Wissenschaftliches Rechnen

Aufgabenblatt 2 (Praxis)

wr@cg.tu-berlin.de

WiSe 2019/2020

Allgemeine Hinweise:

- Die Aufgaben sind von jeder/m Studierenden *einzeln* zu bearbeiten und abzugeben (Plagiate werden entsprechend der Studienordnung geahndet).
- Verwenden Sie die vorgegebene Code-Basis. Die zu implementierenden Funktionen befinden sich in der Datei **main.py**. Ihr Code ist an den mit `# TODO: ...` gekennzeichneten Stellen einzufügen. Die `NumPy`-Funktionen, welche Sie zur Lösung einer Aufgabe nicht verwenden dürfen, sind unter `Forbidden` in der Docstring Beschreibung der entsprechenden Funktion aufgelistet.
- Wir stellen einige rudimentäre Unit-Tests zur Verfügung, welche Sie verwenden sollen, um die Funktionalität ihres Codes zu testen. Sie sollten diese Tests während der Implementierung Ihrer Lösung vervollständigen (Funktionalität beschrieben in Python `unittest`). Sie können die Tests mit dem Aufruf `python3 tests.py -v [Tests.test_<function>]` ausführen.
- Bitte reichen Sie die Datei `main.py` mit Ihren Lösungen bis **Montag, den 02.12.2019, um 14:00 Uhr** auf `https://autolab.service.tu-berlin.de` mit ihren Zugangsdaten ein. Ein mehrfacher Upload bis zum Abgabeende ist möglich. Die letzte Version wird bewertet.

Aufgabe 1: Gaußsches Eliminationsverfahren mit Pivoting (3 Punkte)

In dieser Aufgabe soll die Lösung von linearen Gleichungssystemen durch das Gaußsche Eliminationsverfahren und Rückwärtseinsetzen implementiert werden.

Aufgabe 1.1: Gauß-Elimination (2 Punkte)

Implementieren Sie das Gaußsche Eliminationsverfahren sowohl ohne als auch mit Pivoting in der Funktion `gaussian_elimination()`. Wenn die Eingabedaten inkompatible Größen haben, die Matrix nicht quadratisch oder das Gleichungssystem ohne Pivoting nicht lösbar ist, dann soll ein `ValueError` erzeugt werden. In `tests.py` gibt es bereits einen Test, welcher Ihre Implementierung mit zufällig erzeugten Eingabedaten testet.

Aufgabe 1.2: Rückwärtseinsetzen (1 Punkt)

Implementieren Sie Rückwärtseinsetzen in der Funktion `back_substitution()`. Die Funktion bekommt eine obere Dreiecksmatrix und einen Vektor als Parameter übergeben. Wenn die Argumente der Funktion inkompatibel sind (die Größen nicht übereinstimmen) oder wenn es keine/unendlich viele Lösungen beim Einsetzen gibt, soll ein `ValueError` erzeugt werden.

Testen Sie Ihre Implementierung für $\mathbf{A}\mathbf{x} = \mathbf{b}$ und den Werten

$$\mathbf{A} = \begin{pmatrix} 11 & 44 & 1 \\ 0.1 & 0.4 & 3 \\ 0 & 1 & -1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

und vergleichen Sie Ihre Ergebnisse mit der exakten Lösung $\mathbf{x} = (-1732/329, 438/329, 109/329)^T$.

Aufgabe 2: Lösung von Gleichungssystemen mit der Cholesky-Zerlegung (3 Punkte)

Aufgabe 2.1: Berechnung der Cholesky-Zerlegung (2 Punkte)

Implementieren Sie die Cholesky-Zerlegung einer Matrix in der Funktion `compute_cholesky()`. Die Cholesky-Zerlegung einer symmetrischen, positiv definiten Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ ist gegeben durch $\mathbf{A} = \mathbf{L}\mathbf{L}^T$. Dabei ist $\mathbf{L}$ eine untere Dreiecksmatrix. Ihre Funktion soll den Cholesky-Faktor $\mathbf{L}$ mit der in der Vorlesung vorgestellten Formel berechnen und zurückgeben. Für Eingabematrizen, die nicht die Voraussetzungen für die Cholesky-Zerlegung erfüllen, soll ein `ValueError` erzeugt werden.

Aufgabe 2.2: Lösung von Gleichungssystemen (1 Punkt)

Mithilfe der Cholesky-Zerlegung $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ können lineare Gleichungssysteme $\mathbf{A}\mathbf{x} = \mathbf{b}$ effizient gelöst werden. Implementieren Sie die Funktion `solve_cholesky()`, welche einen Cholesky-Faktor $\mathbf{L}$ und den Vektor $\mathbf{b}$ als Parameter übergeben bekommt und die Lösung $\mathbf{x}$ des Gleichungssystems $\mathbf{L}\mathbf{L}^T\mathbf{x} = \mathbf{b}$ zurückgibt. Für die Implementierung dürfen sie die Funktion `back_substitution` aus Aufgabe 1.2 des Aufgabenblatts verwenden. Überprüfen Sie, ob die Eingabematrix $\mathbf{L}$ tatsächlich eine quadratische, untere Dreiecksmatrix ist und werfen Sie einen `ValueError`, falls dies nicht der Fall ist.

Aufgabe 3: Lineare Ausgleichsrechnung (4 Punkte)

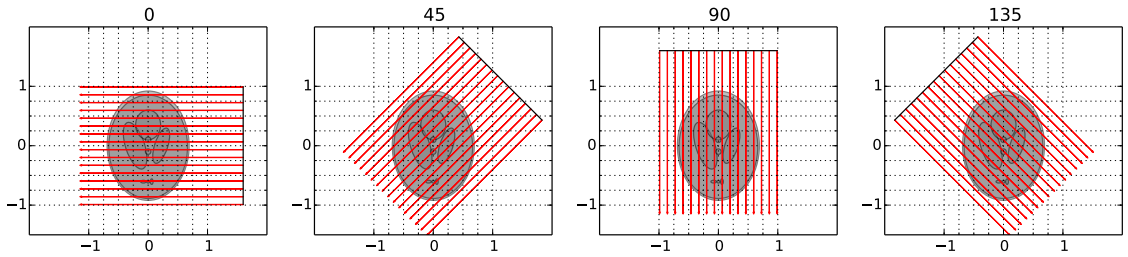


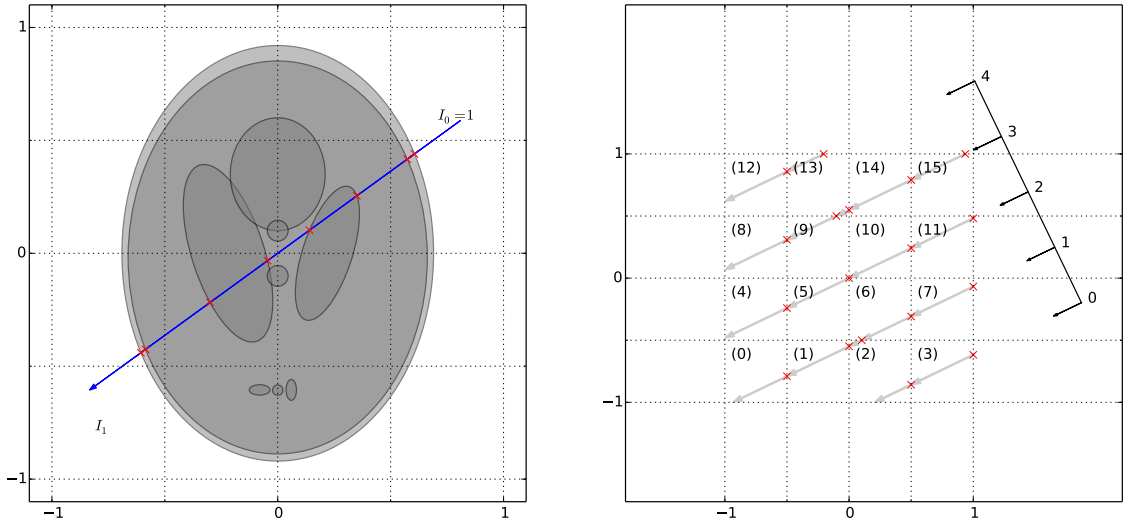
Abbildung 1: Bei der Computertomographie werden aus verschiedenen Winkeln eine Reihe paralleler Röntgenstrahlen ausgesendet und der Intensitätsverlust der Strahlen gemessen, welcher beim Durchdringen des Objektes auftritt.

In dieser Aufgabe implementieren Sie den Rekonstruktionsschritt eines Tomographen, welcher die (logarithmischen) Dichten c'_i in den Zellen eines 2D-Volumens bestimmt. Diese Dichten können als ein sogenanntes tomographisches Bild dargestellt werden, siehe Abb. 3.

Um die Eingangsdaten zu bestimmen, steht Ihnen die Funktion `take_measurements()` zur Verfügung, welche jeweils eine Messung des Tomographen aus einem Winkel θ durchführt (siehe Abb. 1) und ihnen sowohl die (log-)Intensitätsverhältnisse g'_i aller m parallelen Strahlen nach dem Durchgang durch das Volumen zurückgibt als auch die Längen der Strecken, welche jeder Strahl der Messung durch die Zellen des Volumens zurücklegt, siehe Abb 2. Dabei gibt die Funktion nur die Längen von Zellen zurück, die auch tatsächlich von einem Strahl getroffen wurden (genauere Informationen finden Sie in der Beschreibung der Funktion).

Um eine robuste Lösung zu erhalten, müssen k Messungen aus verschiedenen Richtungen durchgeführt werden. Kombiniert man die Daten aus den Messungen, so kann das tomographische Rekonstruktionsproblem durch folgendes lineare Gleichungssystem beschrieben werden:

$$\underbrace{\begin{pmatrix} l_{0,0} & l_{0,1} & \cdots & l_{0,n-1} \\ l_{1,0} & l_{1,1} & \cdots & l_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ l_{m\cdot k,0} & l_{m\cdot k,1} & \cdots & l_{m\cdot k,n-1} \end{pmatrix}}_{\mathbf{L}} \cdot \underbrace{\begin{pmatrix} c'_0 \\ c'_1 \\ \vdots \\ c'_{n-1} \end{pmatrix}}_{\mathbf{c}} = \underbrace{\begin{pmatrix} g'_0 \\ g'_1 \\ \vdots \\ g'_{m\cdot k\cdot} \end{pmatrix}}_{\mathbf{g}} . \tag{1}$$



(a) (b)

Abbildung 2: Links: Abschwächung der Intensität eines Strahles beim Durchgang durch ein Volumen. Dies wird verwendet, um die Dichte über einem diskreten Gitter zu rekonstruieren. Rechts: Länge von verschiedenen, parallelen Strahlen in Gitterzellen.

Aufgabe 3.1: Aufstellen des linearen Systems (2 Punkte)

Stellen Sie mit Hilfe der Funktion `setup_system_tomograph()` das lineare Gleichungssystem in Gl. 1 auf. Benutzen Sie dazu `take_measurement()`, um `n_shot` Messungen aus äquidistant in $[0, \pi)$ verteilten Richtungen durchzuführen. Achten Sie darauf, dass die von `take_measurement()` zurückgegebenen Daten an den korrekten Stellen in $\mathbf{L}$ und $\mathbf{g}$ eingefügt werden.

Aufgabe 3.2: Rekonstruktion des Bildes (2 Punkte)

Implementieren Sie die Funktion `compute_tomograph()` und rekonstruieren Sie mit Hilfe Ihrer Implementierung der Cholesky-Zerlegung das tomographische Bild (falls Sie die Cholesky-Zerlegung nicht selbst implementiert haben, können Sie auf die Implementierung in `NumPy` zurückgreifen). Eine Beispielrekonstruktion für 64 Messungen mit jeweils 64 Strahlen und einem 32×32 Gitter ist in Abb. 3 (ganz links) gezeigt.

Hinweis: Auch wenn die Matrix $\mathbf{A}^T\mathbf{A}$ aus mathematischer Sicht positiv definit ist, kann es auf Grund der Verwendung von Gleitkommazahlen passieren, dass dies numerisch nicht der Fall ist. Da eine Computer-Implementierung der Cholesky-Zerlegung nur auf numerisch positiv definiten Matrizen das gewünschte Ergebnis liefert, behilft man sich in der Praxis damit, ein kleines (z.B. das 10-fache der Maschinengenauigkeit) positives Vielfaches der Identitätsmatrix auf eine Eingangsmatrix zu addieren, wenn diese nicht numerisch positiv definit ist.

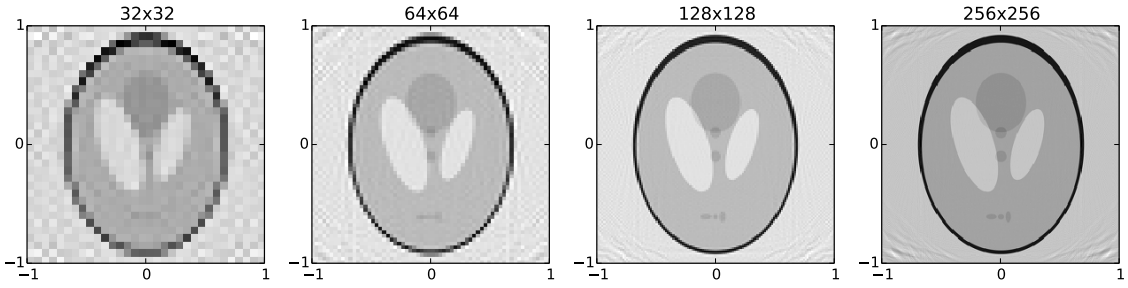


Abbildung 3: Tomographische Bilder, durch die entsprechenden Dichtewerte bestimmt, für eine verschiedene Anzahl von Messungen und Strahlen pro Messung.