Python – Testat 2

1 Aufgabenstellung und Bewertungskriterien

☞ Implementieren Sie die drei Funktionen `e_field()`, `voltage()` und `gauss_law()`, wie unten im Detail beschrieben, und liefern Sie diese als einzelne Datei mit dem Namen `testat2_<PID>.txt` der Dozierenden bis spätestens am 19. Mai 2020 um 12 Uhr per Email (smalacar@hsr.ch) ab.

Bewertungskriterien: Für jedes Detail, das richtig implementiert wird, gibt es Punkte. Rechts neben jeder Detailbeschreibung, wird die Zahl der möglichen Punkte angegeben. In diesem Testat 2 gibt es total 8 Punkte. Die unten **angegebenen Funktions-, und Parameter-Namen müssen exakt übernommen** werden, sonst werden für den betroffenen Teil keine Punkte vergeben. Es wird empfohlen, den eigenen Code ausgiebig zu testen. Die Beispiele in den grauen Listing-Boxen eignen sich für einen ersten Vergleich.

Prüfungszulassung: Um an der Prüfung zugelassen zu werden, muss mindestens 40% der erreichbaren Punkte pro Testat 1 & 2 erreicht werden.

2 Allgemeine Bewertungskriterien

- Alle Funktionen besitzen aussagekräftige Docstrings. (0.5 P)
- Python-Codestil entspricht den PEP8-Empfehlungen. 0.25 P Abzug pro Stilfehler[1]. (0.5 P)

2.1 Nach PEP8-Stilfehlern suchen

Das Flake8-Programm[2] kann wie folgt installiert werden:

1. "Anaconda Prompt" als Administrator öffnen und folgende Zeile ausführen:
2. `conda install -c anaconda flake8`

Überprüfen Sie ihr Python-Code nach Stilfehlern, indem Sie das Flake8-Programm[3] wie folgt im "Anaconda Prompt" aufrufen:

```
flake8 python_datei.py
```

[1] der Codestil wird mit der Flake8-Software geprüft, `http://flake8.pycqa.org/en/latest/`

[2] `http://flake8.pycqa.org/`

[3] Die Anaconda Prompt muss beim Filepfad des zu prüfenden Pythonskripts geöffnet sein. Zum Filepfad navigieren Sie mit dem Befehl `cd <path>`

3 Funktion `e_field(q_values, q_locations, points)`

In einem Raum sind – wie in Abb. 1 dargestellt – verschiedene Punktladungen Q_m verteilt und an jedem Punkt $\vec{P}_n$ kann die elektrische Feldstärke $\vec{E}_n$ ermittelt werden.

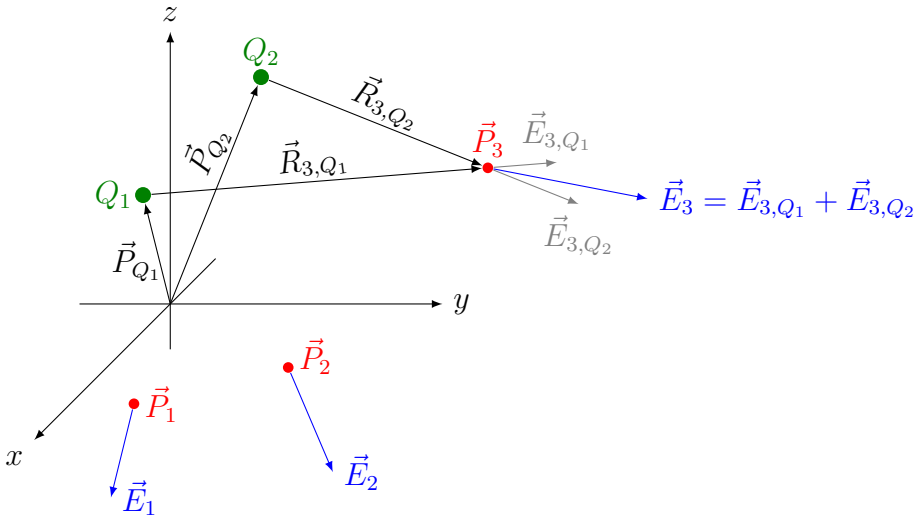

Abbildung 1: Die Ladungen Q_m, die Punkte $\vec{P}_n$ und die Feldstärkevektoren $\vec{E}_n$ im dreidimensionalen Raum.

Der elektrische Feldstärkevektor $\vec{E}_n$ an einem Punkt $\vec{P}_n$ kann als Superposition der Feldstärkevektoren $\vec{E}_{n,Q_m}$ berechnet werden

$$\vec{E}_n = \sum_{m=1}^{M} \vec{E}_{n,Q_m} \,. \tag{1}$$

Der elektrische Feldstärkevektor $\vec{E}_{n,Q_m}$, welcher von einer Punktladung Q_m an einem bestimmten Ort (gegeben durch den Distanzvektor $\vec{R}$) ausgeht, kann wie folgt berechnet werden:

$$\vec{E}_{n,Q_m} = \begin{bmatrix} E_{n,Q_m,x} \\ E_{n,Q_m,y} \\ E_{n,Q_m,z} \end{bmatrix} = \frac{Q_m}{4\pi\varepsilon_0} \frac{\vec{R}_{n,Q_m}}{\left\|\vec{R}_{n,Q_m}\right\|^3} = \frac{Q_m}{4\pi\varepsilon_0 \left\|\vec{R}_{n,Q_m}\right\|^3} \begin{bmatrix} R_{n,Q_m,x} \\ R_{n,Q_m,y} \\ R_{n,Q_m,z} \end{bmatrix} , \tag{2}$$

wobei $\|.\|$ der Betrag des Vektors bezeichnet (welcher mit der `np.linalg.norm()`-Funktion berechnet werden kann), $\varepsilon_0 \approx 8.854 \cdot 10^{-12}$ die elektrische Feldkonstante ist und der Distanzvektor $\vec{R}_{n,Q_m}$ wie folgt berechnet werden kann:

$$\vec{R}_{n,Q_m} = \begin{bmatrix} R_{n,Q_m,x} \\ R_{n,Q_m,y} \\ R_{n,Q_m,z} \end{bmatrix} = \vec{P}_n - \vec{P}_{Q_m} = \begin{bmatrix} P_{n,x} - P_{Q_m,x} \\ P_{n,y} - P_{Q_m,y} \\ P_{n,z} - P_{Q_m,z} \end{bmatrix} \tag{3}$$

Hinweis: Den Wert der elektrischen Feldkonstante können Sie mit

```
>>> from scipy.constants import epsilon_0
```

importieren und dann über die Variable `epsilon_0` direkt nutzen.

Es soll eine Funktion `e_field(q_values, q_locations, points)` implementiert werden, welche die elektrischen Feldstärkevektoren $\vec{E}_n$ aller im Raum verteilten Ladungen anhand der gegebenen Informationen (Q_m, $\vec{P}_{Q_m}$) an jedem Punkt im dreidimensionalen Raum berechnen kann. Die Parameter und Rückgabewerte der Funktion sind in Tab. 1 und Tab. 2 aufgelistet.

Tabelle 1: Parameterliste der `e_field()`-Funktion.

Parameter	**Datentyp**	**Beschreibung**
`q_values`	`int, float` oder `list` oder 1D `np.array`	Hiermit werden die Ladungsmengen Q_m aller im Raum verteilter Punktladungen angegeben.
`q_locations`	`list` oder 2D `list` oder 1D `np.array` oder 2D `np.array`	Hiermit werden die kartesischen Koordinaten $\vec{P}_{Q_m}$ aller im Raum verteilter Punktladungen Q_m angegeben.
`points`	`list` oder 2D `list` oder 1D `np.array` oder 2D `np.array`	Hiermit werden die kartesischen Koordinaten der Punkte $\vec{P}_n$ im Raum angegeben, an welchen die elektrischen Feldstärkevektoren $\vec{E}_n$ bestimmt werden sollen.

Tabelle 2: Rückgabewerte der `e_field()`-Funktion.

Wert	**Datentyp**	**Beschreibung**
$\vec{E}_n$	1D `np.array` oder 2D `np.array`	Hiermit werden die elektrischen Feldstärkevektoren $\vec{E}_n$ an den angefragten Punkten $\vec{P}_n$ zurückgegeben.

☞ Die Funktion berechnet den elektrischen Feldstärkevektor $\vec{E}_1$ am einzigen Punkt $\vec{P}_1$, der durch die einzige Ladung Q_1 hervorgerufen wird. Die Parameter können als Liste oder 1D `np.array` angegeben werden. Der Rückgabewert ist ein 1D `np.array`. (1 P)

```
>>> q = 2e-7
>>> q_loc = [1, 2, 3]
>>> p = [4, 1, -9]
>>> e_field(q_values=q, q_locations=q_loc, points=p)
array([  2.82170479,  -0.94056826, -11.28681915])
```

```
>>> q = [2e-7]
>>> q_loc = np.array([1, 2, 3])
>>> p = np.array([4, 1, -9])
>>> e_field(q_values=q, q_locations=q_loc, points=p)
array([  2.82170479,  -0.94056826, -11.28681915])
```

☞ Die Parameter können auch mehrere Punkte $\vec{P}_n$, Ladungen Q_m und Ladungspositionen $\vec{P}_{Q_m}$ enthalten. Die Parameter können als 2D Listen oder 2D `np.array` angegeben werden. Der Rückgabewert ist in diesem Fall ein 2D `np.array`. (1 P)
Hinweis: Die Vektorenwerte sind entlang der zweiten Dimension angeordnet, d.h. `[[x1, y1, z1], [x2, y2, z2], ...]`.

```
>>> q = [2e-7, -5e-7]
>>> q_loc = [[1, 2, 3], [3, 2, 1]]
>>> p = [[4, 1, -9], [1, 8, 0]]
>>> e_field(q_values=q, q_locations=q_loc, points=p)
array([[ -1.54055125,   3.42168778,  32.33574124],
       [ 34.2346286 , -66.97628271,  -0.74648725]])
```

```
>>> q = np.array([2e-7, -5e-7])
>>> q_loc = np.array([[1, 2, 3], [3, 2, 1]])
>>> p = np.array([[4, 1, -9], [1, 8, 0]])
>>> e_field(q_values=q, q_locations=q_loc, points=p)
array([[ -1.54055125,   3.42168778,  32.33574124],
       [ 34.2346286 , -66.97628271,  -0.74648725]])
```

4 Funktion `voltage(e_vectors, path)`

Wie in Abb. 2 dargestellt, wurden an mehreren Punkten $\vec{P}_n$ entlang einem Pfad im Raum die elektrischen Feldstärkevektoren $\vec{E}_n$ ermittelt.

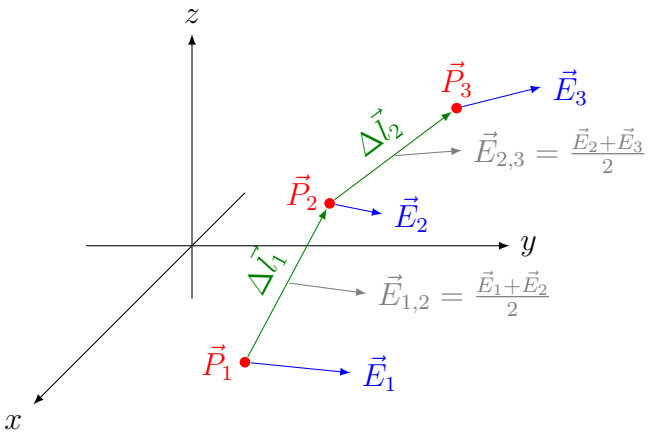

Abbildung 2: Die Feldstärkevektoren $\vec{E}$ entlang einem Pfad im dreidimensionalen Raum.

Mit Hilfe dieser Information kann man die elektrische Spannung (oder auch Potentialdifferenz genannt) zwischen dem Startpunkt und irgend einem anderen Punkt entlang des Pfades berechnen. Die Spannung U_{AB} zwischen zwei Punkten A und B kann als Linienintegral der elektrischen Feldstärke entlang einem Pfad von A nach B berechnet werden:

$$U_{AB} = \int_A^B \vec{E} \cdot d\vec{l}, \tag{4}$$

wobei $d\vec{l}$ ein infinitesimal kleiner Pfadabschnitt ist. In diesem Fall kennt man die elektrischen Feldstärkevektoren $\vec{E}_n$ aber nur an bestimmten Punkten $\vec{P}_n$ entlang des Pfades, d.h. $d\vec{l}$ wird durch die Differenzvektoren

$$\Delta\vec{l}_n = \vec{P}_{n+1} - \vec{P}_n\,. \tag{5}$$

ersetzt und das Integral wird mit einer Summe approximiert:

$$U_N \approx \sum_{n=1}^{N-1} \vec{E}_{n,n+1} \cdot \Delta\vec{l}_n = \sum_{n=1}^{N-1} \frac{\vec{E}_n + \vec{E}_{n+1}}{2} \cdot \left(\vec{P}_{n+1} - \vec{P}_n\right), \tag{6}$$

wobei $\vec{E}_{n,n+1}$ der Mittelwert der Feldstärkevektoren vor und nach dem Pfadabschnitt $\Delta\vec{l}_n$ ist.

Es soll die Funktion `voltage(e_vectors, path)` implementiert werden, welche die elektrische Spannung entlang dem vorgegebenen Pfad berechnet. Die Parameter und Rückgabewerte der Funktion sind in Tab. 3 und Tab. 4 aufgelistet.

Tabelle 3: Parameterliste der `voltage()`-Funktion.

Parameter	**Datentyp**	**Beschreibung**
`e_vectors`	2D `list` oder 2D `np.array`	Hiermit werden die elektrischen Feldstärkevektoren $\vec{E}_n$ an den Punkten $\vec{P}_n$ angegeben.
`path`	2D `list` oder 2D `np.array`	Hiermit werden die kartesischen Koordinaten $\vec{P}_n$ der Pfadpunkte im Raum angegeben.

Tabelle 4: Rückgabewerte der `voltage()`-Funktion.

Wert	**Datentyp**	**Beschreibung**
U_N	float	Hiermit wird die elektrische Spannung entlang des angegebenen Pfades zurückgegeben.

☞ Die Funktion berechnet die elektrische Spannung U_N entlang des Pfades und gibt diese zurück. Die Parameter können entweder als 2D Listen oder 2D `np.array` angegeben werden. (2 P)

Hinweis: Die Vektorenwerte sind entlang der zweiten Dimension angeordnet, d.h. `[[x1, y1, z1], [x2, y2, z2], ...]`.
Der Rückgabewert ist eine `float`-Zahl.

```
>>> e_vectors = [[1, 2, 3], [5, 1, 7], [-2, 0, 7]]
>>> path = [[0, 0, 1], [0, 1, 0], [0, 0, 1]]
>>> voltage(e_vectors, path)
3.0
```

```
>>> e_vectors = np.array([[1, 2, 3], [5, 1, 7], [-2, 0, 7]])
>>> path = np.array([[0, 0, 1], [0, 1, 0], [0, 0, 1]])
>>> voltage(e_vectors, path)
3.0
```

5 Funktion `gauss_law(e_func, p1, p2, n_res=100)`

Das Gausssche Gesetz besagt, dass wenn man die elektrische Feldstärke $\vec{E}$ über eine geschlossene Oberfläche S integriert den Gesamtfluss durch diese Oberfläche erhält, d.h.:

$$\Phi = \oint_S \vec{E} \cdot d\vec{S} \,. \tag{7}$$

Der Gesamtfluss hängt nur von den Ladungen Q_m ab, welche von der Oberfläche S eingeschlossen werden. Wie in Abb. 3 dargestellt, wird in diesem Fall die geschlossene Oberfläche eines Quaders benutzt, der parallel zu allen drei Koordinatenachsen liegt und durch zwei äusserste Punkte definiert wird. Die Oberfläche S wird in kleine vektorielle Flächenelemente $\Delta\vec{S}$ unterteilt, deren Betrag der Flächeninhalt des Elements ist, und deren Richtung senkrecht auf der Fläche steht (Normalenvektor) und immer von innen nach aussen zeigt, z.B.:

$$\Delta\vec{S}_1 = \begin{bmatrix} S_{1,x} \\ S_{1,y} \\ S_{1,z} \end{bmatrix} = \Delta x \Delta y \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \Delta\vec{S}_2 = \Delta x \Delta z \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \Delta\vec{S}_3 = \Delta y \Delta z \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \tag{8}$$

Somit wird das Integral mit einer Summe approximiert:

$$\Phi \approx \sum_n \vec{E}_n \cdot \Delta\vec{S}_n \,, \tag{9}$$

wobei die elektrischen Feldstärkevektoren $\vec{E}_n$ jeweils in der Mitte der Teilflächen ΔS_n ermittelt werden.

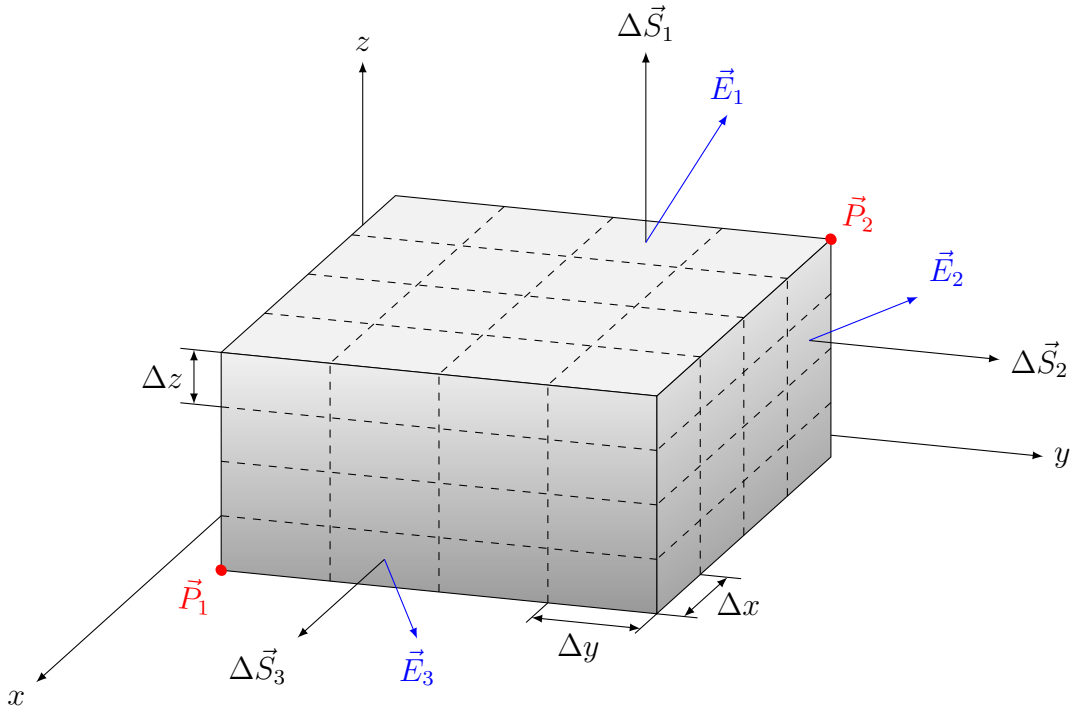

Abbildung 3: Die geschlossene Oberfläche S des Quaders, welcher mittels den äussersten Punkte $\vec{P}_1$ und $\vec{P}_2$ definiert wird.

Es soll die Funktion `gauss_law(e_func, p1, p2, n_res=100)` implementiert werden, welche den Gesamtfluss Φ anhand der obigen Summe berechnet und zurückgibt. Die Parameter und Rückgabewerte der Funktion sind in Tab. 5 und Tab. 6 aufgelistet.

Tabelle 5: Parameterliste der `gauss_law()`-Funktion.

Parameter	**Datentyp**	**Beschreibung**
`e_func`	Funktion	Hiermit wird eine Referenz auf eine externe Funktion angegeben, welche die elektrischen Feldstärkevektoren $\vec{E}$ an den gewünschten Punkten `p` im Raum zurückgibt. Aufruf: `e_func(p)` Parameter: `p` kann entweder eine 1D/2D Liste oder 1D/2D `np.array` sein. Rückgabewert: 1D oder 2D `np.array` der Vektoren $\vec{E}$. **Hinweis:** Die Vektorenwerte sind im 2D-Fall entlang der zweiten Dimension angeordnet, d.h. `[[x1, y1, z1], [x2, y2, z2], ...]`.
`p1`	`list` oder 1D `np.array`	Hiermit werden die kartesischen Koordinaten des ersten Punktes $\vec{P}_1$ vom Quader angegeben.
`p2`	`list` oder 1D `np.array`	Hiermit werden die kartesischen Koordinaten des zweiten Punktes $\vec{P}_2$ vom Quader angegeben.
`n_res`	`int`	Hiermit wird die Anzahl der Unterteilungen pro Dimension angegeben, z.B. in Abb. 3 wurde `n_res=4` gewählt. Default-Wert: 100.

Tabelle 6: Rückgabewerte der `gauss_law()`-Funktion.

Wert	**Datentyp**	**Beschreibung**
Φ	`float`	Hiermit wird der Gesamtfluss Φ durch die Oberfläche S zurückgegeben.

☞ Die Funktion berechnet den Gesamtfluss Φ durch die Oberfläche S des Quaders, welcher mittels den äussersten Punkte (über irgendeine Raumdiagonale) definiert wird. (3 P)

```
>>> def myfunc(p):
        return e_field([2e-9, -3e-9], [[0, 0, 1], [0, 0, 3]], p)
```

```
>>> gauss_law(myfunc, [-3, -3, -3], [3, 3, 4])
-112.9424808203396
```

```
>>> gauss_law(myfunc, np.array([3, -3, 2]), [-3, 3, -3])
225.88734353382404
```

```
>>> gauss_law(myfunc, [3, -3, -3], [-3, 3, 0], n_res=1000)
9.418974444871128e-06
```