- Größere Themen
  - Einleitung TN
  - Schlusswort
  - Logo
  - Reihenfolge (alle Sections entweder mit keiner Subsection oder min. 3 subsections)
  - Alle Sätze noch einmal durchlesen und holprige Sätze verbessern.
- Kleinere Themen
  - Alle Einheiten nicht kursiv
  - Alle Formeln durchschauen, ob sie korrekt formatiert sind.
  - Alle Absätze müssen mindestens zwei Sätze haben.
  - Alle Posterthemen: Autor*innennamen. Posterautor*in dahinter in Klammern.
  - Vereinheitlichen: Subsections vs Paragraphs (z.B. „Algorithus von Siddon“)
  - Alle Abbildungen aussagekräftige Bildunterschriften (Bild nur beschreiben, Interpretation im Text).
  - Checken, dass alle Abbildungen im Text referenziert sind und ggf referenzieren.
  - In Formeln kein `,` sondern `.` für Kommazahlen verwenden
  - Gleichungen müssen in Satz eingebettet sein und ggf am Ende einen Punkt haben.
  - Alle Gleichungen mit Nummern (z.B. mit `equation` oder `align`).
- KL
  - (Einleitung KL)

INHALTSVERZEICHNIS

DIE THEORIE DER INFORMATION

WIE AUS DATEN BILDER WERDEN

2.1 Cox's Theorem

Patricia Hackl, Mara Germann, (Lara Müller)

Wie wird die Wahrscheinlichkeit eines Ereignisses, dem ein anderes Ereignis zugrunde liegt, berechnet? Um diese Frage zu beantworten, bedarf es der Wahrscheinlichkeitsrechnung.

Bei Cox's Theorem wird die Wahrscheinlichkeitsrechnung aus der Logik hergeleitet. Im Grenzfall für absolute Sicherheit, für das Eintreten beziehungsweise Nichteintreten von Ereignissen, geht die Wahrscheinlichkeitsrechnung in wahr/falsch Aussagen und boolsche Logik über. Mit Cox's Theorem entsteht eine in sich konsistente (einheitliche) Theorie für die Wahrscheinlichkeitsrechnung, aus der sich herleiten lässt, dass man Wahrscheinlichkeiten als Grad der Plausibilität interpretieren kann. Es wird später deutlich werden, dass die Plausibilität der Wahrscheinlichkeit entspricht. Cox's Theorem beruht auf 3 Axiomen, die die Begründung der Bayes'schen Wahrscheinlichkeitsrechnung sind.

1. Der **Grad der Plausibilität** eines Ereignisses B unter der Bedingung, dass A wahr ist $\{B|A\}$, wird als **reelle Zahl** dargestellt. Für hohe Plausibilitäten werden hohe Zahlenwerte gewählt. Dies ermöglicht den **universellen** Vergleich voneinander unabhängiger Plausibilitäten.

2. Sinnvolle Ergebnisse werden unter qualitativem Miteinbezug des **Verstandes** und durch logische Schlussfolgerungen erzielt.

3. Es müssen **alle** verfügbaren Informationen miteinbezogen werden. An die Schlussfolgerungen wird die Anforderung der **Konsistenz** gestellt, sodass alle Sätze, die gleiches Wissen vermitteln, auf gleiche Plausibilitäten hinführen müssen.

Aus Cox's Theorem lassen sich unter anderem die Summen- und Produktregel, die essentiell für die Wahrscheinlichkeitsrechnung sind, aus einfachen Axiomen herleiten. Es verbindet die Logik mit der Wahrscheinlichkeitsrechnung. Im Folgenden sind A, B und C drei Ereignisse.

Die Produktregel beschreibt die Plausibilität w der Ereignisse B und C unter der Bedingung, dass A wahr ist $w(BC|A)$. Dies entspricht der Plausibilität des Eintretens von B unter der Bedingung, dass A bereits eingetreten ist $w(B|A)$, multipliziert mit der Plausibilität des Eintretens von C unter der Bedingung, dass AB wahr ist $w(C|AB)$. Als Formel ausgedrückt:

$$w(\{BC|A\}) = w(\{B|A\}) \cdot w(\{C|AB\}). \tag{2.1}$$

Die Produktregel kann auf bedingte Wahrscheinlichkeiten übertragen werden:

$$P(B \wedge C|A) = P(B|A) \cdot P(C|AB).$$

Dabei versteht man unter $P(B \wedge C|A)$ die Wahrscheinlichkeit für B und C unter der Bedingung A. Aus der Produktregel folgt, dass 1 für die Wahrheit eines Ereignisses und 0 für die Unmöglichkeit eines Ereignisses steht. Dies dient als Grundlage für die Summenregel:

$$w_{ges} = w(\{B|A\}) + w(\{\bar{B}|A\}) = 1. \tag{2.2}$$

Die Gesamtplausibilität unter der Bedingung A ist die Plausibilität von B unter der Bedingung A und die Plausibilität des Gegenereignisses von B, ebenfalls unter der Bedingung A. Die Summe der beiden Wahrscheinlichkeiten ist 1.

2.2 Wahrscheinlichkeitstheorie

Patricia Hackl, Mara Germann, (Natalie Teplitska)

Die Bayes'sche Statistik beruht auf dem Rechnen mit bedingten Wahrscheinlichkeiten. Darunter versteht man die Wahrscheinlichkeit, dass ein bestimmtes Ereignis B eintritt, unter der Bedingung, dass ein Ereignis A bereits eingetreten ist. Diese Gleichung wird Satz von Bayes genannt:

$$P(B|A) = \frac{P(A \wedge B)}{P(A)} = \frac{P(B|A) \cdot P(A)}{P(B)}. \tag{2.3}$$

Bei unendlich vielen Ereignissen kann nicht jedem Ereignis eine bestimmte Wahrscheinlichkeit zugeordnet werden. Daher gibt man die Wahrscheinlichkeitsdichte an. Dabei beschreibt x alle möglichen Ereignisse. Weil alle Wahrscheinlichkeiten in Summe 1 ergeben müssen, gilt für die Fläche unter der gesamten Funktion:

$$\int_{-\infty}^{+\infty} P(x)\,\mathrm{d}x = 1. \tag{2.4}$$

Den Satz von Bayes wird verwendet, um aus Daten d, die aus einem Experiment gewonnen wurden, das Signal s zu berechnen. Wenn d eine verrauschte Tonaufnahme wäre, dann wäre s das Tonsignal ohne Rauschen. Die Hintergrundinformation I definiert das Modell.

$P(s|d, I)$ gibt die Wahrscheinlichkeitsverteilung des Parameters s an und wird Posterior genannt. Nach dem Satz von Bayes gilt:

$$P(s|d, I) = \frac{P(d|s, I) \cdot P(s|I)}{P(d|I)}. \tag{2.5}$$

- $P(s|I)$ gibt die Wahrscheinlichkeitsverteilung des Parameters vor dem Einbeziehen der Daten an (Prior).
- $P(d|I)$ ist der Normierungsparameter (Evidenz).
- $P(d|s, I)$ beschreibt die Wahrscheinlichkeit für die Messdaten mit gegebenem Parameter (Likelihood).

Das Experiment kann mehrfach wiederholt werden, dabei werden neue Daten gewonnen. So dient der Posterior bei erneuter Durchführung als Prior. Wir lernen sukzessive von den neuen Daten d und aktualisieren unser Wissen über Signal s.

2.3 Normalverteilung und Kovarianz

Patricia Hackl, Lara Müller

Im Hinblick auf die Bayes'sche Wahrscheinlichkeitstheorie ist die Gauß'sche Normalverteilung (auch: Gauß'sche Glockenkurve) von Bedeutung. Es lässt sich zeigen, dass die Summe von identisch verteilten und voneinander unabhängigen Zufallszahlen im Limes (Anzahl Wiederholungen $\rightarrow \infty$) zur Normalverteilung konvergiert. Die einem Zufallsexperiment zugehörige Gauß'sche Normalverteilung ist vollständig definiert über ihren Erwartungswert $\bar{x}$ und die Kovarianz Σ:

$$\mathcal{G}(x - \bar{x}, \Sigma) = \frac{1}{\sqrt{2\pi\Sigma}} \exp\left(-\frac{1}{2} \cdot (x - \bar{x}) \cdot \Sigma^{-1} \cdot (x - \bar{x})\right).$$

Analog zur Summenregel gilt auch für das Integral über die Normalverteilung:

$$\int_{-\infty}^{\infty} \mathcal{G}(x - \bar{x}, \Sigma) dx = 1.$$

2.3.1 Erwartungswert

Der Erwartungswert selbst ist definiert als die gewichtete Summe aus den Wahrscheinlichkeiten für die Zufallsvariable x_i und bestimmt im spezifischen Fall der Normalverteilung, an welcher Stelle das Maximum auftritt (Verschiebung der Glockenkurve entlang der x-Achse). Dieser Wert muss nicht zwingend realisierbar sein. In der Formel entspricht χ dem Raum, auf dem $P(x)$ definiert ist:

$$\mathbb{E}_{P(x)}[x] = \int_{\chi} xP(x)dx.$$

2.3.2 Varianz und Standardabweichung

Genauer betrachtet stellt die Kovarianz Σ die Streuung um den Erwartungswert dar und entspricht somit der Abweichung vom Mittelwert. Diese ist immer positiv:

$$\text{Cov}[x,x] = \mathbb{E}_{P(x)}[x\,x^\dagger] - \mathbb{E}_{P(x)}[x]\cdot(\mathbb{E}_{P(x)}[x])^\dagger = \int_\chi (x-\mathbb{E}_{P(x)}[x])(x-\mathbb{E}_{P(x)}[x])^\dagger P(x)dx \geq 0.$$

Die Varianz ist als Diagonale der Kovarianzmatrix definiert. Die Standardabweichung σ beschreibt das Maß der Streubreite und stellt die Wurzel der Varianz dar:

$$\sigma_{P(x)}[x] = \sqrt{\text{Var}_{P(x)}[x]}.$$

Mithilfe der Gauß'schen Glockenkurve lassen sich nun vielfältige Zufallsexperimente beschreiben und entsprechende Wahrscheinlichkeitsverteilungen konstruieren. In Bezug auf den Wiener Filter wird die Normalverteilung nicht nur als Prior, sondern auch als Likelihood benötigt, um Datenlücken zu füllen, die aufgrund des Abstands der Antennen des Radioteleskops auftreten. Zudem ist das Rauschen, also die Ungenauigkeit in den Messdaten, normalverteilt und kann so aus den Daten extrahiert werden.

2.4 Komplexe Zahlen

Natalie Teplitska, Ole Fleck, (Mara Germann)

Da in der Radioastronomie die Messdaten komplexe Zahlen sind, hat sich unser Kurs auch mit diesen beschäftigt.

Die Menge $\mathbb{C}$ ist die Menge aller Zahlen $z = a + b \cdot i$ für $a, b \in \mathbb{R}$. Dabei wird a als Realteil und b als Imaginärteil bezeichnet. Weiterhin ist i die imaginäre Einheit und definiert über die Eigenschaft: $i^2 = -1$. Für $b = 0$ erhalten wir die bereits bekannten reellen Zahlen. Man kann $\mathbb{C}$ somit als zweidimensionalen reellen Vektorraum interpretieren.

Die komplex konjugierte von $z = a + b \cdot i$ ist definiert als $\bar{z} = a - b \cdot i$. Das Produkt aus z und $\bar{z}$ hat die Eigenschaft, dass $z \cdot \bar{z} = (a + b \cdot i) \cdot (a - b \cdot i) = a^2 - b \cdot i^2 = a^2 + b^2$ eine reelle Zahl ist.

