Technische Universität Chemnitz
Fakultät für Mathematik

Prof. Dr. O. Ernst
J. Blechschmidt

8. Oktober 2018

Einführung in Data Science

Übung 1: Einführung in Python

Aufgabe 1: Einführung – Python

In dieser Übungsaufgabe sollst du den grundsätzlichen Umgang mit Python lernen. Der Workflow zum Initilisieren der Umgebung ist immer derselbe: Öffne dazu eine Konsole auf deinem Rechner. Anschließend kannst du mit dem Befehl

`source /LOCAL/Software/DataScience2018/setup_env`

eine Python-Umgebung aktivieren, die speziell für unsere Aufgaben auf den Rechnern im MRZ-Pool erstellt wurde. Mit dem Befehl `python --version` kannst du die aktuelle Version überprüfen, diese sollte in unserem Fall `Python 3.6.6` sein. Mit dem Befehl `which python` solltest du den Installationspfad überprüfen, er sollte

`/LOCAL/Software/DataScience2018/miniconda3/envs/DS2018/bin/python`

lauten. Wenn dem so ist, kannst du eine interaktive Python-shell mit dem Befehl

`ipython`

starten. Der Befehl `python` startet hingegen eine normale Python-shell, ohne Kolorierung und anderen netten Features. In dieser Aufgabe sollst du dich mit den wichtigsten Funktionalitäten von Python vertraut machen.

(a) Wir starten mit dem Befehl

```
a = [1, 2, 3]
```

Mit

```
type(a)
```

erfahren wir, dass die Variable `a` vom Typ “Liste” ist. Mit

```
a[1]
```

können wir auf das **zweite** Element der Liste `a` zugreifen. Im Gegensatz zu anderen Programmiersprachen wie MATLAB beginnt die Indizierung in Python mit `0`. Mittels

```
len(a)
```

findest du die Länge der Liste heraus, also die Anzahl der Elemente. Definiere dir eine zweite Liste, z.B.

```
b = [4, 5, 6]
```

und teste, was

```
a + b
```

ergibt. Was stellst du fest? Was passiert bei der Multiplikation einer Liste mit einem Skalar?

(b) Um herauszufinden, was wir mit einer Liste alles machen können, kannst du `a.` tippen, und anschließend die Tabulator-Taste drücken. Es erscheint eine Auflistung von verschiedenen Methoden der Klasse, z.B. kannst du mittels

```
a.append(1)
```

eine `1` an deine Liste `a` anhängen. Wenn du nicht weißt, wofür eine Funktion da ist, kannst du immer ein Fragezeichen vor die Funktion setzen, z.B. `? a.append` Abhängig davon, wie gut die Funktion/Methode kommentiert ist, ist diese Hilfe mehr oder weniger hilfreich. Jetzt wollen wir die Liste `a` absteigend sortieren. Mit

```
? a.sort
```

erfahren wir, dass die Methode zwei optionale Inputs hat, zum einen `key` mit dem Standardwert `None` und zum anderen `reverse` mit dem Standardwert `False`. Der Hilfe-Dialog kann durch Drücken von 'q' beendet werden.

Da wir die Liste absteigend sortieren wollen, müssen wir also `reverse` auf `True` setzen. Dies können wir mittels

```
a.sort(reverse = True)
```

realisieren.

(c) Füge die Zahl `5` an die zweite Stelle von `a` ein. Suche dazu die richtige Methode heraus und mache dich mittels `?` mit dieser vertraut. Abschließend sollte `a` gleich `[3, 5, 2, 1, 1]` sein.

Es sollte nun klar sein, dass das reine Python ohne Zusatzmodule für mathematische Berechnungen nur bedingt geeignet ist. Pakete erweitern die Funktionalitäten von Python extrem, und sollen in den nächsten Aufgaben diskutiert werden. Grundsätzlich sollte klar sein, dass bei den meisten Problemen die Dokumentationen der Funktionen, Methoden und Klassen extrem hilfreich sind.

Aufgabe 2: Einführung – Numpy

Wie bereits angedeutet, lebt Python von Paketen. Das wichtigste Paket für wissenschaftliches Rechnen stellt `numpy` dar

```
import numpy as np
```

Bitte mache dich mit dem Tutorial unter
https://docs.scipy.org/doc/numpy/user/quickstart.html
vertraut. Für alle, die vorher viel mit MATLAB gearbeitet haben, sind hier die wichtigsten Unterschiede aufgelistet:
https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html.

Hinweis: Es ist nicht üblich, alle Funktionen eines Pakets in den normalen Namespace mittels `from numpy import *` zu importieren. Stattdessen verwendet man `import numpy` bzw. `import numpy as np` und kann dann auf die Funktionen des Pakets durch Voranstellen von `numpy.` bzw. `np.` zugreifen, z.B. `numpy.ndarray` bzw. `np.ndarray` zugreifen.

(a) Erstelle die Matrix

$$A = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

durch direkte Eingabe mittels `np.matrix`.

(b) Erstelle die Matrix

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

mit den Befehlen `np.arange`, `np.reshape` und `np.asmatrix`. Achte auf den Unterschied zwischen Matrizen und Arrays.

(c) Berechne

$$C = A\,B^\top$$

mit `A * B.T` und mit `A.dot(B.T)`. Was ist der Unterschied zwischen `C**2` und `np.power(C,2)`?

(d) Erstelle A und B als Arrays anstatt Matrizen. Was stellst du fest, wenn du $D = A\,B^\top$ berechnen willst? Was beobachtest du bei `D**2` und `np.power(D,2)`?

(e) Mit

```
np.random.randint(5, size=(2, 4))
```

lässt sich ein 2×4 Array mit zufälligen Ganzzahlen im Bereich $0, \ldots, 4$ erstellen. Erstelle ein 40×4 Array `X` mit standard-normalverteilten (Pseudo)-Zufallszahlen mit dem Befehl `np.random.randn`. Berechne mit den Befehlen `np.mean` und `np.var` die Mittelwerte und Varianzen der Spalten von `X`. Da es sich hierbei um (Pseudo)-Zufallszahlen handelt, wird das Ergebnis bei jedem Durchlauf anders aussehen.

Um Algorithmen zu testen ist es vorteilhaft, zu Beginn den Seed des Zufallszahlengenerators festzusetzen, zum Beispiel mit

```
np.random.seed(0)
```

Setze den Seed auf `0`, und führe die obige Aufgabe nochmals aus. Vergleiche deine Werte mit denen deiner Kollegen.

Aufgabe 3: Einführung – Plots

In dieser Aufgabe sollst du dich mit den wichtigsten Plot-Routinen vertraut machen. Das Paket `matplotlib` ist das populärste Paket zur grafischen Darstellung in Python. Vieles funktioniert ähnlich wie in MATLAB. Mit dem Befehl

```
import matplotlib.pyplot as plt
```

lässt sich `pyplot` importieren, mit dem wir uns in diesem Intro beschäftigen wollen. Für genaue Informationen findest du unter
https://realpython.com/python-matplotlib-guide/
ein gutes Tutorial.

(a) Erstelle mit dem Befehl

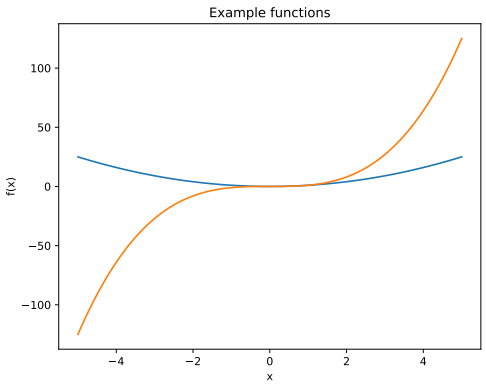
```
x = np.linspace(-5,5,101)
```

einen Vektor `x` (hier als Array), der das Interval $[-5, 5]$ äquidistant in 101 Zahlen unterteilt. Berechne jetzt $y = x^2$ sowie $z = x^3$. Mit den Befehlen

```
plt.plot(x,y)
plt.show()
```

kannst du einen einfachen Plot erzeugen. Im Gegensatz zu MATLAB erzeugt Python nicht für jeden Plot ein neues `figure`-Object, sondern plottet alles in die aktuelle `figure`, bis `plt.show()` aufgerufen wird.

Versuche nun, sowohl die Funktion $y(x)$ als auch $z(x)$ in eine Figure zu plotten. Ändere mit `plt.xlabel`, `plt.ylabel` und `plt.title` die Bezeichnungen der x-Achse, der y-Achse sowie den Titel, **bevor** du `plt.show()` aufrufst. Das Ergebnis sollte so aussehen:



(b) Jetzt wollen wir die 2-dimensionale Funktion

$$f(x, y) = e^{-(x^2+y^2)}$$

darstellen. Führe den folgenden Code aus. Erkläre, was in jeder Zeile passiert.

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
z = np.linspace(-3, 3, 201)
x, y = np.meshgrid(z, z)
f = np.exp(-(np.power(x,2) + np.power(y,2)))
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(x,y,f, cmap=cm.coolwarm)
plt.show()
```

(c) Erstelle einen Konturplot für die Funktion

$$g(x, y) = \sin(3\,x)\,\cos(2\,y)$$

mit `plt.contour` oder `plt.contourf`.

Bemerkung: Mit `plt.ion()` bzw. `plt.ioff()` können Grafiken interaktiv gestaltet werden, das heißt man sieht direkt, was man plottet, also vor einem Aufruf von `plt.show()`.