Python – Übung 09

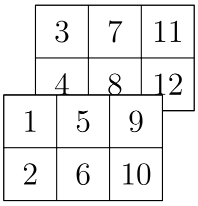
1 Mehrdimensionale Arrays

Gegeben seien folgende drei Vektoren: $\boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}, \quad \boldsymbol{z} = \begin{bmatrix} 9 \\ 10 \\ 11 \\ 12 \end{bmatrix}.$

☞ Erzeugen Sie aus den gegebenen Vektoren die nachfolgend gezeigte Vektoren und Matrizen, indem Sie die passenden NumPy-Methoden verwenden.

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \qquad \boldsymbol{c} = \begin{bmatrix} 1 & 5 & 9 & 2 & 6 & 10 & 3 & 7 & 11 & 4 & 8 & 12 \end{bmatrix}$$

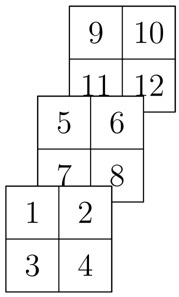
$$\boldsymbol{d} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$$

$\boldsymbol{E} =$ 

$=$

```
[[[ 1 5 9]
  [ 2 6 10]]

 [[ 3 7 11]
  [ 4 8 12]]]
```

$\boldsymbol{F} =$ 

$=$

```
[[[ 1 2]
  [ 3 4]]

 [[ 5 6]
  [ 7 8]]

 [[ 9 10]
  [11 12]]]
```

☞ Überlegen Sie sich was bei den folgenden Codesequenzen die jeweiligen Ausgaben sind.

```
g = np.array([1, 2, 3, 4, 5])
h = g[1:3]
h[0] = 200
print(g[1])
```

200

```
J = np.full((2, 3), 5)
k = J.ravel()
k[-1] = 0
print(k)
print(J)
```

J: 5 5 5
5 5 0

k: 5 5 5 5 5 0

```
L = np.arange(0, 10).reshape(5, 2)
M = np.tile(L, (1, 3))
print(M)
```

0 1 0 1 0 1
2 3 2 3 2 3
4 5 4 5 4 5
6 7 6 7 6 7
8 9 8 9 8 9

```
n = np.r_[np.linspace(0, 1, 3), 20:27]
print(n)
```

0 0,5 1 20 21 22 23 24 25 26

```
P = np.c_[0:4, np.eye(4)]
print(P)
```

0 1 0 0 0
1 0 1 0 0
2 0 0 1 0
3 0 0 0 1

2 Parabel-Koeffizienten bestimmen

Eine Parabel kann als quadratische Funktion beschrieben werden

$$y = ax^2 + bx + c \; . \tag{1}$$

Von einer unbekannten Parabel sind nur drei Punkte bekannt:

$$P_1 = (x_1, y_1) = (-1, 6) \; , \tag{2}$$
$$P_2 = (x_2, y_2) = (2, 0) \text{ und} \tag{3}$$
$$P_3 = (x_3, y_3) = (3, 4) \; . \tag{4}$$

Wie in Abb. 1 gezeigt, werden nun die Koeffizienten a, b und c gesucht, damit die Funktion in (1) mit den angegebenen Punkten übereinstimmt.

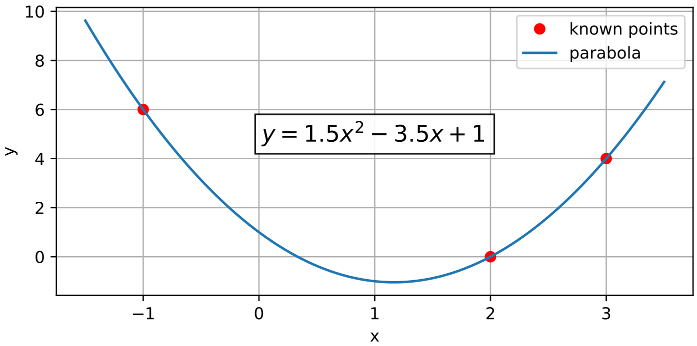


Abbildung 1: Quadratische Funktion mit den passenden Koeffizienten.

Das Gleichungssystem sieht wie folgt aus

$$ax_1^2 + bx_1 + c = y_1 \tag{5}$$
$$ax_2^2 + bx_2 + c = y_2 \tag{6}$$
$$ax_3^2 + bx_3 + c = y_3 \tag{7}$$

oder als Matrizengleichung

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{y} \; , \tag{8}$$

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \; . \tag{9}$$

Um die Gleichung zu lösen, muss von links her mit der Inversen $\boldsymbol{M}^{-1}$ multipliziert werden:

$$\boldsymbol{M}^{-1}\boldsymbol{y} = \boldsymbol{x} \; , \tag{10}$$

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \; . \tag{11}$$

☞ Schreiben Sie ein Programm, welches die Matrizengleichung (11) löst.

Hinweis: Benutzen Sie `np.column_stack()`[1] oder `np.c_[]`[2], um die Matrix zu bauen. Benutzen Sie `np.linalg.solve()`, um die Gleichung zu lösen.

[1] https://docs.scipy.org/doc/numpy/reference/generated/numpy.column_stack.html
[2] https://docs.scipy.org/doc/numpy/reference/generated/numpy.c_.html

3 Projektion berechnen

Mit Hilfe der nachfolgend gezeigten Formel ist es möglich, die Orthogonalprojektion $P_g(\vec{x})$ eines Punktes $\vec{x}$ auf eine Gerade g mit Stützvektor $\vec{r}_0$ und Richtungsvektor $\vec{u}$ zu bestimmen:

$$P_g(\vec{x}) = \vec{r}_0 + \frac{(\vec{x} - \vec{r}_0) \cdot \vec{u}}{\vec{u} \cdot \vec{u}} \vec{u} \, . \tag{12}$$

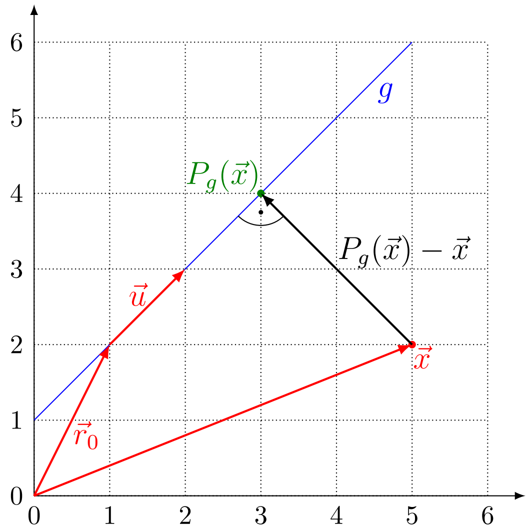


Abbildung 2: Orthogonalprojektion $P_g(\vec{x})$.

☞ Implementieren Sie die Funktion `projection(x, r0, u)`, welche die Orthogonalprojektion auf Grund der gegebenen Vektoren rechnet und diese als NumPy-Array zurückgibt.

```
>>> projection(x=[5, 2], r0=[1, 2], u=[1, 1])
array([3., 4.])
```

4 Matrizen potenzieren

Die Berechnung der n-ten Potenz $\boldsymbol{A}^n$ einer allgemeinen Matrix $\boldsymbol{A}$ kann – je nach Form der gegebenen Matrix – sehr rechen- und zeitaufwändig sein. Um diesen Aufwand zu umgehen, kann die Matrix $\mathbf{A}$ mit einer geeigneten Basistransformation diagonalisiert werden. Eine Matrix in Diagonalform lässt sich ganz einfach potenzieren und mit Hilfe der Transformationsmatrix $\boldsymbol{T}$ wieder rücktransformieren. Ganz allgemein gilt:

$$\boldsymbol{A}^n = \boldsymbol{T}\boldsymbol{D}^n\boldsymbol{T}^{-1} \, . \tag{13}$$

Zur Bestimmung der beiden Matrizen $\boldsymbol{D}$ und $\boldsymbol{T}$ benötigt man die Eigenwerte $\lambda_1, \lambda_2 \dots \lambda_n$ und Eigenvektoren $\boldsymbol{v}_1, \boldsymbol{v}_2 \dots \boldsymbol{v}_n$ der gegebenen Matrix $\boldsymbol{A}$. Sind die Eigenwerte und die dazugehörigen Eigenvektoren bestimmbar, so ist die Matrix $\boldsymbol{A}$ diagonalisierbar und die beiden Matrizen $\boldsymbol{T}$

und $\boldsymbol{D}$ können wie folgt aufgebaut werden. Die Matrix $\boldsymbol{D}$ ist eine Diagonalmatrix mit den Eigenwerten $\lambda_1, \lambda_2 \ldots \lambda_n$ auf ihrer Hauptdiagonalen:

$$\boldsymbol{D} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \tag{14}$$

Die Matrix $\boldsymbol{T}$ ist eine Matrix mit den jeweiligen Eigenvektoren $\boldsymbol{v}_1, \boldsymbol{v}_2 \ldots \boldsymbol{v}_n$ als Spalten:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \ldots & \boldsymbol{v}_n \end{bmatrix} \tag{15}$$

☞ Implementieren Sie die Funktion `power(A, n)`, welche die `n`-te Potenz der gegebenen Matrix `A` berechnet und zurückgibt.

```
>>> A1 = np.array([[1, -1],
                   [-3, -1]])
>>> A2 = np.array([[0, 1, 1],
                   [1, 0, 1],
                   [1, 1, 0]])
>>> power(A1, 5)
array([[ 16., -16.],
       [-48., -16.]])
>>> power(A2, 3)
array([[2., 3., 3.],
       [3., 2., 3.],
       [3., 3., 2.]])
```