Prof. Dr. Reinhard Klein
Alexander Schier

Universität Bonn
Institut für Informatik II
Wintersemester 2023/2024 13.10.2023

Übungen zu Angewandte Mathematik: Numerik - Blatt 1

Auf diesem Übungsblatt geht es darum die Grundlagen von Python und NumPy zu erlernen mit denen die Programmieraufgaben bearbeitet werden können. Die Abgabe der Lösungen zu Aufgaben 2-4 erfolgt per eCampus als Jupyter-Notebook bis zum 18.10.2023, 18:00 Uhr.

Die Übungsblätter werden regelmäßig Programmieraufgaben enthalten, mit denen die in der Vorlesung erlernten Verfahren implementiert und auf praktische Problemstellungen angewandt werden. Diese Aufgaben sollten in der Programmiersprache Python bearbeitet werden. Python ist eine Mehrzweckprogrammiersprache und die Software, die man zur Verwendung benötigt ist quelloffen und für jeden frei verfügbar. Durch die ebenfalls quelloffenen Bibliotheken NumPy, SciPy und matplotlib und mit Jupyter Notebooks als komfortable Oberfläche erhält Python ähnliche Funktionalitäten wie Matlab und ist somit gut für numerische Berechnungen und Visualisierungen geeignet.

Aufgabe 1 (Python installieren und lernen, *0 Punkte*)

Um numerische Berechnungen mit Python komfortabel durchzuführen, benötigt man die folgende Software.

Grundlagen

- Python Interpreter: Der Python Interpreter ist die Grundlage um in Python geschriebene Programme auszuführen und kann von python.org heruntergeladen werden, oder man benutzt Miniconda, wie unten beschrieben. Zum Testen der Lösungen verwenden wir Python 3. Python 2 ist offiziell nicht mehr unterstützt und wird von uns auch nicht zur Kontrolle der Lösungen eingesetzt.
- Softwarebibliotheken für Python:
  - **NumPy:** Stellt grundlegende Funktionalität zum Umgang mit Arrays und Matrizen zur Verfügung. Verfügbar unter NumPy.org.
  - **SciPy:** Stellt komplexere mathematische Funktionen zur Verfügung. Verfügbar unter SciPy.org.
  - **Matplotlib:** Stellt Funktionen zum Erstellen von Plots zur Verfügung. Verfügbar unter matplotlib.org.
  - **Jupyter:** Stellt eine Weboberfläche bereit, in welcher Textabschnitte und Codeabschnitte eingefügt werden können und Matplotlib Grafiken direkt inline angezeigt werden können. Verfügbar unter jupyter.org.

Für Windows ist das Paketverwaltungssystem Miniconda (`https://docs.conda.io/projects/miniconda/en/latest/`) empfehlenswert, mit dem die benötigten Bibliotheken durch die folgende Befehlszeile installiert werden können:

```
conda install numpy scipy matplotlib jupyter
```

Unter Linux sind die nötigen Libraries in den meisten Distributionen als vom System bereitgestellte Pakete vorhanden, und können zum Beispiel unter Ubuntu via:

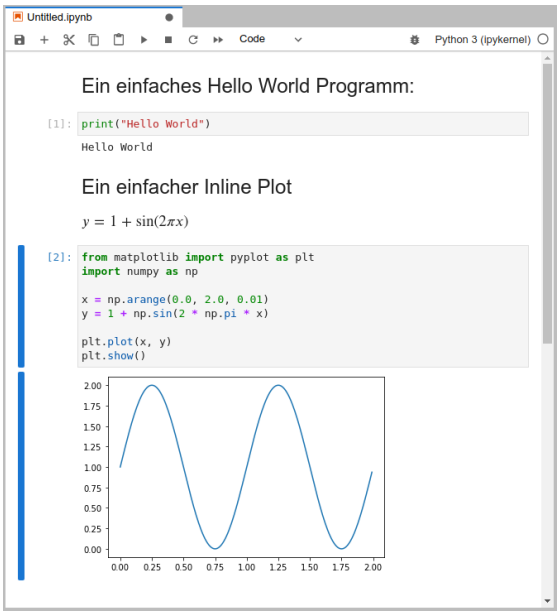
```
apt-get install python3-numpy python3-scipy python3-matplotlib jupyter-
notebook
```

installiert werden. Alternativ kann man `pip` oder ebenfalls (mini)conda verwenden.

Als interaktiver Python-Interpreter ist das Python-Paket `ipython` empfehlenswert, welches die Python-Konsole um Syntax-Highlighting und Tab-Vervollständigung erweitert.

Jupyter Notebooks

Jupyter Notebooks bieten eine Möglichkeit Code in einem einfach zu benutztendem Webinterface zu schreiben und in verschiedene Blöcke zu unterteilen, zwischen denen Textblöcke (mit Unterstützung für Markdown und LaTeX-Formeln) für Erklärungen und Kommentare eingefügt werden können. Die Ausgaben von `matplotlib` lassen sich im Notebook eingebettet anzeigen.



Die Abgaben der praktischen Übungen erfolgen als Jupyter-Notebook. Dazu muss die Notebook Datei (`.ipynb`) abgegeben werden und nicht nur die erzeugte Ausgabe, damit der Tutor den Code ausführen und nachvollziehen kann.

Python- und Numpy-Dokumentation

Machen Sie sich mit Python vertraut indem sie das Tutorial auf `https://docs.python.org/3/tutorial/` lesen. Machem Sie sich mit NumPy anhand der folgenden Kapitel in der offiziellen Hilfe vertraut:

- Array creation: `https://numpy.org/doc/stable/user/basics.creation.html`
- Indexing: `https://numpy.org/doc/stable/user/basics.indexing.html`
- Array manipulation routines:
  `https://numpy.org/doc/stable/reference/routines.array-manipulation.html`
- Random sampling: `https://numpy.org/doc/stable/reference/random/index.html`

Aufgabe 2 (Numpy-Grundlagen, $20 \times 0,5 = 10$ *Punkte*)

Erstellen Sie ein Jupyter-Notebook, welches für jede der nachfolgenden Aufgaben ein Beispiel mit Zahlen enthält. Dabei soll für jede Teilaufgabe mit Code jeweils eine Ausgabe (print) erfolgen, die folgende Form hat: Zuerst kommt der Buchstabe der Teilaufgabe, danach kommt ggf. die Matrix (oder Vektor) an der die Operationen ausgeführt werden sollen und dann die Ergebnisse der geforderten Operationen.

Alle Aufgaben sollen mit Numpy-Methoden und Array-Slicing statt Python-Schleifen gelöst werden.

a) Erstelle einen Markdown-Block mit einer Überschrift und einem Absatz mit den Namen der Teilnehmer der Abgabegruppe als Bulletpoints.
b) Füge eine LaTeX-Formel in den Markdown-Block ein.
c) Erstelle einen Python-Block mit einem Python-Kommentar
d) Erstelle einen 1×4 Zeilenvektor.
e) Erstelle einen 5×1 Spaltenvektor.
f) Erstelle eine Nullmatrix mit einer vorhandenen Funktion.
g) Gib die zweite Zeile einer 4×3 Matrix aus.
h) Gib die dritte Spalte einer 4×4 Matrix aus.
i) Transponiere eine Matrix.
j) Erzeuge zwei Matrizen gleicher Größe ($m \times m$). Multipliziere die Matrizen einmal elementweise und einmal mit der gewöhnlichen Matrixmultiplikation.
k) Füge zwei Matrizen jeweils unter Verwendung einer vorhandenen Funktion sowohl horizontal als auch vertikal zu einer neuen Matrix zusammen.
l) Gib die Größe bzw. Dimension (n, m) einer Matrix aus.
m) Verändere unter Verwendung einer vorhandenen Funktion die Struktur einer 8×7 Matrix so, dass eine 14×4 Matrix entsteht.
n) Vervielfache einen 3×1 Vektor so, dass eine 3×10000 Matrix entsteht.
o) Setze alle negativen Elemente einer Matrix auf 0.
p) Erzeuge einen Vektor, der jede 7. Zahl aus dem Bereich 1-100 enthält, beginnend mit 1.
q) Erzeuge einen Vektor mit 100 Einträgen. Setze anschließend jedes zweite Element des Vektors auf 0.
r) Erzeuge einen Vektor mit 100 Einträgen. Lösche anschließend jedes zweite Element des Vektors.
s) Erzeuge die 3×3 Identitätsmatrix.
t) Gib die Hauptdiagonale einer 5×5 Matrix aus. Benutze dazu eine Funktion, welche die Diagonale als Vektor zurückgibt.

Hinweis: Die Klasse `np.matrix` gilt als deprecated. Als Alternative können Numpy Arrays mit expliziter Matrixmultiplikation über `A.dot(B)` oder `A @ B` verwendet werden. Abgaben, die `np.matrix` verwenden werden von uns aber auch gewertet.

Hinweis zu Bonusaufgaben: Die Bonuspunkte zählen nicht zu den 50% Punkten, die erreicht werden müssen, aber die Bonusaufgaben sind genauso klausurrelevant wie die anderen Aufgaben.

Bonusaufgabe 3 (Effiziente Berechnungen mit Numpy, $3 + 3 = 6$ *Punkte*)

a) Erzeuge zwei Matrizen mit der Dimension 1000×3, welche mit Zufallszahlen gefüllt sind. Dabei können die Zeilen einer solchen Matrix als 1×3 Vektoren interpretiert werden. Berechne das

Skalarprodukt für alle Paare dieser Vektoren unter Verwendung von Python-Schleifen. Dazu muss über die Zeilen der 1000×3 Matrizen iteriert werden und das Skalarprodukt für die Vektoren, welche durch die aktuellen Matrixzeilen gegeben sind, berechnet werden.
Berechne weiterhin diese Skalarprodukte ohne die Verwendung von Schleifen. Vergleiche die Laufzeiten der beiden Varianten. (Hinweis: Zum Zeitmessen kann in Python das Modul `time` oder der Jupyter-Befehl `%timeit` verwendet werden). Was fällt dir auf, wenn du die Laufzeiten miteinander vergleichst?

b) In dieser Aufgabe ist folgendes Szenario gegeben. Wir wollen tausend 2×2 Matrizen $A^{(i)} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ invertieren.

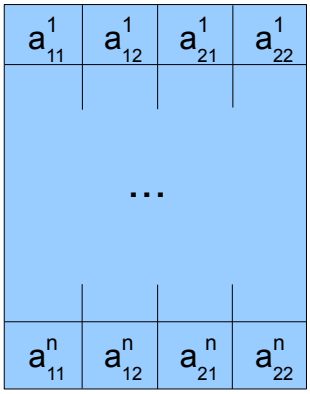


Abbildung 1: Speicherlayout der 2×2 Matrizen.

Die 2×2 Matrizen sind jeweils als Zeile repräsentiert (vgl. Abbildung 1). Die tiefgestellten Zahlen an den Matrixeinträgen stellen die Position des Eintrages in der 2×2 Matrix dar. Die hochgestellte Zahl gibt den Index der aktuellen Matrix an ($n = 1000$). Erzeuge eine 1000×4 Matrix mit zufälligen Einträgen, wobei jede Zeile wie in Abbildung 1 gezeigt, nun einer 2×2 Matrix entspricht. Zur Berechnung der Inversen der Matrizen soll das gegebene Speicherlayout der 2×2 Matrizen **nicht** verändert werden (d.h. eine Zeile des gegebenen Speicherlayouts soll **nicht** in ein 2×2 Array umgewandelt werden, womit dann die Inverse über vorhandene Funktionen berechnet wird). Mit Hilfe der Cramer'schen Regel für 2×2 Matrizen können die 1000 Matrizen invertiert werden. Berechne die Inversen für die erzeugten 2×2 Matrizen mit Numpy ohne dafür Schleifen zu verwenden.

Bonusaufgabe 4 (Matplotlib, $2 + 2 = 4$ *Punkte*)

In dieser Aufgabe sehen wir uns das Plotten mit Matplotlib und einen einfachen Polynomfit mit `numpy` an.

a) Scatter Plot einer linearen Funktion:
- Erzeuge 20 gleichmäßig verteilte X-Werte zwischen 0 und 10.
- Erzeuge 20 Y-Werte $y_i = 2x_i$.
- Addiere ein durch eine numpy-Funktion erzeugtes normalverteiltes Rauschen mit $\sigma = 1, \mu = 0$ auf die Y-Werte.
- Benutze eine numpy-Funktion um das Polynom $y = ax + b$ an die Daten zu fitten.
- Plotte mit matplotlib die X und Y Werte in einem Scatterplot und den Fit als Lineplot im gleichen Diagramm.

b) Scatter Plot einer quadratischen Funktion:

- Erzeuge 20 gleichmäßig verteilte X-Werte zwischen -10 und 10.
- Erzeuge 20 Y-Werte $y_i = x_i^2$.
- Addiere ein durch eine numpy-Funktion erzeugtes normalverteiltes Rauschen auf die Y-Werte mit $\sigma = 20, \mu = 0$.
- Benutze eine numpy-Funktion um das Polynom $y = ax^2 + bx + c$ an die Daten zu fitten.
- Plotte mit matplotlib die X und Y Werte in einem Scatterplot und den Fit als Lineplot im gleichen Diagramm.

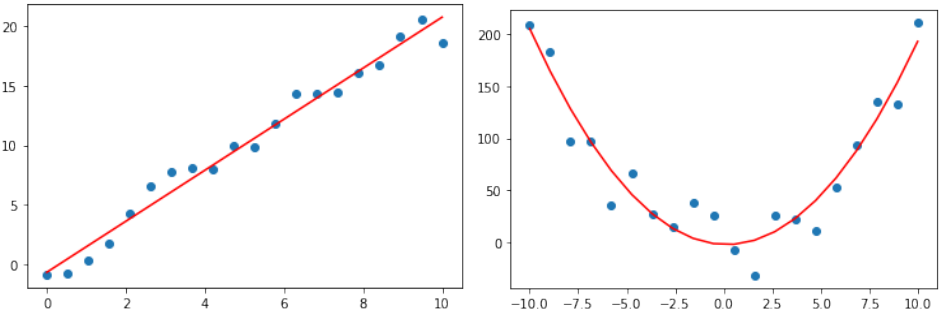


Abbildung 2: So könnte das Ergebnis aussehen

Später in der Vorlesung werden wir uns ansehen wie ein solches Fitting funktioniert. Hier sollen die in `numpy` vorhandenen Funktionen verwendet werden, welche diese Methoden implementieren.