03e_NumPy_Miniuebungen_Loesungen

0.1 Mini-Aufgaben zur Überprüfung des Verständnis: Arbeiten mit ndarrays

Erzeugen Sie für ein beliebiges n die Hilbertmatrix $H \in \mathbb{R}^{n \times n}$, deren Einträge definiert sind über

$$a_{ij} = \frac{1}{i+j+1} \quad \text{für } i, j = 0, \ldots, n-1.$$

Beachten Sie dabei, dass die Nullindizierung bereits in der Definition steckt, der Eintrag $a_{0,0}$ ist bspw. 1. Es ist Ihnen überlassen, ob Sie dazu eine List Comprehension verwenden und das Ergebnis in ein ndarray umwandeln, oder ob Sie eine Nullmatrix passender Dimension erzeugen und danach über die Matrix iterieren, um die Einträge zu erzeugen.

[3]:
```
import numpy as np

n = 5

# Variante 1
H = [1/(i+j+1) for i in range(n) for j in range(n)]
H = np.array(H)
H.shape = (n,n)
print(H)

# Variante 2
H = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        H[i][j] = 1/(i+j+1)
print(H)
```

```
[[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125      0.11111111]]
[[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125      0.11111111]]
```

Programmieren Sie die Regel von Sarrus nach und testen Sie Ihre Implementierung vernünftig für 3×3 Matrizen. Wichtig: Sie dürfen sich hier explizit auf 3×3 Matrizen beschränken. Hinweis: np.linalg.det(A) ist eine geeignete Validierung.

[21]:
```
import numpy as np

def sarrus(A):
    B = np.zeros((3,5))
    B[:,0] = A[:,0]
    B[:,1] = A[:,1]
    B[:,2] = A[:,2]
    B[:,3] = A[:,0]
    B[:,4] = A[:,1]
    retval = 0
    for i in range(3):
        retval += B[i,i]
        retval += B[i,i+1]
        retval += B[i,i+2]
    for i in range(2,-1,-1):
        retval -= B[0,i]
        retval -= B[1,i+1]
        retval -= B[2,i+2]

    return retval

A = np.arange(1,10).reshape((3,3))
print(sarrus(A))
print(np.linalg.det(A))
```

```
0.0
0.0
```

Eine Tücke bei der Implementierung vektorisierter Funktionen sind Fallunterscheidungen, hier am Beispiel der Heaviside-Funktion:

$$f(x) := \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Diese Funktion lässt sich wie gewohnt implementieren, allerdings nicht wie die bisherigen Funktionen vektorisieren. Informieren Sie sich in der NumPy-Dokumentation über die Funktion np.where() und verwenden Sie diese um heaviside(x) vektorisiert zu implementieren.

[23]:
```
import numpy as np

def heaviside(x):
    # Kurzform der if-Anweisung:
    return 1*(x>=0)

def heaviside_vec(x):
```

```
    return np.where(x<0,0,1)

x = np.linspace(-1,1,10)
print(heaviside_vec(x))
```

```
[0 0 0 0 0 1 1 1 1 1]
```

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.

[ ]: