BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Prof. Dr.-Ing. Kristian Hildebrand

Visual and Scientific Computing
Ausgleichsrechnung und Computertomographie

WiSe 2020/21

Das Lernziel dieser Übungsaufgabe ist die lineare Ausgleichsrechnung bzw. die Methode der kleinsten Fehlerquadrate an zwei Beispielen. Erstens das Computertomographie-Verfahren und zweitens das 'Fitting' einzelner Sensordaten. Hierfür müssen Sie lineare Gleichungssysteme selbstständig aufstellen und mit Hilfe von Numpy lösen.

Vorbereitung: Bitte installieren Sie für ihre Anaconda Umgebung noch folgendes Pythonpaket nach:

```
conda install matplotlib
```

Aufgabe 1: Computertomographie (5 Punkte)

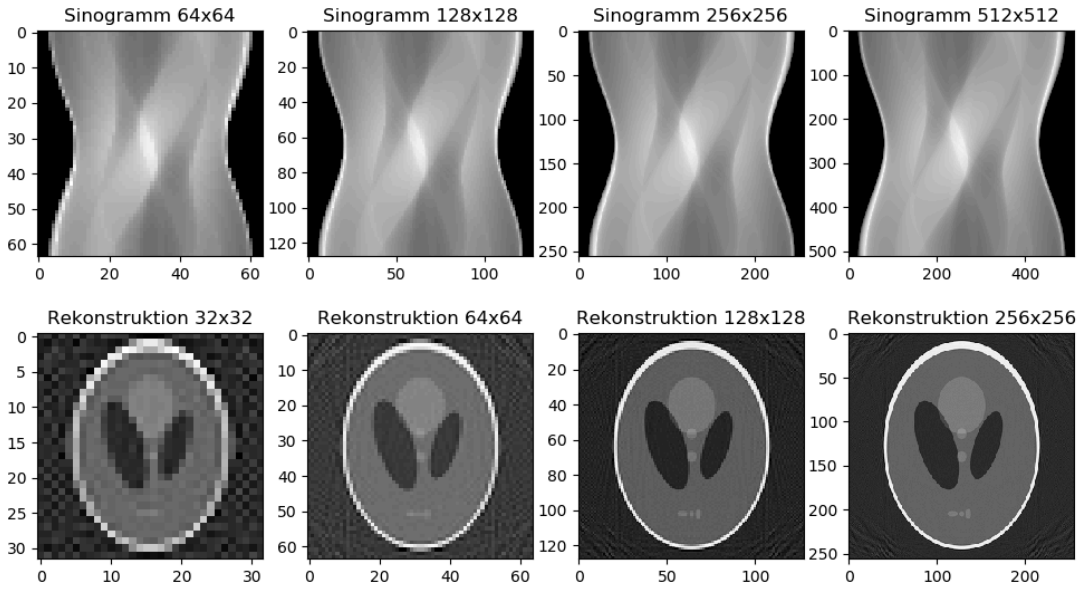

Abbildung 1: Die obere Zeile zeigt Sinogramme welche mit der *trace* Funktion aus *tomography.py* erstellt wurden. Die untere Zeile zeigt die rekonstruierten Dichten. **Für die Abgabe reicht es mir, wenn Sie das für die Auflösungen 32x32 und 64x64 durchrechnen und abgeben. Bitte geben Sie die Plots mit ab, sonst behaltet ich mir Punktabzüge vor.**

Hinweis: Schauen Sie vor der Bearbeitung bitte in die Unterlagen zur Vorlesung (Tafelbeispiel und Notebook).

Bei der Computertomographie werden nicht beliebige Strahlen verwendet, sondern eine Reihe von Aufnahmen mit parallelen Strahlen aus verschiedenen Winkeln. Dies wird in den Unterlagen nochmals beschrieben. Eine Matrix, die in jede Zeile die für eine Aufnahme ermittelten Intensitäten $I_1^{(k)}$ enthält, heißt Sinogramm (Abbildung 1).

Nutzen Sie für die Bearbeitung *tomography_solve.py*.

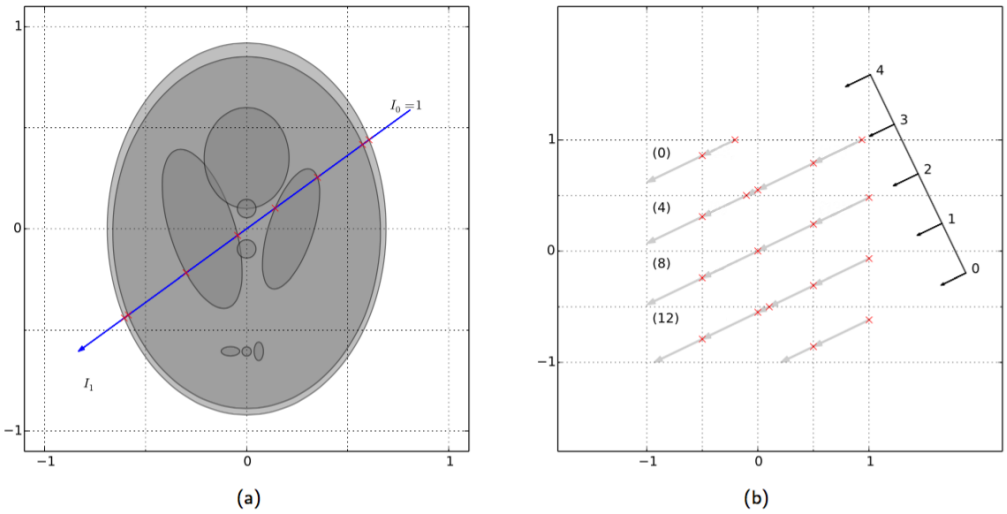

Abbildung 2: (a) Die Funktion trace(r,d) berechnet für einen Strahl die Intensität, welche der Strahl nach dem Durchdringen des Testobjektes hat. (b) Die Funktion grid_intersect(n,r,d) berechnet den Schnitt eines Rasters mit einer Menge von parallelen Strahlen.

a) **Implementieren Sie die Berechnung eines Sinogramms in Python.** Die Eingabe ist die Anzahl der Strahlen pro Aufnahme und die Anzahl der Aufnahmen. Plazieren Sie die Aufnahmen ($\phi \in [0, 180°)$) uniform entlang des oberen Halbkreises. Verwenden Sie dazu die in *tomograph.py* bereitgestellte Funktion *trace(r,d)* (Abbildung 2(a)). Als Eingangsintensität wird dabei $I_0 = 1$ angenommen. Erstellen Sie Sinogramme mit der Auflösung 64×64 und 128×128. Visualisieren Sie diese mit der Funktion *imshow* von matplotlib und reichen Sie die Visualisierungen in Form einer PNG Datei ein. Hinweis: Sie finden einige hoffentlich hilfreiche Kommentare im *tomography_solve.py*.

b) **Implementieren Sie eine Python Funktion**, welche die Matrix A für eine gegebene Anzahl von Strahlen K und ein Raster der Größe $n \times n$ berechnet. Verwenden Sie dazu die in *tomograph.py* bereitgestellte Funktion *grid_intersect(n,r,d)* (siehe Abbildung 2(b)), um die Längen der Strahlen in einem Quadranten zu bestimmen.

c) **Bestimmen Sie die Dichten** für die in (a) berechneten Sinogramme durch Lösen des Ausgleichsproblems $A^T Ax = A^T b$. Nutzen Sie dafür die numpy-Funktion *np.linalg.solve*. Visualisieren Sie die ermittelten Dichten mit der Funktion *imshow* von matplotlib und reichen Sie die Visualisierungen in Form einer PNG Datei ein (so wie in Abbildung 1 zu sehen) (Es reichen die Rekonstruktionen mit Auflösung 32×32 und 64×64).

Hinweis: Es gibt mehrere zusätzliche Hilfestellungen im SourceCode.

Aufgabe 2: Methode der kleinsten Fehlerquadrate oder Ausgleichsrechnung (4 Punkte)

Angenommen wir haben eine Menge von gemessener Datenpunkte in 2D $(x1, y1), (x2, y2), \ldots, (x_n, y_n)$ und wir vermuten, dass es einen linearen Zusammenhang zwischen x_i und y_i gibt. D.h. wir suchen eine Funktion $f(x) = y = ax + b$, die für alle Datenpaare möglichst "gut" erfüllt ist, es soll also gelten $f(x_i) \approx y_i$. Die Terme $ax_i + b = y_i$ lassen sich in Matrixschreibweise ausdrücken.

$$X = \begin{pmatrix} x_0 & 1 \\ x_1 & 1 \\ \vdots & \\ x_n & 1 \end{pmatrix}, \vec{a} = \begin{pmatrix} a \\ b \end{pmatrix}, \vec{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

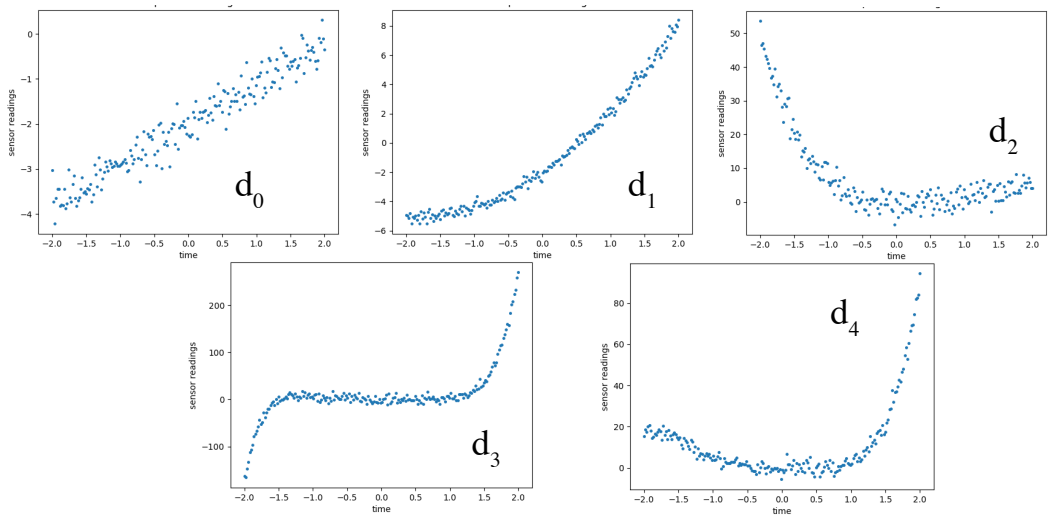

Abbildung 3: Rauschbehaftete Sensordaten in 2D. Wir suchen das Polynom, dass uns einen möglichst geringen Fehler zu allen Messdaten liefert, aber möglichst niedrigen Grades ist.

Dann gilt:

$$X \begin{pmatrix} a \\ b \end{pmatrix} = \vec{y}$$

Dies ist ein überbestimmtes Gleichungssystem und wir werden (höchstwahrscheinlich) keine Koeffizienten a, b finden, sodass das System exakt gelöst wird. Wir können aber eine Lösung a', b' finden, sodass der Abstand des Vektors $X \begin{pmatrix} a' \\ b' \end{pmatrix}$ zu $\vec{y}$ minimal ist. Wir suchen also das Minimum des Fehlers E:

$$E = \left\| X \begin{pmatrix} a \\ b \end{pmatrix} - \vec{y} \right\|$$

In der Vorlesung haben wir gesehen, dass diese Bedingung genau dann erfüllt ist, wenn gilt:

$$X^T X \begin{pmatrix} a \\ b \end{pmatrix} = X^T \vec{y}$$

Dieses Problem lässt sich auf gleiche Weise auch für Funktionen höheren Grades lösen, wenn man die Matrix X dementsprechend anpasst. Zum Beispiel $f(x) = y = ax^3 + bx^2 + cx + d$ wäre eine Funktion dritten Grades mit den Koeffizienten a, b, c, d. Die Matrix X sowie $\vec{a}$ und $\vec{y}$ sähen dann so aus:

$$X = \begin{pmatrix} x_0^3 & x_0^2 & x_0 & 1 \\ x_1^3 & x_1^2 & x_1 & 1 \\ \vdots & & & \\ x_n^3 & x_n^2 & x_n & 1 \end{pmatrix}, \vec{a} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \vec{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Gegeben sei eine Reihe von Messdaten im $\mathbb{R}^2$, wie in Abbildung 3 zu sehen, Wir suchen nun das Polynom, dass uns einen möglichst geringen Fehler zu allen Messdaten liefert, aber möglichst niedrigen Grades ist. Nutzen Sie für die Bearbeitung *polynomial_fit.py*.

a) **Laden Sie die Punktdaten** (die y-Werte) aus dem Ordner *data* und stellen diese mit matplotlib dar. Hinweis: Schauen Sie sich die Funktion *np.load* an. Die 200 Punktdaten wurde an folgenden x-Werten gemessen: *x = np.linspace(-2, 2, 200)*.

b) **Stellen Sie entsprechend des Grades der Funktion eine Matrix X auf**. Für eine Funktion ersten Grades (lineare Funktion) haben wir das in der Vorlesung bereits gemacht (bzw. siehe oben). Schreiben Sie dafür die Funktion *create_matrix*, die das für einen angegebenen Grad macht, sodass $X\vec{a} = \vec{y}$ gilt.

c) **Lösen Sie das lineare Ausgleichsproblem für die gegebenen Punkte nach dem oben beschriebenen Prinzip**. Um den richtigen Grades des Polynoms zu finden, sollen Sie in einem Loop jeweils den Grad der Funktion erhöhen (bis max. 20) und den Fehler E (Residual) messen und ausgeben. Der Fehler ergibt sich durch den Abstand der Messdaten ($\vec{y}$) zu dem gefundenen Polynom.

d) **Stellen Sie die Messdaten sowie das Polynom mit Hilfe der ermittelten Koeffizienten mit matplotlib dar** und speichern diese als PNG Datei ab. Hinweis: Schauen Sie sich die Funktion *np.poly1d* an.

e) **Diskutieren Sie in einem kleinen Text** (einfach eine kleine Textdatei mit abgeben), welchen Polynom-Grad Sie zu den entsprechenden Messdaten wählen würden und warum? Bzw. warum es sinnvoll ist möglichst niedrige Polynomgrade zu verwenden.

Abgabe Die Bearbeitungszeit der Teilaufgabe ist für zwei Wochen ausgelegt. Die Abgabe soll via Moodle bis zu dem dort angegebenen Termin erfolgen. Verspätete Abgaben werden mit einem Abschlag von 3 Punkten je angefangener Woche Verspätung belegt. Geben Sie bitte jeweils nur eine einzige .zip-Datei mit den Quellen Ihrer Lösung ab.

Ich behalte mir vor stichprobenartig Ihre Lösungen in der Übung demonstrieren und erläutern zu lassen. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung!