70	80	90	100
Lernpunkte	135	234	328	473	575	687	809	930	1058	1167
Testpunkte	1032	933	839	694	592	480	358	237	109	0

Exakte Anzahl Lernpunkte und Testpunkte bei einem Mesh mit 308 Punkten:

Prozent	10	20	30	40	50	60	70	80	90	100
Lernpunkte	35	53	79	120	145	179	210	242	278	308
Testpunkte	273	255	229	188	163	129	98	66	30	0

Tabelle 7.4: Der Fehler der Trainingsdaten des Mesh mit geringerer Auflösung in Bezug auf das Mesh der höchsten Auflösung in der 2-Norm.

Anzahl Punkte	308	1167	4541
Fehler der Trainingswerte (2-Norm)	0.011796	0.006276	0.003473

Tabelle 7.5: Die Fehler der Interpolation auf dem feinen Mesh mit einer Funktion die auf gröberen Triangulierungen gelernt wurde. Horizontal (Pkt): Anzahl Punkte des Meshs auf dem gelernt wurde. Vertikal (%): Anteil der Punkte, welche als Trainingspunkte gewählt wurden.

% \ Pkt	308	1167	4541
100	0.0119	0.0136	0.0158
90	0.0144	0.0788	0.1331
80	0.0204	0.3802	0.1108
70	0.0277	0.0654	0.1220
60	0.0311	0.0574	0.1678
50	0.0346	0.0617	0.1795
40	0.0372	0.0643	0.1301
30	0.0334	0.0797	0.1368
20	0.0323	0.0771	0.1844

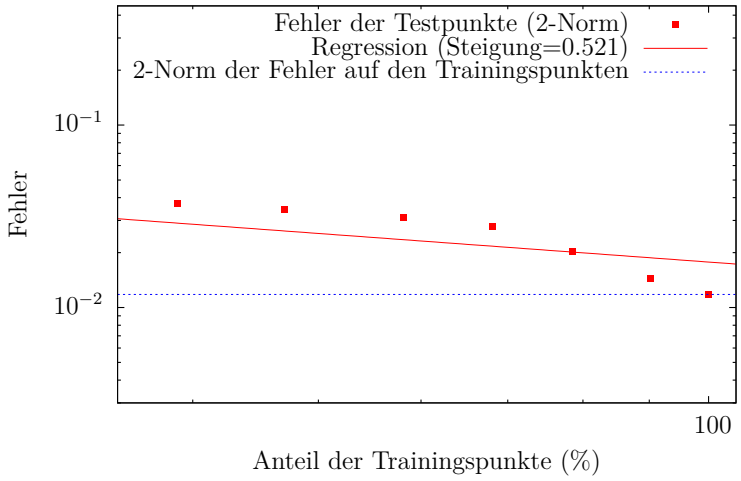
zu müssen. Als Testpunkte werden die Punkte des feinen Meshs gewählt, welche auf dem groben Mesh nicht vorhanden sind und als Mannigfaltigkeitspunkte werden die Punkte des groben Meshs gewählt, welche keine Trainingspunkte sind.

Der Fehler der Trainingswerte der verschiedenen Meshes in Bezug auf das am feinsten aufgelöste Mesh mit 17913 Punkten sind in Tabelle 7.4 zu sehen. In Abbildung 7.7 sind die Konvergenzplots für das Lernen der Daten auf Meshes mit unterschiedlichen Auflösungen im Vergleich zu einer Rechnung auf dem feinsten Mesh zu sehen.

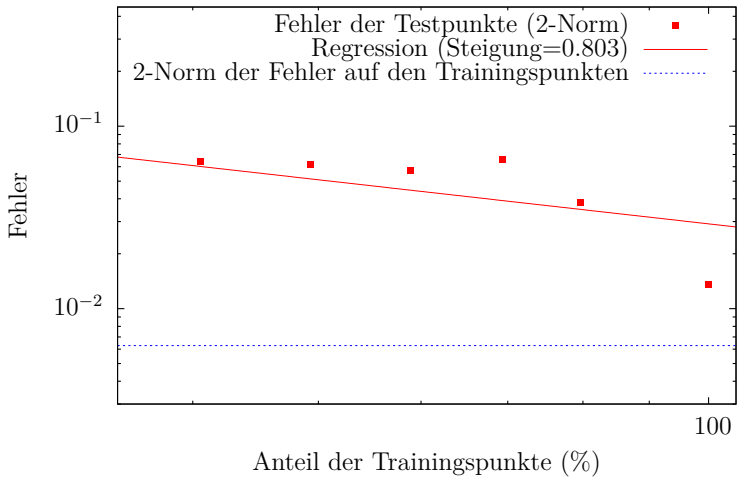
In den Konvergenzplots in Abbildung 7.7 sind die Fehler im Bereich von 40% bis 100% aufgetragen, da die Messungen mit 10 bis 30 Prozent der Punkte nicht aussagekräftig waren. Im Konvergenzplot zum Mesh mit 1167 Punkten wurde der Wert für die Messung mit 90% der Punkte entfernt, da die Punkte bei der Messung unglücklicherweise gerade so gewählt wurden, dass die Hausdorffdistanz der Punkte etwa gleich groß war wie bei den Werten für 70% und 80% Trainingspunkte und der Fehler mit einem Wert von 0.0789 ein deutlicher Ausreißer nach oben. Die Fehler der Funktion in der 2-Norm sind in Tabelle 7.5 aufgelistet.

Während die Funktion auf dem gröbsten Mesh so gelernt werden kann, dass sie mit 308 Kernvektoren (100% der Punkte des groben Meshs) auf den Testpunkten im feinen Mesh einen Fehler hat, der nur um $6.69 \cdot 10^{-5}$ vom Simulationsfehler der Trainingspunkte im groben Mesh abweicht, ergeben sich bei feineren Meshes deutlich größere Fehler auf den Testpunkten, selbst wenn sämtliche Punkte des groben Gitters gelernt wurden.

Fehler einer Funktion, die auf einem Mesh mit 308 Punkten gelernt wurde:



Fehler einer Funktion, die auf einem Mesh mit 1167 Punkten gelernt wurde:



Fehler einer Funktion, die auf einem Mesh mit 4541 Punkten gelernt wurde:

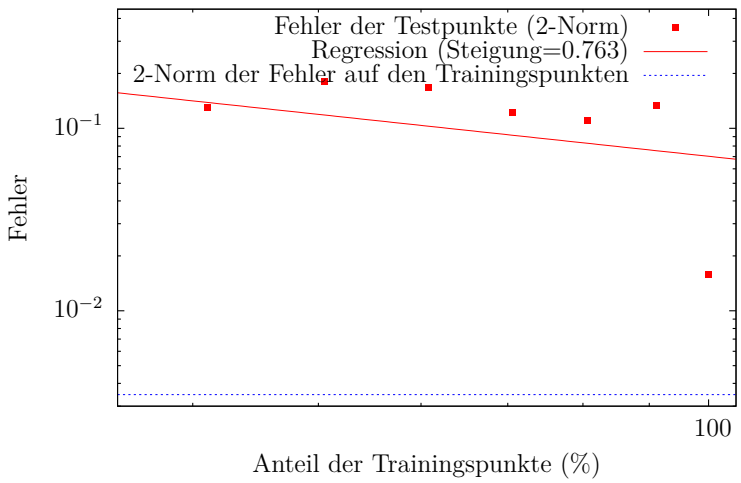


Abb. 7.7: Das Ergebnis der Auswertung einer Funktion, die auf einem grobem Mesh gelernt wurde, auf einem feinerem Gitter.

Da die Methode des Lernens mit einem Kern im Allgemeinen mit mehr Punkten bessere Ergebnisse erzielen kann, liegt der Fehler vermutlich an einer ungünstigen Wahl der Punkte durch die Hausdorffdistanz. Auch kann es bei zu feinen Meshs der Lernpunkte zu einer numerisch instabilen Kernmatrix kommen, da sich die Werte der Kernfunktion zwischen zwei nahen Punkten nur noch wenig unterscheiden.

8 Zusammenfassung

In dieser Arbeit wurden zwei große Themen erklärt, bei welchen es zunächst scheint, als gäbe es keine Gemeinsamkeiten, um dann den Zusammenhang herauszustellen und zu untersuchen inwiefern mit der Kombination beider Methoden die Ergebnisse verbessert werden können.

Zuerst wurde der *Discrete Exterior Calculus* als mächtiges Framework um PDEs zu lösen definiert, welches Fehler durch Anwendung von Operatoren wie dem Differentialoperator oder auch dem Laplaceoperator vermeidet, welches dann verwendet wurde um verschiedene physikalische Probleme zu diskretisieren. Die Rechnungen haben gezeigt, dass gute Ergebnisse möglich sind und sich mit dem dazu entwickelten Programm sowohl auf zweidimensionalen Oberflächenmodellen, als auch auf dreidimensionalen Tetraedermeshes, welche z.B. für Simulationen von Mechanikproblemen nötig sind, rechnen lässt. Es ist auch möglich aus hochdimensionalen Daten Simplex-Komplexe zu berechnen, auf welchen dann mit den gleichen Operatoren in höheren Dimensionen gerechnet werden kann, was unter anderem für die Kombination mit dem maschinellem Lernen interessant ist.

Das *maschinelle Lernen* mit Kernen wurde im nächsten Kapitel eingeführt, um zu zeigen, dass Kerne es ermöglichen auch für schwierige Funktionen eine gute Approximation zu finden. Dabei wurde der Laplaceoperator als ein zusätzlicher Regularisierungsterm beim *semi-supervised learning* betrachtet, um zu sehen, dass er hilft die Struktur der Daten besser zu erfassen und in hochdimensionalen Daten Mannigfaltigkeiten niedrigerer Dimension zu erkennen zu können.

Damit war es möglich zusätzliche Trainingsdaten ohne bekannten Trainingswert zu verwenden, welche helfen das zu lösende Problem zu erkennen, um das Ergebnis des Lernens deutlich zu verbessern. So ließen sich dann auch Probleme lösen, deren Trainingsdaten selber nur eine schlechte Approximation zulassen, bei denen aber viele Datenpunkte mit unbekanntem Wert bekannt sind.

Schließlich wurden die beiden Themen kombiniert, um mit dem Laplaceoperator des Discrete Exterior Calculus das maschinelle Lernen auf Daten, welche auf den Punkten eines Meshs definiert sind, zu optimieren. Am Beispiel eines Laplacekerns, welcher an die Daten auf denen er arbeitet angepasst ist, wurde deutlich, dass ein generierter Laplaceoperator für ein Mesh und der DEC-Laplaceoperator des Meshs einen großen Unterschied aufweisen und dass es sogar Meshes gibt, bei denen es nicht möglich ist einen Laplaceoperator aus den Punkten zu generieren, welcher es schafft die Struktur des Meshs so ausnutzen, wie der DEC-Laplaceoperator es ermöglicht.

Ausblick

Gerade im Bereich des Discrete Exterior Calculus liegen noch einige interessante Themen. Es gibt viele Probleme, die sich mit dieser Methode effizient lösen lassen, wie zum Beispiel eine Berechnung der Navier-Stokes-Gleichung über die Wirbelstärke [ETK+07] oder auch die Berechnung inkompressibler Elastizität in der Strukturmechanik [AY13].

Auch DEC-Berechnungen mit einem Hodgestar, welcher baryzentrische Mittelpunkte verwendet, ist von Interesse, da dieser es ermöglicht auf die Bedingung zu verzichten, dass alle Dreiecke wohlzentriert sein müssen. Dies ist insbesondere nützlich für Meshes, welche aus Rechtecksgittern erzeugt wurden und daher rechte Winkel aufweisen. Bei diesen wird der circumcentrische Hodgestar bei Winkeln nahe 90° instabil und bei exakt 90° ist er nicht definiert, da das Volumen der dualen Kante zum Face, welches dem rechten Winkel gegenüber liegt, 0 wird.

Im Bereich des semi-supervised learnings ist interessant, welche Laplaceoperatoren für verschiedene Problemstellungen die besten Ergebnisse liefern, mit welchen Verfahren die Parameter für die Graph-Laplaceoperatoren gewählt werden sollten und eine weitergehende Untersuchung der Kombination von Mesh-Laplaceoperatoren, wie dem des DEC, mit der Laplace-Regularisierung bzw. verschiedenen Laplacekernen.

Literaturverzeichnis

[Aro50] Aronszajn, Nachman: *Theory of reproducing kernels.* Transactions of the American mathematical society, 68(3):337–404, 1950. `http://www.jstor.org/stable/1990404`.

[AY13] Angoshtari, Arzhang und Arash Yavari: *A Geometric Structure-Preserving Discretization Scheme for Incompressible Linearized Elasticity.* 2013.

[BEHW87] Blumer, Anselm, Andrzej Ehrenfeucht, David Haussler und Manfred K Warmuth: *Occam's razor.* Information processing letters, 24(6):377–380, 1987.

[Bel56] Bellman, Richard: *On a routing problem.* Technischer Bericht, DTIC Document, 1956.

[Bel86] Bellman, Richard: *Dynamic programming and lagrange multipliers.* The Bellman Continuum: A Collection of the Works of Richard E. Bellman, Seite 49, 1986.

[Ber97] Berkeley, Istvan SN: *A revisionist history of connectionism.* Unpublished manuscript, 1997. `http://www.universelle-automation.de/1969_Boston.pdf`.

[BH09] Bühler, Thomas und Matthias Hein: *Spectral clustering based on the graph p-Laplacian.* In: *Proceedings of the 26th Annual International Conference on Machine Learning*, Seiten 81–88. ACM, 2009.

[BM76] Bondy, John Adrian und Uppaluri Siva Ramachandra Murty: *Graph theory with applications*, Band 290. Macmillan London, 1976.

[BN04] Belkin, Mikhail und Partha Niyogi: *Semi-supervised learning on Riemannian manifolds.* Machine learning, 56(1-3):209–239, 2004.

[BNS06] Belkin, Mikhail, Partha Niyogi und Vikas Sindhwani: *Manifold regularization: A geometric framework for learning from labeled and unlabeled examples.* The Journal of Machine Learning Research, 7:2399–2434, 2006.

[CDS03] Carlsson, Gunnar und Vin De Silva: *Topological approximation by small simplicial complexes.* preprint, 2003.

[CFL28] Courant, Richard, Kurt Friedrichs und Hans Lewy: *Über die partiellen Differenzengleichungen der mathematischen Physik*. Mathematische Annalen, 100(1):32–74, 1928.

[CGS09] Croce, R., M. Griebel und M. A. Schweitzer: *Numerical Simulation of Bubble and Droplet-Deformation by a Level Set Approach with Surface Tension in Three Dimensions*. International Journal for Numerical Methods in Fluids, 62(9):963–993, 2009. Also available as SFB 611 Preprint no 431.

[Col] Colins, Karen D.: *Cayley-Menger Determinant*. `http://mathworld.wolfram.com/Cayley-MengerDeterminant.html`, From MathWorld–A Wolfram Web Resource. last checked: 2013.10.25.

[Dau04] Daumé III, Hal: *From Zero to Reproducing Kernel Hilbert Spaces in Twelve Pages or Less*. Available at `http://www.isi.edu/~hdaume/docs/daume04rkhs.ps`, February 2004.

[DKT08] Desbrun, Mathieu, Eva Kanso und Yiying Tong: *Discrete differential forms for computational modeling*. In: *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia '08, Seiten 15:1–15:17, New York, NY, USA, 2008. ACM.

[EKS83] Edelsbrunner, H., D. Kirkpatrick und R. Seidel: *On the shape of a set of points in the plane*. Information Theory, IEEE Transactions on, 29(4):551–559, 1983, ISSN 0018-9448.

[ES05] Elcott, S. und P. Schröder: *Building your own DEC at home*. In: *ACM SIGGRAPH 2005 Courses*, Seite 8. ACM, 2005.

[ETK+07] Elcott, Sharif, Yiying Tong, Eva Kanso, Peter Schröder und Mathieu Desbrun: *Stable, circulation-preserving, simplicial fluids*. ACM Trans. Graph., 26(1), Januar 2007, ISSN 0730-0301. `http://doi.acm.org/10.1145/1189762.1189766`.

[Fie93] Fiesler, E: *Minimal and High Order Neural Network Topologies*. In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Band 2204, Seite 173, 1993.

[Flo56] Flood, Merrill M: *The traveling-salesman problem*. Operations Research, 4(1):61–75, 1956.

[FT87] Fredman, Michael L und Robert Endre Tarjan: *Fibonacci heaps and their uses in improved network optimization algorithms*. Journal of the ACM (JACM), 34(3):596–615, 1987.

[GD98] Gardner, MW und SR Dorling: *Artificial neural networks (the multilayer perceptron)–a review of applications in the atmospheric sciences*. Atmospheric environment, 32(14-15):2627–2636, 1998.

[GH88] Goldberg, David E und John H Holland: *Genetic algorithms and machine learning*. Machine learning, 3(2):95–99, 1988.

[GHL03] Grigor'yan, Alexander, Jiaxin Hu und Ka Sing Lau: *Heat kernels on metric measure spaces and an application to semilinear elliptic equations*. Transactions of the American Mathematical Society, 355(5):2065–2095, 2003.

[Gre13] Gretton, Arthur: *Introduction to RKHS, and some simple kernel algorithms*. `http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/lecture4_introToRKHS.pdf`, lecture notes, 2013.

[Hav99] Haveliwala, Taher: *Efficient computation of PageRank*. 1999. `http://research.taherh.org/pubs/efficient-pr.pdf`, Stanford University Technical Report.

[Hir03] Hirani, A.N.: *Discrete exterior calculus*. Dissertation, California Institute of Technology, 2003.

[HNC08] Hirani, A.N., K.B. Nakshatrala und J.H. Chaudhry: *Numerical method for Darcy flow derived using Discrete Exterior Calculus*. arXiv preprint arXiv:0810.3434, 2008.

[HSS05] Hofmann, Thomas, Bernhard Schölkopf und Alexander J Smola: *A Tutorial Review of RKHS Methods in Machine Learning*. 2005.

[HW79] Hartigan, John A und Manchek A Wong: *Algorithm AS 136: A k-means clustering algorithm*. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1):100–108, 1979.

[Koh90] Kohonen, Teuvo: *The self-organizing map*. Proceedings of the IEEE, 78(9):1464–1480, 1990.

[Kön01] König, Dénes: *Theorie der endlichen und unendlichen Graphen*. AMS Bookstore, 2001.

[Kru56] Kruskal, Joseph B: *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proceedings of the American Mathematical society, 7(1):48–50, 1956.

[LC98] LeCun, Yann und Corinna Cortes: *The MNIST database of handwritten digits*, 1998. `http://yann.lecun.com/exdb/mnist/`.

[Mit] Mitra, Ambar K: *Finite Difference Method for the Solution of Laplace Equation*. Department of Aerospace Engineering, Iowa State University.

[MS69] Minsky, Marvin und Papert Seymour: *Perceptrons*. MIT press, 1969.

[MS11] Minh, Ha Q und Vikas Sindhwani: *Vector-valued manifold regularization.* In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, Seiten 57–64, 2011.

[MVLH08] Maier, M., U. Von Luxburg und M. Hein: *Influence of graph construction on graph-based clustering measures.* Proc. of Neural Infor. Proc. Sys.(NIPS 2008), 2008.

[Pap77] Papadimitriou, Christos H: *The Euclidean travelling salesman problem is NP-complete.* Theoretical Computer Science, 4(3):237–244, 1977.

[PBMW99] Page, Lawrence, Sergey Brin, Rajeev Motwani und Terry Winograd: *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66, Stanford InfoLab, November 1999. `http://ilpubs.stanford.edu:8090/422/`, Previous number = SIDL-WP-1999-0120.

[PS91] Park, Jooyoung und Irwin W Sandberg: *Universal approximation using radial-basis-function networks.* Neural computation, 3(2):246–257, 1991.

[Ros58] Rosenblatt, Frank: *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review, 65(6):386, 1958.

[SB98] Sutton, Richard S und Andrew G Barto: *Reinforcement learning: An introduction*, Band 1. Cambridge Univ Press, 1998. `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html`.

[Sch06] Schölkopf: *Learning with kernels.* Presentation, 2006. `http://dip.sun.ac.za/~hanno/tw796/lesings/mlss06au_scholkopf_lk.pdf`.

[Sha09] Shashua, Amnon: *Introduction to Machine Learning: Class Notes 67577.* CoRR, abs/0904.3664, 2009.

[SHS01] Schölkopf, Bernhard, Ralf Herbrich und Alex J Smola: *A generalized representer theorem.* In: *Computational learning theory*, Seiten 416–426. Springer, 2001.

[Sni06] Sniedovich, M: *Dijkstra's algorithm revisited: the dynamic programming connexion.* Control and Cybernetics, 35(3):599–620, 2006.

[Sou10] Souza, César R.: *Kernel functions for machine learning applications.* Web, Mar 2010. `http://crsouza.blogspot.de/2010/03/kernel-functions-for-machine-learning.html`, last checked 2013-09-26.

[Tou80] Toussaint, Godfried T.: *The relative neighbourhood graph of a finite planar set.* Pattern Recognition, 12(4):261 – 268, 1980, ISSN 0031-3203. `http://www.sciencedirect.com/science/article/pii/0031320380900667`.

[VL07] Von Luxburg, Ulrike: *A tutorial on spectral clustering.* Statistics and computing, 17(4):395–416, 2007.

[Wei] Weisstein, Eric W.: *Sigmoid Function.* `http://mathworld.wolfram.com/SigmoidFunction.html`, From MathWorld–A Wolfram Web Resource. last checked: 2013.09.30.

[WMKG07] Wardetzky, Max, Saurabh Mathur, Felix Kälberer und Eitan Grinspun: *Discrete Laplace operators: no free lunch.* In: *Symposium on Geometry processing*, Seiten 33–37, 2007.

[WW93] Wagner, Dorothea und Frank Wagner: *Between Min Cut and Graph Bisection.* In: Borzyszkowski, AndrzejM. und Stefan Sokołowski (Herausgeber): *Mathematical Foundations of Computer Science 1993*, Band 711 der Reihe *Lecture Notes in Computer Science*, Seiten 744–750. Springer Berlin Heidelberg, 1993, ISBN 978-3-540-57182-7. `http://dx.doi.org/10.1007/3-540-57182-5_65`.

[ZKLG06] Zhu, Xiaojin, Jaz Kandola, John Lafferty und Zoubin Ghahramani: *Graph kernels by spectral transforms.* In: Chapelle, Olivier, Bernhard Schölkopf, Alexander Zien *et al.* (Herausgeber): *Semi-supervised learning*, Band 2. MIT press Cambridge, MA:, 2006.