Zwei komplexe Zahlen werden addiert, indem ihre Real- und Imaginärteile separat addiert bzw. subtrahiert werden. Auch die Multiplikation funktioniert ähnlich wie in $\mathbb{R}$, es sollte jedoch stets bedacht werden, dass $i \cdot i = -1$.Die Division von komplexen Zahlen gestaltet sich etwas komplizierter. Hier hilft es oft, mit der konjugierten komplexen Zahl des Nenners zu erweitern, sodass im Nenner kein Imaginärteil mehr vorkommt und der Ausdruck sich so erheblich vereinfacht.

Die Menge $\mathbb{C}$ lässt sich geometrisch in ein einem kartesischen Koordinatensystem darstellen, wobei die x-Achse für a und die y-Achse für b steht. Jede komplexe Zahl $z = a + b \cdot i$ entspricht dann genau einem Vektor $\begin{pmatrix} a \\ b \end{pmatrix}$. Bisher haben wir mit den komplexen Zahlen in kartesischen Koordinaten gerechnet. Sie lassen sich aber auch als Polarform angeben:

$$z = r \cdot e^{i \cdot \phi}.$$

Dabei ist ϕ der Winkel des Vektors zur x-Achse und r seine Länge. Bei der Multiplikation solcher Vektoren werden Winkel addiert und Längen multipliziert. Eine konjugierte komplexe Zahl ist dadurch äquivalent zur Spiegelung des Vektors an der reellen Achse.

2.5 Lineare Abbildungen zwischen Vektorräumen

Lara Müller, Chuyang Wang, (Patricia Hackl)

Der folgende Abschnitt befasst sich mit den Grundlagen linearer algebraischer Berechungen und definiert in diesem Kontext lineare Abbildungen und zugehörige Vektorräume.

2.5.1 Grundlagen linearer Algebra

Ein Vektorraum über dem Körper K einer Zahlenmenge ist definiert als Menge von Vektoren V, die einen entsprechenden Vektorraum beschreiben. Für die Vektoren a, b und c eines linearen Vektorraums müssen folgende grundlegende Rechenregeln gelten:

- Kommutativgesetz: $a + b = b + a$
- Assoziativgesetz: $a + b + c = (a + b) + c = a + (b + c)$
- Distributivgesetz: $a \cdot (b + c) = a \cdot b + a \cdot c$

Der Gegenvektor eines Vektors a ist definiert als $z = -a$, sodass gilt $a + z = 0$. Eine skalare Vervielfachung bechreibt allgemein die Multiplikation eines Vektors a mit einer reellen Zahl $\lambda \in \mathbb{R}$, wodurch die Vektorlänge variiert wird. Für $\lambda < 0$ wird zudem die Richtung des Vektors umgekehrt. Im Beispiel eines dreidimensionalen Vektors a ergibt sich dementsprechend:

$$a \cdot \lambda = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \lambda = \begin{pmatrix} a_3\lambda \\ a_2\lambda \\ a_1\lambda \end{pmatrix}$$

2.5.2 Lineare Abbildungen

Als lineare Abbildung wird jede Abbildung definiert, deren Additivität ($f(v+w) = f(v)+f(w)$ für alle v, $w \in V$) und Homogenität ($f(\lambda v) = \lambda f(v)$ für alle $\lambda \in K$, $v \in V$) gegeben sind. Bezüglich der zwei K-Vektorräume V und W ist die Abbildung $f : V \to W$ linear, sofern gilt:

$$f(\lambda v + \mu w) = \lambda f(v) + \mu f(w) \quad \forall\, \lambda, \mu \in K \quad \forall\, v, w \in V.$$

2.5.3 Eigenschaften von Vektorräumen

Darüber hinaus bezeichnet man die Menge der Vektoren, die auf den Nullvektor $\vec{o}$ abgebildet werden, als Kern der zugehörigen linearen Abbildung. Der Kern $\ker(f) := \{v \in V \mid f(v) = 0\}$ entspricht demnach einem Vektor aus dem Vektorraum V ($\ker(f) \subseteq V$), in dem auch der Nullvektor enthalten ist ($0 \in \ker(f)$). Vektorräume können unendlichdimensonal sein und somit beispielsweise als reeller ($V \in \mathbb{R}^n$) oder komplexer ($V \in \mathbb{C}^n$) Vektorraum auftreten. Die jeweilige

Dimension eines endlichdimensionalen Vektorraums K^n kann über die zur Darstellung zwingend notwendige Mindestanzahl n an Basisvektoren bestimmt werden. Diese sind so definiert, dass sich jeder andere Vektor c über Linearkombinationen der Basisvektoren b_i beschreiben lässt:

$$c = \sum_{i}^{n} \alpha_i b_i.$$

2.5.4 Darstellung linearer Abbildungen durch Matrizen

Lineare Abbildungen lassen sich auch über Matrizen darstellen. Beispielsweise gilt für eine lineare Abbildung in dem zweidimensionalen Vektorraum K^n, der über die Basisvektoren $\vec{u}$ und $\vec{v}$ dargestellt werden kann:

$$x \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} + y \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} u_1 x + v_1 y \\ u_2 x + v_2 y \end{pmatrix} = \begin{pmatrix} u_1 & v_1 \\ u_2 & v_2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Die resultierende 2x2-Matrix beinhaltet entsprechend der Matrix-Spalten sowohl das Bild des ersten, als auch das des zweiten Basisvektors. Allgemein lassen sich lineare Abbildungen $f(\vec{x})$ über eine zugehörige Matrix A darstellen:

$$f(\vec{x}) = A \cdot \vec{x}.$$

Bei einer linearen Abbildung stellt $f(\vec{x})$ selbst auch stets wieder einen Vektor dar. Dabei bleiben Streckenverhältnisse und Lagebeziehungen der Vektoren zueinander unverändert. Im Gegensatz zur Drehung, Scherung und Projektion sind Krümmungen bei einer linearen Abbildung nicht möglich. Daher werden diese Abbildungen auch als strukturerhaltende Abbildungen zwischen Vektorräumen bezeichnet.

2.5.5 Vektor-Matrix-Multiplikation

Das Vektor-Matrix-Produkt wird für die Matrix $A \in \mathbb{R}^{N \times M}$ und den Vektor $x \in \mathbb{R}^M$ wie folgt definiert:

$$\begin{aligned} Ax &= \begin{pmatrix} a_{11} & a_{12} & ... & a_{1M} \\ a_{21} & a_{22} & ... & a_{2M} \\ ... & ... & ... & ... \\ a_{N1} & a_{N2} & ... & a_{NM} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_M \end{pmatrix} \\ &= \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + ... + a_{1M}x_M \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2M}x_M \\ ... + ... + ... + ... \\ a_{N1}x_1 + a_{N2}x_2 + ... + a_{NM}x_M \end{pmatrix} \end{aligned} \tag{2.6}$$

Dabei wird jede Zeile der Matrix mit dem Vektor elementweise multipliziert. Die Summe der Produkte bildet die Komponente des resultierenden Vektors.

2.6 Eigenwertproblem

Lara Müller, Chuyang Wang, (Benjamin Knöbel del Olmo)

2.6.1 Definitionen

Die Eigenvektoren einer Matrix sind diejenigen Vektoren, welche nach Anwendung dieser Matrix immer auf einem Vielfachen von sich selbst liegen. Formal werden die Eigenvektoren einer quadratischen Matrix $A \in \mathbb{R}^{x \times x}$ definiert als $v \in \mathbb{R}^x, v \neq 0$, sodass $A \cdot v = \lambda \cdot v$ erfüllt ist. Man nennt das Skalar λ den zu v zugehörigen Eigenwert.

Durch Äquivalenzumformung erhält man

$$\begin{aligned} & A \cdot v = \lambda \cdot v \\ \Leftrightarrow \quad & A \cdot v = \lambda E \cdot v \qquad \text{mit } v = Ev \\ \Leftrightarrow \quad & (A - \lambda E) \cdot v = 0 \\ \Leftrightarrow \quad & Bx = 0 \qquad \text{mit } B = (A - \lambda E) \end{aligned} \tag{2.7}$$

Die *Determinante* einer Matrix ist ein skalarer Wert, welcher die Lösbarkeit dieser Matrix beschreibt. Dieses lineare Gleichungssystem aus Gleichung (2.7) hat genau dann nicht-triviale Lösungen ($x \neq 0$), wenn die Determinante von B gleich 0 ist. Also gilt $\det(B) = \det(A - \lambda E) = 0$.

Bei dem Eigenwertproblem gilt es, diese Vektoren sowie die zugehörigen Eigenwerte zu finden.

2.6.2 Das charakteristische Polynom

Im Allgemeinen wird das charakteristische Polynom der Matrix A definiert als $\chi_A(\lambda) = \det(A - \lambda E)$. Aus Abschnitt 2.6.1 folgt, dass man die Nullstellen dieses Polynoms finden muss, um den Eigenwert zu berechnen.

Betrachtet man nun beispielsweise das Problem in 2D und sei $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, so gilt

$$\begin{aligned} 0 &= \chi_A(\lambda) \\ &= \det(A - \lambda E) \\ &= \det \begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix} \\ &= (a - \lambda) \cdot (d - \lambda) - c \cdot b \end{aligned} \tag{2.8}$$

Eine allgemeine Lösung für Gleichung (2.8) kann dann mithilfe der pq-Formel berechnet werden:

$$\lambda = \frac{a + d}{2} \pm \sqrt{\left(\frac{(a + d)}{2}\right)^2 - ad + cb} \tag{2.9}$$

2.6.3 Power Iteration

Für 2×2 Matrizen lässt sich das Eigenwertproblem relativ gut lösen, da eine allgemeine Formel (vgl. pq-Formel / abc-Formel) für das charakteristische Polynom existiert. Ab dem 5. Grad ist es jedoch unmöglich, eine allgemeine Formel herzuleiten (Ramond 2022). Mit dem Power-Iteration-Algorithmus versucht man, eine Annäherung an den höchsten Eigenwert zu berechnen. Der iterative Algorithmus wählt am Anfang einen willkürlichen Wert für $b_0 \in \mathbb{R}^n$. Bei jeder Iteration aktualisiert man diesen Vektor b wie folgt:

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|} \tag{2.10}$$

Nach ausreichenden Iterationen kann man den größten Eigenwert λ berechnen, indem man die Gleichung $Bb_k = \lambda b_k$ nach λ auflöst.

2.6.4 Anwendungen

Eigenwerte und Eigenvektoren sind wichtig in der Mathematik. Ein Beispiel dafür ist die Drehung durch eine Matrix: Wenn die Matrix A eine Drehung um einen bestimmten Winkel beschreibt, dann ist der Eigenvektor die Drehachse, da seine Richtung durch die Drehung nicht verändert wird.

2.7 Conjugate Gradient

Benjamin Knöbel del Olmo, Leo Bergmann, (Chuyang Wang, Cedric Balzer)

Um lineare Gleichungssysteme (LGS) effizient und exakt zu lösen, existiert ein Algorithmus namens Conjugate Gradient, welcher auch auf der Minimabestimmung basiert.

Ein LGS hat die Formel $Ax = b$, wobei A eine $n \times n$-Matrix (invariant unter Transposition) und b, x Vektoren sind. Eine quadratische Funktion $f(x) = 1/2xAx - bx + c$ wird definiert, (mit selben A,b und x), dessen komponentenweise Ableitung genau dann null ist, wenn das ursprüngliche LGS nach x gelöst ist, da gilt: $f'(x) = Ax - b = 0$. Daher ist das Optimieren der dem LGS entsprechenden Funktion äquivalent zum Lösen ebenjenes.

Der Algorithmus funktioniert folgendermaßen: Conjugate Gradient läuft das Residuum (der dem Gradienten entgegengesetzte Vektor, $r = b - Ax$) ab, aber korrigiert die „Abstiegsrichtung“ in Betrachtung aller vorher gegangenen Residuen so, dass diese A-orthogonal (engl. conjugate, def.: $x^T Ay = 0, x, y$ A-orthogonal) zu allen vorherigen ist. Daraus folgt, dass wir nach maximal n-Schritten am Ziel ankommen, da wir dann alle n Dimensionen der $n \times n$-Matrix optimal abgegangen sind. Der Algorithmus hat also zwei Abbruchbedingungen, zum einen, wenn n Iterationen erreicht sind, oder wenn die Differenz zum Endergebnis, also dass $f''(x) = 0$ gilt, klein genug ist.

Gradient Descent und Conjugate Gradient ermöglicht es uns also, lineare Gleichungssysteme zu lösen und dadurch Daten, die in Matrizen gespeichert sind, weiterzuverarbeiten. Ein Beispiel wäre die Lösung der Gleichung $D^{-1}m = j$, auf die im Kapitel „Wiener Filter“ eingegangen wird.

2.8 Programmierparadigmen

Alexander Gitnik, Clemens Ljungh, (Jonas Fiedler, Aaron Gschwendt)

2.8.1 Objektorientierte Programmierung

Die objektorientierte Programmierung ist eins der am weitesten verbreiteten Programmierparadigmen. Beim objektorientierten Programmieren ist das ausschlaggebende Kriterium, dass man den Code zu sinnvollen Strukturen zusammenfasst. Dafür erstellt man Klassen, welche oftmals reale Objekte repräsentieren.

Eine Klasse ist eine allgemeine Vorlage, aus der einzelne Objekte, die *Klasseninstanzen*, initialisiert werden können. Diese Klassen besitzen zum einen Eigenschaften, *Attribute* genannt, und zum anderen Handlungsmöglichkeiten, welche *Methoden* genannt werden. Jede Klasse besitzt einen Konstruktor, welcher bei der Erstellung einer *Instanz* der Klasse aufgerufen wird. Zudem werden gegebenenfalls die zum Erstellen einer Klasse benötigten Parameter übergeben. Objekte werden durch Zuweisung des Konstruktoraufrufs, der jeweiligen Klasse, an eine Variable initialisiert, welche nun zu einem Objekt des Klassentyps wird. Wenn sich Klassen ähneln, können gleiche *Attribute* und *Methoden* in einer *Superklasse* definiert werden. Diese vererben den Klasseninhalt an die *Unterklassen*, welche die Merkmale übernehmen.

Typischerweise werden *Attribute* lediglich über *Getter-* und *Setter-Methoden* abgefragt und verändert, damit der Zustand des Objekts immer nur kontrolliert verändert werden kann. Dies hat den Grund, dass man nicht versehentlich einen Fehler in den inneren Zustand des Objekts einbauen möchte. Eine *Setter-Methode*, kann überprüfen, welche Werte ein Attribut annimmt. Um eine Methode aufrufen zu können ist es im Normalfall nötig dies über eine bereits initialisierte Instanz der Klasse zu tun. Ein spezieller Dekorator macht die Methode statisch, was bedeutet, dass die Methode über die Klasse aufgerufen werden kann. Dies hat den Vorteil, dass man nicht erst ein Objekt initialisieren muss un die Methode nutzen zu können.

2.8.2 Funktionale Programmierung

Die funktionale Programmierung ist ein gänzlich anderes Konzept als die objektorientierte Programmierung. Funktionale Programmierung hat ihren Ursprung im Lambda-Kalkül. Das Lambda-Kalkül ist eine formale Sprache, um logische Systeme darzustellen. Es ist Turing-vollständig, dies bedeutet, dass man jedes Programm damit ausdrücken kann.

Anders als bei der objektorientierten Programmierung werden Daten bei der funktionalen Programmierung nicht in Objekten gespeichert, sondern ausschließlich durch einzelne Funktionen verarbeitet. Diese Funktionen müssen dabei *pur* sein. *Pure Funktionen* definieren sich dadurch, dass sie bei gleicher Eingabe immer die gleiche Ausgabe geben, da sie keinen inneren Zustand haben, welcher die Eingabe verändern könnte. Weiterhin ändern die puren Funktionen auch nichts außerhalb der eigenen Funktion.

Funktionen können als Parameter den Rückgabewert einer anderen Funktion nehmen, sodass jene ineinander verschachtelt werden. So entsteht in der funktionalen Programmierung natürlicherweise eine Kette aus verschachtelten Befehlen. Häufig wird in der funktionalen Programmierung *Rekursion* verwendet, also Funktionen, die sich selbst aufrufen.

Daten sind bei der objektorientierten Programmierung zwar besser strukturiert, aber dafür sind gerade die puren Funktionen der funktionalen Programmierung einfacher für Entwickler

zu verstehen, da die puren Funktionen keinen möglicherweise komplizierten inneren Zustand haben. Wichtig für die funktionale Programmierung ist das Konzept der *Lazy Evaluations*. Dies bedeutet, dass Ausdrücke, erst dann ausgerechnet werden, wenn sie für ein Ergebnis benötigt werden. Dadurch kann in manchen Fällen die Laufzeit eines Programmes verringert werden, was vorteilhaft ist.

Für unsere Kursarbeit verwenden wir oft funktionale Programmierung, da wir Datensätze und probabilistische Modelle rein mathematisch verarbeiten. Außerdem ist Python gut für die funktionale Programmierung geeignet.

2.8.3 Gegenüberstellung

Insgesamt unterscheidet sich damit die objektorientierte Programmierung von der funktionalen Programmierung bereits in ihrem Ansatz. Sie ist realtitätsbezogen, wohingegen die funktionalen Programmierung lediglich an der Programmeffektivität orientiert ist. Die Umsetzung ist daher sehr verschieden, sodass in der objektorientierten Programmierung Klassen und globale Variablen gefordert werden, während die funktionale Programmierung ebendiese verbietet, und mittels puren Funktionen den Code in kleinstmögliche und unabhängige Abschnitte unterteilt.

2.9 NumPy

Cedric Balzer, (Leo Bergmann)

Numerisches Python (kurz NumPy) ist eine in der Programmiersprache C geschriebene Libary für Python. Sie ermöglicht das effiziente Rechnen mit Matrizen, mehrdimensionalen Arrays und Vektoren in Python.

NumPy ist eine open-source Library, welche auf den früheren Python-Modulen Numeric und Numarray basiert. Da Python nicht für numerische Rechnung optimiert ist, greift NumPy auf C-Code zurück. Eine der Kernfunktionalitäten von NumPy ist das sogenannte `np.ndarray()`, das ein beliebig-dimensionales Array repräsentiert. Im Gegensatz zu Python-Listen müssen alle im Array gespeicherten Elemente vom selben Datentyp sein, zudem ist die Größe eines NumPy-Arrays statisch. Dies ermöglicht im Vergleich zu Listen eine schnellere Handhabung.

NumPy ist nicht im Python3-Standard enthalten und muss daher separat installiert werden:

```
pip install numpy
```

Möchte man ein Array aus einer Liste erstellen, bietet sich folgender Code an:

```
>>> import numpy as np
>>> a = np.array([1, 2, 3, 4, 5])
>>> a
array([1, 2, 3, 4, 5])
```

Besteht eine übergebene Liste aus mehreren Teillisten wird ein mehrdimensionales Array erstellt:

```
>>> m1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> m1
array([[1,2,3],
       [4,5,6],
       [7,8,9]])
```

Möchte man auf ein Element eines eindimensionalen Arrays zugreifen, so funktioniert dies wie bei Listen. Um in mehrdimensionalen NumPy-Arrays Werte zu selektieren, wird folgende Syntax verwendet:

```
>>> m1[0, 0], m1[2, -1]
1, 9
```

Auf Arrays können elementweise Rechenoperatoren, wie Plus und Minus, angewandt werden. Es werden stets neue Arrays erzeugt und die Original-Arrays bleiben unverändert.

```
>>> r = np.arange(10)
>>> r
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> r+1
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> r**2
array([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
```

Des Weiteren sind in NumPy spezielle Operationen für Matrizen und Vektoren implementiert. Beispielsweise können mit `np.dot()`, `np.inner()` und `np.outer()` Multiplikationen von Matrizen beziehungsweise Vektoren durchgeführt werden (siehe **??** für eine mathematische Beschreibung).

```
>>> np.dot([[1, 0], [0, 1]], [[4, 1], [2, 2]])
array([[4, 1],
       [2, 2]])
```

Neben `np.dot()` liefert NumPy den Operator `@` (und alternativ `np.matmul()`), welcher ebenfalls Arrays und Matrizen multipliziert. Bei einer Multiplikation von Vektoren und 2D Matrizen erzeugen diese Funktionen identischen Output:

- Skalarmultiplikation zweier Vektoren

  ```
  >>> a = np.array([1, 2])
  >>> b = np.array([2, 2])
  >>> np.dot(a, b)
  6
  >>> a @ b
  6
  ```

- Multiplikation zweier Matrizen

```
>>> a = np.array([[1, 2],[5, 3]])
>>> b = np.array([[2, 2],[8, 1]])
>>> np.dot(a, b)
array([[18  4]
       [34 13]])
>>> a @ b
array([[18  4]
       [34 13]])
```

NumPy ist also die Standardbibliothek für effiziente Matrix-Vektor-Rechnung in Python. Sie liefert die Grundlagen für viele wissenschaftliche Arbeiten, so auch für unser Projekt, um Bayes'sche Bildgebungsalgorithmen zu implementieren.

2.10 Pytest

Finja Hoffmann, (Alexander Gitnik)

Pytest ist ein Testing-Framework für Python, mit welchem sich die Fehlerbehebung und allgemeine Testphase in der Softwareentwicklung automatisieren lässt. Anstatt sich immer mal wieder an ausgewählten Stellen Werte ausgeben zu lassen und diese manuell zu überprüfen, kann man Test-Methoden schreiben, um so automatisiert Fehler zu erkennen. Pytest überprüft mit synthetischen Eingaben die zu testende Funktion und vergleicht die Ausgabe mit dem eingegebenen Erwartungswert.

Das Überprüfen implementiert man durch:

```
assert f(A) == B
```

Hierbei wird die Funktion `f` mit Eingabe `A` und Erwartungswert `B` getestet.

In einem konkreten Beispiel addiert eine Funktion den eingegebenen Wert mit 2. Hier wird 1 eingegeben und der Erwartungswert ist 3:

```
assert add_2(1) == 3
```

Pytest führt nun die zu testende Methode mit den eingegebenen Werten aus. Durch das `assert`-statement wird automatisch die Ausgabe überprüft. Sollte die erwartete Ausgabe nicht mit der tatsächlichen Ausgabe übereinstimmen wird eine Fehlermeldung angezeigt. Es wird zudem angezeigt, welche Eingabe und Code-Zeile zur Fehlermeldung geführt haben.

Die zu testende Methode beginnt mit `test_`.

Mit dem Dekorator `@pytest.fixture` vor einer Funktion lässt sich die Eingabe von zu testenden Methoden global definieren und dessen Spezifikationen automatisieren.

Auch kann man Parameter eingeben, bei denen man eine Fehlermeldung erwartet. Es wird überprüft, ob nicht erwartete Eingaben wirklich als Fehler erkannt werden. Dafür gibt es den Befehl

```
with pytest.raises({FehlerTyp})
```

mit dem überprüft wird, ob die Funktion wirklich die erwartete Fehlermeldung zurückgibt. Weiterhin kann man Funktionen mit einer Test-Methode durch den Dekorator

```
@pytest.mark.parametrize({Eingabe})
```

kennzeichnen. In diesem Fall werden die Ein- und Ausgabewerte paarweise in Klammern untereinandergeschrieben.

Wenn die automatisierten Test-Methoden ohne Fehlermeldung durchlaufen, so ist der Algorithmus scheinbar fehlerfrei. Dabei können Fehler aber nicht vollständig ausgeschlossen werden, da die Testung immer nur so gut ist wie die Ideen-Vielfalt zu potenziellen Fehlerquellen. Nicht bekannte, ungetestete Fehler können nicht vermieden werden.

2.11 Git

Jonas Fiedler, (Clemens Ljungh)

Mit der Versionsmanagement-Software Git kann Kooperation zwischen mehreren Programmierern ermöglicht werden. Git beschäftigt sich hauptsächlich mit dem Verwalten von Versionen des Codes und der Zusammenführung von Code Änderungen. Dafür erstellt Git einen Versionsverlauf, welcher es ermöglicht, Versionen des Codes zurückrollen um Bugs zu identifizieren und das Nachverfolgen von Code-Änderungen von wichtigen Updates zu ermöglichen. Dabei wird zwischen lokaler Dateispeicherung und Speicherung im serverseitigen Bereich unterschieden. Der Server ermöglicht den Nutzern, Code herunterzuladen, ihn zu bearbeiten und letztendlich in überarbeiteter Version wieder an den Server zu übermitteln, wo dieser dann mit anderen Nutzern ausgetauscht werden kann.

Generell ist es in Git möglich verschiedene Versionsstränge (Branches) aufzubauen. Diese Branches ermöglichen die Modifikation an verschiedenen Bereichen und ein strukturierte Entwicklung von Features. In diesen Branches können somit bspw. Bugs unabhängig gelöst werden und neue Funktionen getestet sowie implementiert werden. Sobald diese ordnungsgemäß funktionieren, werden sie auf den nächst höheren Branch, eine in der Hierarchie übergeordnete Ebene, geschoben (Merge) und somit mit dem Rest des Codes wieder vereint.

In Git gibt es verschiedene Stadien, der „Codeverarbeitung“ um eine Codeversion zu speichern. Dabei beschreibt `git add` den Vorgang wo die Datei von dem Working-Directory (lokal), in die Staging Area (auch lokal) verschoben wird. Der wichtigste Befehl in Git ist `git commit`, dieser hält den aktuellen Zustand des gesamten Projektes fest und speichert diesen im Localrepo. Danach ermöglicht `git push` das Übertragen des Codes an den Server. Der umgekehrte Schritt dazu ist `git pull`. Dies lädt die aktuellste Version des Projektes vom Remoterepo in das Localrepo des Users.

Aufgrund der diversen Verwaltungsoptionen und der Möglichkeit kollaborativ in Git zu arbeiten, wird dieses Tool von vielen Software-Developern, sowie Unternehmen verwendet.

2.12 Radioastronomie

Constantin Burmeister, (Ole Fleck)

Radioastronomie dient der Beobachtung von astrophysikalischen Prozessen, die Radiowellen, also Wellen langer Wellenlänge, aussenden. Um hohe Auflösungen zu erreichen, werden Radiointerferometer verwendet, welche Signale mehrerer Antennen so zusammenfügen, als würde es sich um eine Antenne handeln.

Allgemein ist der minimal auflösbare Bildwinkel $\delta\theta$ eines Teleskops abhängig von der Wellenlänge λ, des beobachteten Lichts, und dem Spiegeldurchmesser D des Teleskops mit

$$\delta\theta \approx 1.22\frac{\lambda}{D}. \tag{2.11}$$

Da die Radioastronomie große Wellenlängen verwendet, sind hier auch bei Teleskopen mit großen Spiegeldurchmessern von über 100m nur geringe Auflösungen möglich.

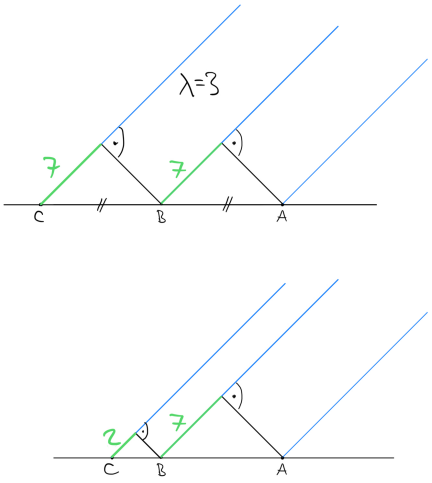

Abbildung 2.1: Gangdifferenzen von Radiowellen bei Antennenarrays (A,B und C) mit Wellenlänge $\lambda = 3$

Zur Kompensation dieses Problems wurde die Radiointerferometrie entwickelt. Das Verfahren dient zur Überlagerung von Radiowellen, welche von mehreren Antennen aufgezeichnet wurden, um eine höhere Auflösung zu erzielen als es mit einer einzelnen Antenne möglich wäre. Das hieraus resultierende Signal ist vergleichbar mit dem Signal, welches von einer Antenne mit dem Spiegeldurchmesser entsprechend des Antennenabstandes erzeugt worden wäre.

Ein Interferometer kann nur die Laufzeitdifferenz zwischen Wellen bestimmen, da die einzelnen Wellenberge nicht unterschieden werden können. So können, wie im Beispiel Abb. 2.1, bei gleichen Abständen zwischen den Teleskopen A,B und C nicht zwischen den Gangdifferenzen modulo der Wellenlänge, welche im Beispiel $\lambda = 3$ beträgt, unterschieden werden. In diesem Beispiel ist aus den Messwerten nicht ersichtlich, ob es sich tatsächlich um $2\cdot\lambda+1 = 7$ Gangunterschied handelt, sondern jedes $n \in \mathbb{N}$ in $n\cdot\lambda+1$ ist möglich. Die Differenz könnte also immer

auch um vielfache von 3 Längeneinheiten verschieden sein. Werden Antennen mit unterschiedlichen Abständen verwendet, sinkt die Anzahl der Kombinationen der Gangunterschiede, für welche sich eine solche Gleichung lösen lässt wie im Beispiel. Hier müssen die Ergebnisse der Gleichung für das untere Bild kongruent zu $3 \bmod 2$ sein. Daher werden bei Antennengruppen möglichst viele unterschiedliche Abstände verwendet.

Für die Messung des Radiosignals gilt:

$$d_{uv\lambda} = \int\int I(l,m) e^{-2\pi i \frac{1}{\lambda}(lu-mv) dl dm} \tag{2.12}$$

Hier bezeichnet l die Nord-Süd-Achse und m die Ost-West-Achse, λ bezeichnet die Beobachtungswellenlänge, und u und v ist der Verbindungsvektor zwischen zwei Antennen. Zusätzlich bezeichnet $n_{uv\lambda}$ ein additives Rauschen. Diese Messgleichung hat große Ähnlichkeit zur Fourier-Transformation (siehe Abschnitt 2.13) und kann als nicht-äquidistante Fourier-Transformation interpretiert werden.

Dieses Verfahren wurde beispielsweise auch beim *Event Horizon Telescope* verwendet. Das *Event Horizon Telescope* besteht aus einer Verschaltung von elf Teleskopen, wobei deren Abstand bis zu $10000\,\text{km}$ beträgt. Mit diesem Teleskop wurde das erste Mal der Schatten eines Schwarzen Lochs beobachtet.

Radiointerferometrie ist also ein essenzieller Bestandteil der Astronomie. Das Verfahren ermöglicht die Ursprungsbestimmung eines Radiosignals von astrophysikalischen Prozessen bei hoher Auflösung.

2.13 Fourier-Transformation

Natalie Teplitska, Ole Fleck, (Constantin Burmeister)

Bei der Fourier-Transformation handelt es sich um eine mathematische Methode zur Übersetzung eines Signals in ein Frequenzspektrum. Dank ihrer sehr allgemeinen und umfassenden Formulierung findet sie sowohl Anwendung in mathematischen Bereichen, aber auch in der Physik, wie beispielsweise in unserem Kurs bei der Verarbeitung von Radiosignalen.

Ein anschauliches Beispiel ist die Spektralanalyse von Schallwellen. Die Daten (in der Abb. 2.2 links), die man beispielsweise mit einem Mikrofon aufnimmt, bestehen aus Schallwellen unterschiedlicher Wellenlängen. Mithilfe der Fourier-Transformation wird das Signal auf Schwingungen mit verschiedenen Wellenlängen aufgeteilt (in der Abb. 2.2 rechts), wodurch man das Frequenzspektrum erhält.

Je nachdem, welche Art von Rohdaten man erhält und welche Ausgabe erwünscht ist, nimmt die Fourier-Transformation eine leicht andere Form an.

Die kontinuierliche Fourier-Transformation (FT) benötigt als Eingabe eine Funktion, über die mit

$$\mathcal{F}(x)(s) = \int_{-\infty}^{\infty} e^{-ist} x(t)\, dt$$

integriert wird. Diese Transformation lässt sich nur auf kontinuierliche Funktionen anwenden und gibt eine kontinuierliche Funktion aus.

Die diskrete Fourier-Transformation (DFT) kommt mit diskreten Werten aus und ist daher am

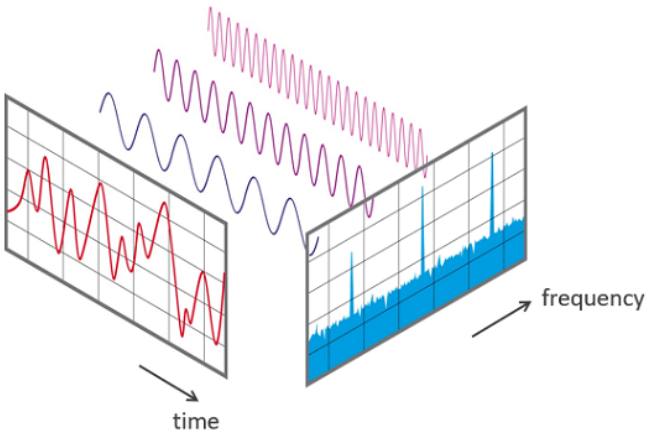

Abbildung 2.2: Anschauliche Darstellung der Fourier-Transformation (Quelle: www.nti-audio.com)

Besten für digitale Signalverarbeitung geeignet, da die Eingabesignale meistens aus einzelnen Datenpunkten und nicht aus kontinuierlichen Funktionen bestehen. Die Formel dazu ist entsprechend kein Integral, sondern eine Summe aller gegebenen Datenpunkte:

$$\hat{a}_k = \sum_{j=0}^{N-1} e^{-2\pi i \cdot \frac{jk}{N}} \cdot a_j.$$

Die DFT ist mit einer quadratischen Laufzeit allerdings vergleichsweise rechenintensiv. Stattdessen wird in der Signalverarbeitung meist die Fast Fourier Transformation (FFT) verwendet, welche mit Hilfe des *Divide and Conquer* (Teile-und-Herrsche) Prinzips in der Lage ist, die Rechenzeit drastisch zu reduzieren.

Manche Probleme, welche im Wertebereich, den eigentlichen Signalen, nur durch viele komplizierte Operationen lösbar sind, können im Bildbereich, dem durch die Fourier-Analyse berechneten Frequenzspektrum, mit wenigen einfachen Operationen gelöst werden. Dazu wendet man die Fourier-Analyse auf die Daten an, führt die notwendigen Operationen im Bildbereich aus und transformiert die Lösung zurück in den Wertebereich. Möchte man beispielsweise eine Tonaufnahme von Störsignalen bereinigen, wendet man die FFT an, löscht die unerwünschten Frequenzen im Bildbereich und übersetzt das Ergebnis zurück in Musik oder Sprache. Diese Aufgabe wäre im Wertebereich erheblich rechenintensiver gewesen, da an einer Tonaufnahme über der Zeit nicht direkt das Störsignal ablesbar ist.

Das „Rückübersetzen“ der Signale erfolgt mit der inversen (kontinuierlichen oder diskreten) Fourier-Transformation (IFT). Diese erzeugt die Funktion oder eine Menge von Datenpunkten, welche aus einem gegebenen Spektrum entsteht. Dazu werden dieselben Operationen wie bei der Berechnung des Spektrums lediglich mit anderen Faktoren angewandt.

2.14 Wiener Filter

Constantin Burmeister, Clemens Ljungh, Natalie Teplitska

Für unseren Kurs benötigen wir eine Möglichkeit, aus Rohdaten auf das zugrundeliegende physikalische Signal zurückzuschließen. Allerdings können Messungen nur mit einer endlichen Sicherheit erfolgen, was bildgebende Verfahren vor Probleme stellt. So lässt sich Rauschen sowie eine gewisse Ungenauigkeit in der Messung nicht vermeiden; zudem sind die Daten unvollständig. Um solche Probleme zu lösen, brauchen wir den Wiener Filter. Dieses mathematische Verfahren basiert auf dem Bayes'schen Theorem (2.2) , welches mathematisch beschreibt, wie Wissen optimal verändert werden muss, nachdem Daten erhoben wurden.

Der Wiener Filter kann nur verwendet werden, wenn die zwei folgenden Annahmen erfüllt sind. Zum Einen setzt man voraus, dass a priori das untersuchte Signal Gauß-verteilt ist: $P(s) = \mathcal{G}(s, S)$. Zum Anderen ist das Rauschen normalverteilt mit $P(n) = \mathcal{G}(n, N)$ und additiv. Aus der Normalverteilung des Rauschens sowie unter Einbezug der Messgleichung ergibt sich auch eine Normalverteilung für die Likelihood. Der Messoperator R muss linear sein. Insgesamt ergibt sich folgende Messgleichung:

$$d = Rs + n \tag{2.13}$$

In der Formel beschreibt d die Daten und s die *Quantity of Interest*, also das ursprüngliche Signal. R ist die Response, das heißt die Abbildung des Signals in den Datenraum. Die Wahrscheinlichkeit, die Daten d für das Signal s zu erhalten, ist eine Gauß-Verteilung.

$$P(d|s) = \mathcal{G}(d - Rs, N) \tag{2.14}$$

Setzen wir $P(s)$ und $P(d|s)$ in Bayes (siehe Abschnitt 2.2) ein, erhalten wir

$$P(s|d) = \mathcal{G}(s - m, D) \tag{2.15}$$

mit

$$D^{-1} = R^\dagger N^{-1} R + S^{-1} \tag{2.16}$$

und

$$m = D R^\dagger N^{-1} d. \tag{2.17}$$

Der Posterior $P(s|d)$ ist also auch eine Gauß-Verteilung und hat den Erwartungswert m. Die Anwendung des Wiener Filters ist für beliebige Datenmengen möglich. Er ist somit eine wichtige Methode der IFT.

2.15 Inferenz: Lernen aus Daten

Natalie Teplitska, Clemens Ljungh, Constantin Burmeister

Mit dem Wiener Filter Abschnitt 2.14 besitzen wir ein Werkzeug, unser Wissen über die Welt aufgrund von Daten beständig anzupassen. Allerdings gilt der Wiener Filter nur für den Spezi-

alfall, dass $d = Rs + n, P(s) = \mathcal{G}(s, S), P(n) = \mathcal{G}(n, N)$ gilt.

Unser Ziel ist es, aus Daten das ursprüngliche Signal, welches diese Daten erzeugte, zu rekonstruieren. Dazu müssen wir - bildlich gesprochen - die Formel $d = Rs + n$ invertieren und so das *Signal* s bestimmen. Diese Fragestellung wird als inverses Problem bezeichnet.

Der Posterior ist zwar durch Messgleichung, $P(s)$ und $P(n)$ vollständig definiert, doch seine Berechnung ist insbesondere bei großen Datenmengen, wenn der Posterior nicht analytisch lösbar ist, viel zu rechenintensiv und somit teuer. Zur Bewältigung dieser Herausforderung existieren zwei prominente Lösungsansätze.

Das Verfahren Markov-Chain-Monte-Carlo basiert darauf, dass aus $P(s|d)$ lediglich Stichproben gezogen werden und dadurch eine optimale Lösung gefunden wird. Leider ist auch diese Methode noch zu rechenintensiv, weswegen wir sie nicht nutzen.

Eine weitere Möglichkeit besteht darin, den Posterior lediglich anzunähern. Dieser Ansatz benötigt ein Maß für die Quantifizierung des Approximationsfehlers, welches mit der sogenannten Kullback-Leibler-Divergenz (KL-Divergenz) wie folgt definiert ist:

$$\mathrm{KL}(P, P_a) = \int P(s|d) \ln \frac{P(s|d)}{P_a(s|d)} ds \tag{2.18}$$

Das Minimieren von $\mathrm{KL}(P, P_a)$ liefert die optimale Approximation an den Posterior.

Die KL-Divergenz hat einige nennenswerte Eigenschaften. Dazu gehört unter anderem die Lokalität, die dazu führt, dass nur Stellen in die Fehlerberechnung einfließen, über welche die erste Verteilung P eine Aussage trifft. Außerdem folgt die KL-Divergenz dem Prinzip der Gescheitheit, welches besagt: „Wann immer möglich, wähle die zweite Verteilung P_a so, dass diese mit der ersten übereinstimmt“. Zuletzt ist zu erwähnen, dass die KL-Divergenz invariant unter Koordinatentransformationen ist und sich daher nicht verändert, wenn die Koordinatenachsen transformiert werden.

Die KL-Divergenz wendet man bei der variativen Inferenz an. Dabei minimiert man jedoch nicht $\mathrm{KL}(P, P_a)$, sondern $\mathrm{KL}(P_a, P)$, da diese Optimierung weniger rechenintensiv ist. Die Optimierungen im Folgenden minimieren die variative $\mathrm{KL}(P_a, P)$.

2.16 Hierarchisches Modell für korrelierte Felder

Natalie Teplitska, Clemens Ljungh, Constantin Burmeister

Das Projekt *Hierarchisches Modell für korrelierte Felder* beschäftigte sich mit der Bestimmung eines Priors $P(s)$ für korrelierte Felder. Diese Aufgabe wurde mit den Hilfsmitteln der Python-Bibliothek *NIFTy8* gelöst.

Ziel eines bildgebenden Verfahrens ist die Ermittlung der quantity of interest s. Dies kann bei der Radioastronomie die Helligkeit des Himmels oder bei der CT die Dichte des untersuchten Objektes sein. Da weder Helligkeit noch Dichte negativ sein kann, setzen wir $s \geq 0$ voraus. Damit lässt sich eine Normalverteilung für $P(s)$ bereits ausschließen, da eine solche in ganz $\mathbb{R}$ definiert wäre. Außerdem wird angenommen, dass die einzelnen Bildpunkte miteinander korreliert sind, also voneinander abhängen. Ist zum Beispiel ein Pixel schwarz, so ist es unwahrscheinlich, dass sich in direkter Nähe dazu weiße Pixel befinden. In unserem Fall gilt, dass die

Abhängigkeit zunimmt, je kleiner dar Abstand zwischen zwei Pixeln ist. Daraus ergibt sich eine schwierig zu verarbeitende Kovarianzmatrix, die die Korrelation der Pixel zueinander codiert.

Die Annahme, die Korrelation zwischen Pixeln sei normalverteilt, stellt häufig eine gute Näherung dar. In unserem Fall ist es jedoch das Ziel, ein hierarchisches Modell zu erzeugen, um einen Prior mit besseren Näherungen zu verwenden.

Dazu verwenden wir als Zufallsvariable nicht mehr s, sondern ξ, welches mit $P(\xi) = \mathbb{G}(\xi, \mathbb{1})$ standard-normalverteilt ist. Diese ist in jedem Freiheitsgrad mit Erwartungswert 0 und Varianz 1 normalverteilt.

Unsere Aufgabe bestand nun darin, eine Funktion f so zu definieren und zu implementieren, sodass $P(f(\xi))d\xi = P(s)ds$. Die Schwierigkeit liegt hierbei insbesondere in der oben erwähnten Komplexität der Kovarianzmatrix.

Die *Cholesky-Zerlegung* uni-muenster.de stellt eine Möglichkeit dar, einen solchen Prior zu erzeugen, wobei nur eine Dreiecksmatrix mit halb so vielen Einträgen gespeichert werden muss. Dies geschieht, indem im übertragenen Sinn die Wurzel aus S gezogen wird, sodass $S = AA^\dagger$. Auf diese Weise lässt sich eine Funktion definieren, die aus zufälligen Zahlen ξ den Prior berechnet: $s = f(\xi) = A \cdot \xi$. Allerdings ist der Speicherverbrauch der Dreiecksmatrix für $\mathcal{O}(100000)$ Datenpunkte immer noch enorm.

Ein alternatives Verfahren stellt das *Wiener-Chintchin-Theorem* dar. Hier wird die Annahme über die Kovarianz zwischen Pixeln nur über deren Entfernung, nicht jedoch über die Position getroffen. Dabei gilt

$$S = \mathcal{F}^\dagger diag(p(|k|))\mathcal{F} \tag{2.19}$$

Mit $diag(p(|k|))$ wird nur die Diagonale einer Diagonalmatrix gespeichert, welche durch eine Hartley-Transformation[1] erzeugt wird. Sie arbeitet mit der power function $p(|k|)$, die die Kovarianz kodiert. Dabei repräsentiert der Betrag von k die Entfernung zwischen einzelnen Pixeln. Dieses Verfahren ist effizienter, da die Menge der Zahlen, welche in der Diagonalmatrix gespeichert werden muss, linear skaliert und auch die Hartley-Transformation keine vollständige Matrix benötigt, um korrekt ausgeführt zu werden.

Bisher haben wir mit einer vorgegebenen Kovarianz S gearbeitet. Um aus Daten lernen zu können, dürfen wir uns allerdings nicht auf einen bestimmten Prior festlegen. Die Berechnung des Posteriors gestaltet sich wesentlich einfacher, wenn die power function - mit einigen vorausgesetzten Annahmen - über eine Menge von Parametern definiert wird. Diese können jeweils mit Bayes Abschnitt 2.2 immer weiter angepasst werden. Für unseren Fall arbeiteten wir mit drei Parametern .

- Der Parameter *scale* steht für die Standardabweichung der power function und beschreibt somit, wie stark sie variieren kann.
- Wir gehen von einer Funktion aus, die rechts von einem bestimmten Punkt eine Form annimmt, welche einem power-law folgt. Diesen Punkt bezeichnet man als *cutoff*.
- Schließlich bezeichnet *loglogslope* die Steigung der Funktion in dem Intervall nach dem cutoff.

[1] Variante der Fourier-Transformation, welche Real- und Imaginärteil im reellen Raum verrechnet

Alle drei Parameter werden über je zwei Zahlenwerte definiert: den Mittelwert der Erwartungen und seine vermutete Varianz. Im Grunde genommen arbeiten wir also mit sechs Zahlenwerten, die unter Einbezug von Daten immer weiter angepasst werden können.

Damit haben wir es geschafft, einen Prior mittels Zufallszahlen zu generieren, der ein Lernen aus Daten mit kleinstmöglichem Rechen- und Speicheraufwand ermöglicht.

2.17 Radiointerferometrie

Benjamin Knöbel del Olmo, Ole Fleck, Aaron Gschwendt, Mara Germann

2.17.1 Kalibrierung der Radiointerferometriedaten

Das Ziel dieses Projektteils ist es, Kalibrierungsfehler aus den Interferometriedaten herauszuprojizieren. Im Gegensatz zu Beobachtungen durch einzelne Antennenarrays wurde das Schwarze Loch M87 durch das Event Horizon Telescope (EHT) vermessen. Das EHT besteht aus Antennenstationen in verschiedensten Ländern (Chile, Spanien, Hawaii etc.). Der Abstand zwischen den Stationen ermöglicht es sehr kleine Winkel aufzulösen (Ref Radiointerferometrie wg Formel!). Beim EHT kommt nun die Besonderheit der unterschiedlichen Wetterlagen an den verschiedenen Standorten ins Spiel. Es ist mit heutigen Methoden nicht möglich, den Effekt der interferierenden Quellen ausreichend zu bestimmen. Zwei Effekte treten auf: Dynamiken in der Atmosphäre führen zu Phasenverschiebungen und Temperaturschwankungen verursachen eine zeitabhängige Skalierung der Messdaten. Diese Störungen lassen sich durch die Formel

$$d_{abt\lambda} \to g_{at\lambda}\bar{g}_{bt\lambda}d_{abt\lambda}, \qquad g_{at\lambda} \in \mathbb{C} \quad \forall a,t,\lambda$$

darstellen, wobei a und b für ein betrachtetes Antennenpaar (also Antenne a und Antenne b) steht und $\bar{g}_{at\lambda}$, $\bar{g}_{bt\lambda}$ zwei für die jeweiligen Antennenstörung beschreibende komplexe Zahlen sind: Der Abstandsbetrag spiegelt die Signalstärkenreduktion wider, während die komplexe Phase die Phasenverschiebung repräsentiert. Um diese Effekte herauszukürzen bestimmen wir die *Closure Quantities*, eine bereinigte Form der gegebenen Daten. *Closure Phases* behandeln die Phasenverschiebung der empfangenen Welle und *Closure Amplitudes* die Änderung der Signalstärke. Daher muss eine Funktion

$$d^0_{t\lambda} = f_y(d_{ab\lambda}, \quad \forall a,b) \tag{2.20}$$

gefunden werden, die invariant unter

$$d_{abt\lambda} \to e^{i\phi_{at}}e^{-i\phi_{bt}}d_{abt\lambda}d(0) \tag{2.21}$$

$$d_{abt\lambda} \to |g_{at\lambda}||g_{bt\lambda}|d_{abt\lambda}| \tag{2.22}$$

ist. In der ersten Funktion werden die Verschiebungen in der Phase rausgekürtzt, in der zweiten die Amplitudenverschiebung. Für die *Closure Phases* lautet die gesuchte Funktion

$$f(d_{a,b}, d_{b,c}, d_{a,c}) = d_{a,b} + d_{b,c} - d_{a,c} \tag{2.23}$$

Man kann zeigen dass durch Einsetzen in die Funktion die Verschiebung der Daten herausgerechnet wird. Lass $\psi_n = e^{i\phi_{nt}}$ sein.

$$f(\psi_a\bar{\psi}_b d_{a,b}, \psi_b\bar{\psi}_c d_{b,c}, \psi_a\bar{\psi}_c d_{a,c}) = \psi_1\bar{\psi}_b d_{a,b} + \psi_b\bar{\psi}_c d_{b,c} - \psi_a\bar{\psi}_c d_{a,c} \tag{2.24}$$
$$= e^{i\phi_{a,t}-i\phi_{b,t}} d_{a,b} + e^{i\phi_{b,t}-i\phi_{c,t}} d_{b,c} - e^{i\phi_{a,t}-i\phi_{c,t}} d_{a,c} \tag{2.25}$$
$$= d_{a,b} + d_{b,c} - d_{a,c} \tag{2.26}$$
$$= f(d_{a,b}, d_{b,c}, d_{a,c}) \tag{2.27}$$

Für die *Closure Amplitudes* lautet die gesuchte Funktion:

$$f(d_{a,b}, d_{a,c}, d_{a,d}, d_{b,c}, d_{b,d}, d_{c,d}) \stackrel{!}{=} \frac{d_{a,b} \cdot d_{c,d}}{d_{a,c} \cdot d_{b,d}} \qquad bzw. \qquad \frac{d_{a,d} \cdot d_{b,c}}{d_{a,c} \cdot d_{b,d}} \tag{2.28}$$

Folgend werden die *Closure Phases* für die verschiedenen Dreiergruppen bzw. die *Closure Amplitudes* für die Vierergruppen aus Antennenpaaren berechnet. Da jedoch nicht alle Möglichkeiten unabhängig voneinander sind, können Redundanzen herausgerechnet werden. Dafür wird eine Matrix erstellt, in der die Spalten alle möglichen Antennenpaare darstellen und die Zeilen alle linear unabhängige Möglichkeiten die Antennepaare zu verknüpfen darstellt. Bei den *Closure Phases* stehen die Zahlen in der Matrix für das Vorzeichen, bei den *Closure Amplitudes* für den Vorzeichen des Exponents. Folglich bedeuten Nullen, dass die Antennenpaare nicht verwendet werden. Bei n Antennen gibt es $\binom{n-1}{2} = \frac{1}{2}(n-1)\cdot(n-2)$ linear unabhängige Reihen in der Matrix. Daher hat die $n = 4$ *Closure Phase*Matrix $\binom{4-1}{2} = 3$ Reihen:

$$\begin{array}{c} \begin{array}{cccccc} (0,1) & (0,2) & (0,3) & (1,2) & (1,3) & (2,3) \end{array} \\ \left(\begin{array}{cccccc} 1 & -1 & 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 \end{array} \right) \end{array} \tag{2.29}$$

Die *Closure Amplitude* Matrix von n Antennen hat $\frac{1}{2}n*(n-3)$ linear unabhängige Reihen. Die $n = 4$ *Closure Amplitude* Matrix sieht folgend aus:

$$\begin{array}{c} \begin{array}{cccccc} (0,1) & (0,2) & (0,3) & (1,2) & (1,3) & (2,3) \end{array} \\ \left(\begin{array}{cccccc} 1 & -1 & 0 & 0 & -1 & 0 \\ 0 & -1 & 1 & 1 & -1 & 0 \end{array} \right) \end{array} \tag{2.30}$$

Um die *Closure Phase* Matrixen zu generieren verwendet wir die library `numpy` mit folgender Funktion.

```
def genPhaseMatrix(n):
    O=math.comb(n,2)
    mat=np.empty(O,dtype=int)
    pos=[(x,y) for x in range(n) for y in range(x+1,n)]
    maxRank=math.comb(n-1,2)
    for i in range(0,n):
        for j in range(i+1,n):
```

```
for k in range(j+1,n):
    row=np.zeros(O,dtype=int)
    row[pos.index((i,j)),]=1
    row[pos.index((i,k))]=-1
    row[pos.index((j,k))]=1
    newMat=np.vstack([mat,row])
    rnk= np.linalg.matrix_rank(newMat)
    if rnk>np.linalg.matrix_rank(mat):
        mat=newMat
        if rnk==maxRank:
            return mat[1:,:]
```

Die Funktion nimmt als Parameter die Anzahl an Antennen. Zunächst erstellt sie eine leere Numpy Array mit einer Zeile für jedes mögliches Antennenpaar. In dem Array wird die Matrix Zeilenweise erzeugt. Dann werden alle möglichen Wege 3 Antennen miteinander Kombinieren durchlaufen. Ist die Kombination linear unabhängig zu dem Array, wird diese dem Array zugefügt. Dies wird so oft wiederholt, bis entweder alle Möglichkeiten durchgegangen oder die maximale Zeilenanzahl der Matrix erreicht wurde. Der Algorithmus zur *Amplitude Phase* Matrix funktioniert analog. Nachdem die *Closure Quantities* erstellt wurden, werden die Daten der Antennen für jeden Zeitintervall in Phase und Amplitude getrennt und mit den *Closure Quantities* Matrixen multipliziert. Da nicht zu jedem Zeitpunkt gleich viele Antennen aktiv sind (z.B. weil sie im Schatten der Erde liegen) brauchen wir alle *Closure* Matrixen bis $n = 7$. Um am Ende ein Bild zu erzeugen, benötigen wir den Posterior des Bayes Theorem, welcher uns ermöglicht, fehlende Daten durch wahrscheinliche zu ersetzen. Um diesen zu berechnen, setzen wir ein:

$$P(\xi|d) = \frac{P(d|\xi) \cdot P(\xi)}{P(d)} \tag{2.31}$$

$$P(d|\xi) = P(d_{ph}|\xi) \cdot P(d_{amp}|\xi) \tag{2.32}$$

Die Likelihood (1.12) besteht aus zwei Teilen: $P(d_{ph}|\xi)$, der Warscheinlichkeit der *Closure Phases* unter der Bedingung ξ und $P(d_{am}|\xi)$, der Wahrscheinlichkeit der *Closure Amplitudes* unter der Bedingung ξ.

$$-\log P(\xi|d) = \mathcal{H}(\xi|d) = \mathcal{H}(d_{ph}|\xi) + \mathcal{H}(d_{amp}|\xi) + \mathcal{H}(\xi) \tag{2.33}$$

$$= \frac{1}{2} \cdot \left[d_{ph} - f_{ph}(R(sky(\xi)))\right]^{\dagger} N_{ph}^{-1} \left[d_{ph} - f_{ph}(R(sky(\xi)))\right] \tag{2.34}$$

$$+ \frac{1}{2} \cdot \left[d_{amp} - f_{amp}(R(sky(\xi)))\right]^{\dagger} N_{amp}^{-1} \left[d_{ph} - f_{ph}(R(sky(\xi)))\right]$$

$$+ \frac{1}{2} \cdot \xi^{\dagger}\xi$$

Der negative Logarithmus des Posteriors Gleichung (2.33), erlaubt uns, Metric Gaussian Variational Inference, kurz MGVI (**k4.2.mgvi**) anzuwenden. Dabei wird der Parameter ξ so optimiert,

dass man mit einer durch ξ definierte Normalverteilung eine möglichst kleine Differenz zur wahren Posterior erreicht(s. Kullback-Leibler-Divergenz(hier kommt noch eine Referenz!)) Dadurch erhält man folgendes Bild: (Hier wird das Schwarze Loch Foto eingefügt)

2.17.2 Radiointerferometrie-Response und Event-Horizon-Telescope-Daten

Zur Rekonstruktion eines Bildes des Schwarzen Lochs M87* muss basierend auf den Daten des Event-Horizon-Telescopes zunächst eine Response-Funktion implementiert werden, um die Informationen vom Signal- in den Datenraum zu transferieren $s \mapsto d$. Mit der entsprechenden Implementierung wird sich der nachfolgende Abschnitt befassen.

Es werden Daten in Form von acht csv-Dateien verwendet, die im April 2017 aufgenommen wurden. Diese stammen von mehreren Messreihen und wurden zur weiteren Verarbeitung in Form von csv-Dateien gespeichert. Um mit den Messwerten in Python arbeiten zu können, werden die csv-Dateien in NumPy-Arrays konvertiert. Durch das NumPy-Array werden die Daten aus den csv-Dateien auf einem zweidimensionalen Gitter abgebildet.

Zunächst müssen die Frequenzen der Radiowellen ausgelesen werden. Diese elektromagnetischen Wellen werden in unmittelbarer Nähe des Schwarzen Lochs ausgesandt. Es wurden die beiden Frequenzen $f_1 = 227,0707$ GHz und $f_2 = 229,0707$ GHz betrachtet. Darüber hinaus werden auch die Verbindungsvektoren zwischen den Antennen uvw ausgelesen.

Um aus den Messwerten ein Bild erstellen zu können, wird die Anzahl der Pixel sowohl in x-, als auch in y-Richtung auf zunächst 100 festgelegt, wobei dieser Wert variabel ist. Die Größe der Pixel ergibt sich für $\delta\theta$:

$$\delta\theta = 1.22 \cdot \frac{\lambda}{D} = 1.22 \cdot \frac{c}{f \cdot D}$$

Da es sich bei dem Event Horizon Telescope um einen Zusammenschluss von Antennen auf der ganzen Welt handelt, ergibt sich ein Durchmesser von $D_{EHT} \approx D_{Erde} \approx 10.000km$. Desweiteren beschreibt λ die Wellenlänge der Radiowellen und c deren Ausbreitungsgeschwindigkeit, die der Lichtgeschwindigkeit entspricht. In die Formel eingesetzt erhält man:

$$\delta\theta = 1.22 \cdot \frac{3 \cdot 10^8 \frac{\text{m}}{\text{s}}}{229.0707\text{GHz} \cdot 10.000\text{km}} = (1,598 \cdot 10^{-10})\mu\text{as} \tag{2.35}$$

Damit ein erstes Bild des Schwarzen Lochs generiert werden kann, ist es notwendig, die Daten d mittels folgender Formel zu beschreiben. Die Werte für $I_{Amplitude}$ und I_{Phase} sind in den csv-Dateien gegeben:

$$d = I_{Amplitude} \cdot e^{i \cdot I_{Phase}}$$

Aufgrund unvollständiger Informationen, die durch die Distanz der Antennen des Event-Horizon-Telescopes entstehen, muss zur Bilderstellung Bayes'sche Statistik angewandt werden. Im Gegensatz zu NumPy stellt NIFTy entsprechende Funktionen bereit, die dies ermöglichen. Um vom Signalraum I (Signal des betrachteten Objekts) in den Datenraum d (empfangene Daten) zu gelangen, nutzt man in NIFTy die Funktion `dirty2vis` aus der Python-Bibliothek `ducc0`. Diese ist vergleichbar mit der Response R beim Wiener Filter. $R^\dagger$ entspricht der adjungierten Abbildung `vis2dirty`, die es ermöglicht, aus dem bekannten Datenraum auf den

Signalraum zu schließen.

Um diese Funktionen in NIFTy wiederholt anwenden zu können, werden sie in eine Klasse verpackt. Es handelt sich dabei um eine Klasse, die von `ift.LinearOperator` - einem NIFTy-Operator - erbt. Dazu müssen unter Anderem `domain` und `target` initialisiert werden. Der Signalraum wird über `domain` beschrieben, der Datenraum über `target`. In der Klasse werden die Funktionen `dirty2vis` und `vis2dirty` durch die Methoden `TIMES` und `ADJOINT_TIMES` ersetzt.

2.18 Tomographie

2.18.1 Einleitung

Aaron Gschwendt, (Finja Hoffmann)

Die Tomographie ist ein bildgebendes Verfahren, in dem ein Objekt schichtweise untersucht wird. Um diese Schichten zu vermessen, beobachtet man Strahlen, die das Objekt schneiden und auf einer Ebene liegen. Die Menge an Licht, die auf der Strecke absorbiet wurde, entspricht dem Linienintegral:

$$d = \int_\Gamma s(p) dp(\gamma)$$

s ist eine unbekannte Funktion die jedem Punkt im Raum eine optische Dichte zuordnet. d ist das Linienintegral von $s(p)$ entlang dem Pfad γ. d entspricht der Menge an Licht, die zwischen dem Sender und dem Empfänger absorbiert wird und wird experimentell bestimmt.

Gesucht ist die Funktion s, diese lässt sich jedoch nicht analytisch bestimmen. Kennt man aber d von genug Linien in s, kann man mithilfe des Satzes von Bayes mit hoher Auflösung und Sicherheit s bestimmen.

2.18.2 Radioastronomie

In der Astronomie wird die Tomographie z.B. verwendet, um die Form von kosmischen Wolken zu ermitteln. Dafür werden Sterne, deren absolute Helligkeit und Distanz bekannt ist beobachtet. Man vergleicht ihre scheinbare Helligkeit mit der zu erwartende Helligkeit und ermittelt so d für alle Strecken zwischen der Erde und den beobachteten Sternen. d entspricht in dem Fall die Menge an Staub zwischen dem Stern und der Erde. Aus vielen (Distanz, Helligkeit)-Paaren kann man s lernen und Aussagen über die dichte-Verteilung und Form der Wolke treffen.

2.18.3 Computertomografie

Häufig verwendet wird die Tomographie auch in der Medizin, bekannt als Computertomographie(CT).

Bei herkömmlichen Röntgenuntersuchungen wird Röntgenstrahlung durch das abzubildende Objekt auf einen Röntgenfilm oder einer Sensorplatte geleitet. Das 3d-Objekt wird dabei auf eine 2d-Fläche projiziert. Der Nachteil dieser Methode ist, dass sich Teile vom Objekt überlagern

können und nicht erkannt werden kann, ob es sich um ein Objekt mit hoher Absorption oder mehrer Objekte mit geringer Absorption handelt.

Beim CT drehen sich im $180°$ Winkel eine Röntgenröhre und Detektor um den Patienten und nehmen dabei in kleinen Abständen Messpunkte auf. Dies wird für mehrere Schichten entlang des zu untersuchenden Körperteils ausgeführt. Der Röntgenstrahl und Detektor ist so breit, dass bei jedem Messpunkt die ganze Breite des Körperteils in einem Streifen ergriffen wird.

Daraus entsteht für jede Schicht ein Sinogram Abb. 2.4 bei der eine Achse (hier Y-Achse) das Absorptionsprofil und die andere (hier X-Achse) dem Winkel entspricht. Jede Stelle im Objekt bildet eine Sinuskurve; ihr Abstand vom Mittelpunkt entspricht der Amplitude und ihr Winkel im vergleich zum Startpunkt der Phasenverschiebung der Sinuskurve. Mit dem bayeschen Verfahren kann man ein Bild von der Schicht mit hoher Genauigkeit rekonstruieren. Führt man dies für jede durch, Schicht erhält man ein 3d-Rendering des Objekts.

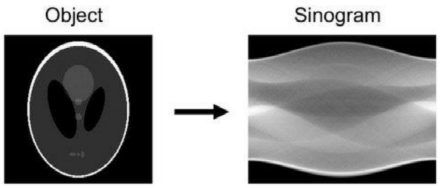

Abbildung 2.4: Abbildung eines Objektes als Sinogram

2.18.4 CT in 3D

Finja Hoffmann, Chuyang Wang

Im Rahmen der Projektarbeit soll aus den gemessenen Daten d die Funktion des Signals $s(p)$ (*quantity of interest*) in Abhängigkeit von einem 3D-Punkt $p \in \mathbb{R}^3$ rekonstruiert werden, welche die Dichte eines Objekts in einem diskreten 3-dimensionalen Raum beschreibt. Generell lässt sich der Datenvektor $d \in \mathbb{R}^N$ wie folgt berechnen:

$$d = Rs + n, \tag{2.36}$$

wobei jeweils $R \in \mathbb{R}^{N \times M}$ das *Response*, $s \in \mathbb{R}^M$ die Signale und $n \in \mathbb{R}^N$ das Rauschen beschreiben. Jede Komponente von d entspricht einem Linienintegral aus **??**.

2.18.5 Line-Of-Sight Response

Angenommen, die Start- und Endpunkte der Messungen seien bereits gegeben und bilden jeweils die Strecken. Aus diesen Daten kann man die Matrix R konstruieren. Anders gesagt repräsentiert R die Start- und Endpunkten. Da die Signale in Form von diskreten Datenwürfel angegeben sind, ist R also eine Gewichtung der gegebenen Signale. Als Gewichtung eines Datenwürfels gilt der euklidische Abstand zwischen den beiden Schnittpunkten der Strecke mit den Seiten des Würfels. Siddon (1985) hat dafür einen effizienten Algorithmus geliefert, dessen Ansatz hier umgesetzt wird.

2.18.6 Algorithmus von Siddon

Siddons Ansatz nach könnte man die Schnittpunkte einer Sichtlinie mit den x-, y- und z-Achsenebenen jeweils einzeln bestimmen und in sortierte Listen A, B und C speichern. Sei jeweils

$$P_{start} = \begin{pmatrix} x_{start} \\ y_{start} \\ z_{start} \end{pmatrix} \tag{2.37}$$

$$P_{end} = \begin{pmatrix} x_{end} \\ y_{end} \\ z_{end} \end{pmatrix} \tag{2.38}$$

Betrachtet man zunächst die x-Achsenebene. Ohne Beschränkung der Allgemeinheit sei $x_{start} < x_{end}$. Man rundet die x-Koordinate des Startpunktes in die Richtung der Strecke auf und berechnet die Differenz x^*. Um die y- und z-Koordinaten dieses ersten Schnittpunkts herauszufinden, muss man noch die Steigungen gegenüber dieser beiden Achsen $m_{yx} = \frac{y_{end} - y_{start}}{x_{start} - x_{end}}$ und $m_{zx} = \frac{z_{end} - z_{start}}{x_{start} - x_{end}}$ berechnen. Somit erhält man den ersten Schnittpunkt

$$S_{x,0} = \begin{pmatrix} x_{start} + x^* \\ y_{start} + x^* \cdot m_{yx} \\ z_{start} + x^* \cdot m_{zx} \end{pmatrix} \tag{2.39}$$

Die weiteren Schnittpunkte $S_{x,i}$ kann man rekursiv konstruieren, indem man jeweils eine Einheit in die x-Richtung geht und die Änderungsrate gegenüber der y- und z-Achsen jeweils aufaddiert, also nämlich

$$S_{x,i} = S_{x,i-1} + \begin{pmatrix} 1 \\ 1 \cdot m_{yx} \\ 1 \cdot m_{zx} \end{pmatrix} \tag{2.40}$$

Die Liste A besteht aus allen Schnittpunkten $[S_{x,0}, S_{x,1}, ..., S_{x,n}]$. Die Listen B und C berechnet man analog.

Anschließend muss man die drei Listen in eine einzelne sortierte Liste $\mathbf{S}$ zusammenführen. Diese geschieht, indem man alle Schnittpunkte aus A, B und C nach den x-Koordinaten aufsteigend sortiert. Sollte $x_{start} = x_{end}$ sein, dann nutzt man die y- oder z-Achse als den Sortierschlüssel.

Hat man alle sortierten Schnittpunkte, so kann man auch die Gewichtungen, i.e. die Abstände zwischen den jeweiligen Punkten, einfach berechnen. Jede dieser Gewichtungen wird einem Datenwürfel zugeordnet, durch den diese Teilstrecke durchdringt. Für diese Sichtlinie kann dann ein Zeilenvektor r_i erstellt werden, welcher diese Zuordnung von Gewichtungen speichert. Man beobachte, dass jede Sichtlinie nur wenige Datenwürfel im 3D-Raum durchgeht. Also sind die meisten Gewichtungen für diese Sichtlinie null.

Daraus entsteht für jede Schicht ein Sinogram (Abbildung 1.1), bei dem eine Achse (hier Y-Achse) das Absorptionsprofil und die andere (hier X-Achse) dem Winkel entspricht. Jede Stelle im Objekt bildet eine Sinuskurve; Ihr Abstand zum Mittelpunkt entspricht der Amplitude und ihr Winkel die Phasenverschiebung. Mithilfe vom Satz von Bayes kann man ein Bild von der Schicht rekonstruieren. Macht man dies für jede Schicht erhält man ein 3d-Rendering.

2.18.7 Walnuss CT-Messdaten

Leo Bergmann, Cedric Balzer

Das Ziel des Projekts ist es ein Bild einer Walnuss, anhand von CT-Scandaten (siehe Abschnitt 2.18), zu rekonstruieren. Die Herausforderung besteht darin die Unsicherheit in der DIchteverteilung des zu vermessenden Objekts zu quantifiziern. Aus ein Bayes'schen Perspektive (siehe Abschnitt 2.2) quantifiziert der Postirior die Unsichertheit. Die präzise Beschreibung ist bei medizinischer Bildgebung wichtig. Die öffentlich zugänglichen Daten verfügbar unter `https://zenodo.org/record/1254206#.Ys6OHnZBw7c`, auf welchen unser Projekt beruht, enthalten Sinogramm und Schichtaufnahme einer Walnuss, welche in `.mat`-Dateien gespeichert sind. In dem Versuch wurde eine Walnuss zwischen eine punktförmige Strahlungsquelle und einen flachen Schirm positioniert (**??**). Nach jeder Messung wurde die Nuss um 3° gegen den Uhrzeigersinn gedreht. Es wurde 120 Messungen durchgeführt, welche von dem ebenen Detektor aufgenommen wurden. Diese Daten wurden auf 328 Pixel heruterskaliert und in einer Matrix `m` gespeichert. Damit das Sinogramm in ein Bild der Dichteverteilung zurückprojeziert werden kann sind alle Start- und Endpunkte der Sichtline notwendig. Diese errechnen sich aus der Position der Quelle und der Position der einzelnen Pixel des Screens.

Anstelle einer Drehung der Nuss gegen den Uhrzeigersinn betrachten wir, äquivalent dazu, eine kreisförmigen Rotation des Detektors (Screen) und der Strahlungsquelle (S) um die Walnuss (N) (siehe Abb. 2.6). Es bieten sich Polarkoordinaten an, um die Position der Quelle und des Detektors relativ zu Walnuss zu bestimmen, jedoch müssen alle Punkte für die Weiterverarbeitung im kartesischen Koordinatensystem angegeben. Da die Winkel und Abstandsmaße in Bezug auf die Walnuss angegeben sind, wird diese als Koordinatenursprung definiert. Ausgehend davon wird die Quelle zunächst an die Stelle $\begin{pmatrix}-110\\0\end{pmatrix}$ gelegt (siehe Abb. 2.5, Abb. 2.6). Dadurch wird die Startposition des Detektors festgelegt, der durch seinen Anfang (U) an Stelle $\begin{pmatrix}190\\57.4\end{pmatrix}$ und seinen Endpunkt (L) an Stelle $\begin{pmatrix}190\\-57.4\end{pmatrix}$ bestimmt wird. S‘ entspricht S nach einer Drehung der Quelle und des Detektors um den Winkel δ. Der Abstand zwischen S‘ und N ist dabei immer 110 mm. Der Versuchsaufbau kann somit als Einheitskreis mit Mittelpunkt N interpretiert werden, der um den Faktor 110 mm gestreckt wurde. Dadurch ist im karthesischen Koordinatensystem die horizontale Koordinaten-Komponente von S‘ gleich $\cos(\delta) * 110\,\text{mm}$ und die vertikale Komponente gleich $\sin(\delta) * 110\,\text{mm}$. Der Winkel $\sphericalangle U - N - L$ hat eine Größe von etwa 34°, somit hat der Winkel $\alpha = \sphericalangle L - N - C$ eine Größe von 17°. Der Abstand von N zu L kann mithilfe des Satzes von Pythagoras berechnet werden $d = \sqrt{(300\,\text{mm} - 110\,\text{mm})^2 + (114.8\,\text{mm}/2)^2} = 198.48\,\text{mm}$. Der Winkel α addiert mit δ ergibt zusammen mit der Länge d die Koordinaten L im Polarkoordinaten. Um U‘ zu berechnen muss α von δ subtrahiert werden. Die entsprechenden karthesischen Koordinaten erhalten wir nun auf gleiche Weise bei S‘. Die restlichen 328 Pixelpositionen des Detektors sind gleichmäßige auf der Verbindumgsstecke zwischen U und L verteilt. Mit den erhaltenen Start- und Endpunkten können die Helligkeitswerte zum Raum der Dichteverteilung zurückprojiziert werden.

Literatur

Ramond, P. (2022). „The Abel-Ruffini Theorem: Complex but Not Complicated“. *The American Mathematical Monthly* 129.3, S. 231–245.

Siddon, R. L. (1985). „Fast calculation of the exact radiological path for a three-dimensional CT array“. *Medical Physics* 12.2, S. 252–255.

Diese Datei wurde mit DSA-Dokumentations-Vorlage Version 1.5 vom 2022/07/12 erstellt.

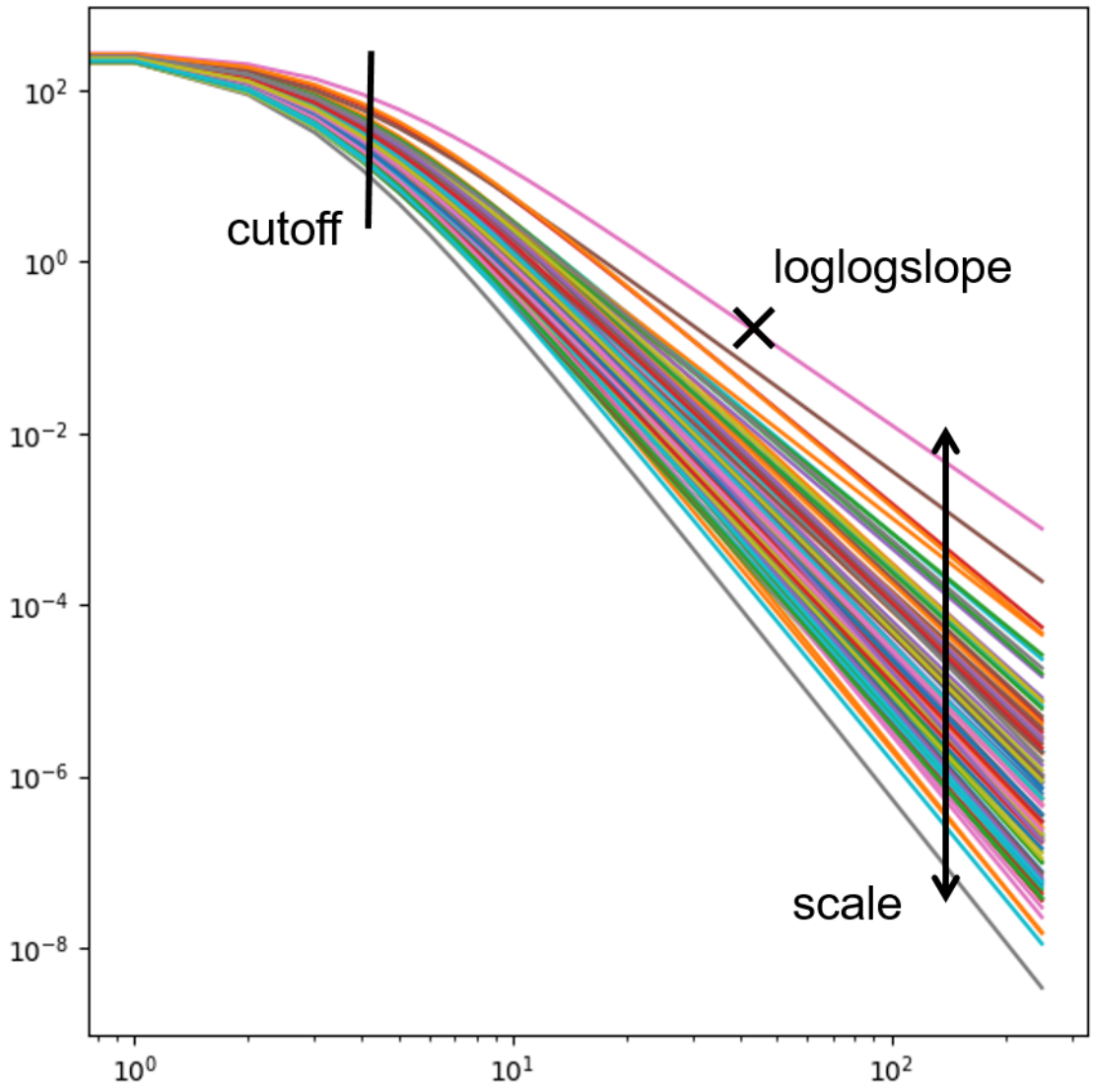

Abbildung 2.3: 100 beispielhafte power functions und ihre Parameter

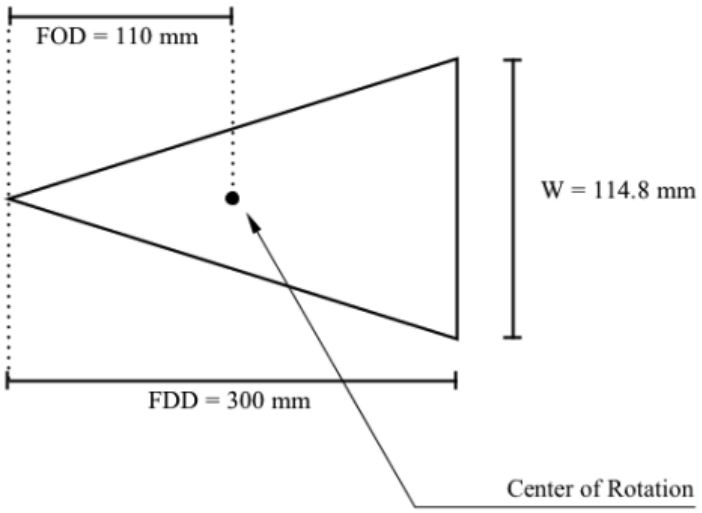

Abbildung 2.5: Aufbau des Versuchs mit dessen Geometrie aus **k4.2.art.walnutXRayempty citation**.
FOD steht für „Focus-to-object“ distance, FDD für „Focus-to-detector“ distance und W für die „Width of the detector“.

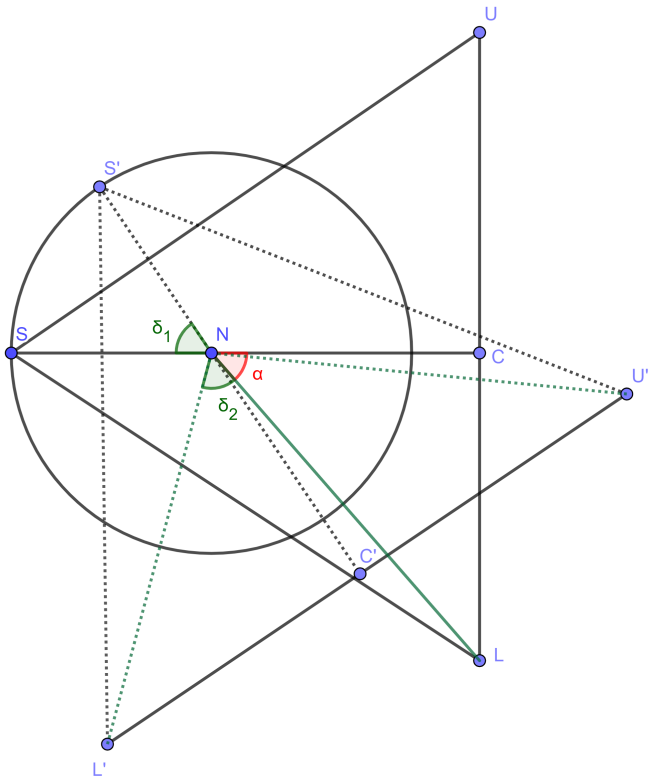

Abbildung 2.6: Versuchsaufbau skizziert.
S steht für die Startpunkt der Strahlenquelle, N ist die gescannte Walnuss, U repräsentiert die obere äußere Grenzen des Detektors und L respektive die Untere, δ steht für den Laufwinkel welcher in 3° Schritten erhöht wird, α für den Winkel zwischen X-Achse und L. C ist der Mittelpunkt des Detektors.