11_Uebungsaufgaben_Loesungen

0.1 Vorschläge für Übungsaufgaben

Die folgenden Übungsaufgaben sind ausführlicher als die unterwegs vorgeschlagenen Mini-Übungen. Das Ziel ist es hier, nicht nur einzelne Themen direkt auszuprobieren, sondern vielmehr verschiedene Themen zu kombinieren, um so ein verknüpfendes Verständnis der Lerninhalte zu erreichen.

0.1.1 Vektorrechnung

In dieser Aufgabe sollen Sie ein Modul mit dem Namen vector_calculus implementieren, welches das Rechnen mit n-dimensionalen Vektoren ermöglicht ($n \geq 2$), hierbei soll ein Vektor durch eine Liste gegeben sein, d.h. vector = [x_1, ...,x_n]. Ihr Modul soll folgende **dokumentierte** Funktionen enthalten:

- vnorm(v): Zur Berechnung der euklidischen Norm $\|v\|_2$ eines Vektors v.
- vsum(vectors): Berechnet die Summe aller übergebenen Vektoren.
- vdotproduct(v1, v2): Berechnet das Skalarprodukt $\langle v1, v2 \rangle_2$ zweier Vektoren.
- vangle(v1, v2): Berechnet den Winkel zwischen zwei Vektoren:

$$\theta(v1, v2) = \left(\frac{\langle v1, v2 \rangle_2}{\|v1\|_2 \|v2\|_2} \right)$$

Stellen Sie sicher, dass die übergebenen Tupel der drei letzten Funktionen die selbe Größe haben, ansonsten soll das Programm mit einer Fehlermeldung abgebrochen werden. Behandeln Sie mögliche logische Fehler, wie Division durch Null **bevor** diese auftreten. Die Ergebnisse der Funktionen sollen als float bzw. Liste zurück gegeben werden.

Hinweis: Es ist sinnvoll, diese und die nächste Aufgabe "simultan" zu bearbeiten.

[1]:
```
"""
Modul zur Vektorrechnung, primitiver Nachbau von NumPy
"""

import math

def check_same_length(v,w):
    """Ueberprueft, ob die beiden Tupel dieselbe Länge haben"""
    if len(v) != len(w):
```

```
        print("Die Vektoren haben nicht dieselbe Länge. Abbruch.")
        quit()

def vnorm(v):
    """Berechnet die Euklid-Norm des übergebenen Vektors"""
    squares = [x**2 for x in v]
    return math.sqrt(sum(squares))

def vsum(*vectors):
    """Addiert alle übergebenen Vektoren komponentenweise"""
    for i in range(len(vectors)-1):
        check_same_length(vectors[i],vectors[i+1])
    result = [0 for i in range(len(vectors[0]))] # Wir brauchen temporär etwas␣
 ↪mutables
    for v in vectors:
        for i in range(len(v)):
            result[i] += v[i]
    return result

def vdotproduct(v,w):
    """Berechnet das Skalarprodukt der beiden Vektoren"""
    check_same_length(v,w)
    sums = [a*b for a,b in zip(v,w)]
    return sum(sums)

def vangle(v,w):
    """Berechnet den Winkel zwischen den beiden Vektoren"""
    check_same_length(v,w)
    v_norm = vnorm(v)
    w_norm = vnorm(w)
    if math.isclose(v_norm,0) or math.isclose(w_norm,0):
        print("Winkel mit dem Nullvektor gibt es nicht")
        quit()
    return math.cos(vdotproduct(v,w)/(v_norm*w_norm))
```

0.1.2 Vektorrechnung als Modul

Speichern Sie die Bearbeitung der vorherigen Aufgabe in einer Datei namens Vektorrechung.py und speichern Sie sie an einem Ort, der bei der Bearbeitung dieser Aufgabe sichtbar ist.

Testen Sie dann dieses Modul ausführlich mit einem neuen Code, der das Modul importiert. Implementieren Sie dazu einen Zugriff auf die Hilfe-Funktion, um Ihre Dokumentation zu "testen", und implementieren Sie für jede Funktion mindestens einen funktionierenden Fall, und einen Fall, der eine von ihnen generierte Fehlermeldung auslöst.

Hinweis: Wenn Sie nur Jupyter Notebooks verwenden, dann schreiben Sie das Modul in einen Codeblock, die Tests in einen anderen, und erinnern Sie sich, dass die Tests nur dann sinnvoll funktionieren, wenn Sie die "Funktionensammlung" (ein Modul ist es ja nicht) vorher ausgeführt

haben.

[3]:
```
# Test der Funktion check_same_length()
#v = (1,2,3,4)
#w = (1,2,3)
#check_same_length(v,w)

# Test der Funktion vnorm
v = [1,2,3,4]
print("||v||_2 = ",vnorm(v))

# Test der Funktion vsum
#v1 = [1,2,3]
#v2 = [1,2,3,4]
#vsum(v1,v2)
v1 = [1,2,3,4]
v2 = [5,6,7,8]
print(vsum(v1,v2))

# Test der Funktion vdotproduct
# kein Test von check_same_length() mehr nötig, Code ist identisch
v1 = (1,2,3,4)
v2 = (5,6,7,8)
print(vdotproduct(v1,v2))

# Test der Funktion vangle
# kein Test von check_same_length() mehr nötig, Code ist identisch
#print(vangle(v,[0,0,0,0]))
print(vangle(v1,v2))
```

```
||v||_2 =  5.477225575051661
[6, 8, 10, 12]
70
0.5662362927267999
```

0.2 Impressum

0.2.1 Programmierkurs Python, Dominik Göddeke https://www.ians.uni-stuttgart.de, Universität Stuttgart

Version vom April 2023

Gefördert durch die Stiftung Innovation in der Hochschullehre. (alle Rechte am Logo vorbehalten)

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie.

[ ]: