

Benedikt Sander, Felix Ulbrich, Niklas Düser

a) $\text{PDF}(\Delta\psi) = \begin{cases} N \cdot \exp(-|\Delta\psi| \cdot k), & \text{if } \Delta\psi \in [-\pi, \pi] \\ 0 & \text{otherwise} \end{cases}$

Bestimme N :

$$\int_{-\infty}^{\infty} \text{PDF}(\Delta\psi) d\Delta\psi = 1$$

$$\text{Aus Symmetrie folgt } \Rightarrow \frac{1}{2} = \int_{-\pi}^{\pi} N \cdot e^{-|\Delta\psi|k} d\Delta\psi = \left[-\frac{N}{k} e^{-|\Delta\psi|k} \right]_{-\pi}^{\pi} = \frac{-N}{k} (e^{-\pi k} - 1) \stackrel{!}{=} \frac{1}{2}$$

$$\Rightarrow N = \frac{-k}{2(e^{-\pi k} - 1)}$$

Somit kann $\text{PDF}(\Delta\psi)$ mittels

```

N = - self.k / (2 * (np.exp(-np.pi * self.k) - 1))

pdf = np.zeros_like(delta_dir)

if np.isscalar(delta_dir):
    if delta_dir >= -np.pi and delta_dir <= np.pi:
        return N * np.exp(-abs(delta_dir) * self.k)
    elif delta_dir > np.pi:
        return 0
    elif delta_dir < -np.pi:
        return 0
    else:
        return 0
else:
    for i in range(0, len(delta_dir)):
        if delta_dir[i] >= -np.pi and delta_dir[i] <= np.pi:
            pdf[i] = N * np.exp(-abs(delta_dir[i]) * self.k)
        elif delta_dir[i] > np.pi:
            pdf[i] = 0
        elif delta_dir[i] < -np.pi:
            pdf[i] = 0
        else:
            pdf[i] = 0
    return pdf

```

implementiert werden.

b)

Unterscheidung in 4 Fälle:

$$\text{I)} \Delta\psi < -\pi, \text{II)} -\pi \leq \Delta\psi < 0, \text{III)} 0 \leq \Delta\psi \leq \pi, \text{IV)} \Delta\psi > \pi$$

$$\text{I)} \text{DF}(\Delta\psi) = 0$$

$$\text{IV)} \text{DF}(\Delta\psi) = 1$$

$$\text{II)} \text{DF}(\Delta\psi) = \int_{-\pi}^{\Delta\psi} \text{PDF}(\Delta\psi) d\Delta\psi = \int_{-\pi}^{\Delta\psi} N \cdot e^{-|\Delta\psi|k} d\Delta\psi$$

$$= \frac{N}{k} \left[e^{-|\Delta\psi|k} \right]_{-\pi}^{\Delta\psi} = \frac{N}{k} (e^{\pi k} - e^{-\pi k}) = \frac{(e^{\pi k} - e^{-\pi k})}{2(e^{-\pi k} - 1)}$$

$$\text{III)} \text{DF}(\Delta\psi) = \int_0^{\Delta\psi} \text{PDF}(\Delta\psi) d\Delta\psi + \frac{1}{2} = \int_0^{\Delta\psi} N \cdot e^{-|\Delta\psi|k} d\Delta\psi + \frac{1}{2}$$

$$= \frac{N}{k} \left[e^{-|\Delta\psi|k} \right]_0^{\Delta\psi} + \frac{1}{2} = \frac{N}{k} (e^{-\pi k} - 1) + \frac{1}{2} = \frac{(e^{-\pi k} - 1)}{2} + \frac{1}{2}$$

$$= \frac{N}{K} [e^{-\Delta\Psi K}]^{\Delta\Psi} + \frac{1}{2} = \frac{N}{K} (e^{-\Delta\Psi K} - 1) + \frac{1}{2} = \frac{(e^{-\Delta\Psi K} - 1)}{2(e^{-\Delta\Psi K} - 1)} + \frac{1}{2}$$

Implementiert wird das dann folgen der maßen:

```

cdf = np.zeros_like(delta_dir)
if np.isscalar(delta_dir):
    if delta_dir <= 0 and delta_dir >= -np.pi:
        cdf = (np.exp(-np.pi * self.k) - np.exp(delta_dir * self.k)) / (2 * ((np.exp(-np.pi * self.k) - 1)))
    elif delta_dir > 0 and delta_dir <= np.pi:
        cdf = (np.exp(-delta_dir * self.k) - 1) / (2 * ((np.exp(-np.pi * self.k) - 1)) + 1/2)
    elif delta_dir > np.pi:
        cdf = 1
    elif delta_dir < -np.pi:
        cdf = 0
    else:
        for i in range(0, len(delta_dir)):
            if delta_dir[i] <= 0 and delta_dir[i] >= -np.pi:
                cdf[i] = (np.exp(-np.pi * self.k) - np.exp(delta_dir[i] * self.k)) / (2 * ((np.exp(-np.pi * self.k) - 1)))
            elif delta_dir[i] > 0 and delta_dir[i] <= np.pi:
                cdf[i] = (np.exp(-delta_dir[i] * self.k) - 1) / (2 * ((np.exp(-np.pi * self.k) - 1)) + 1/2)
            elif delta_dir[i] > np.pi:
                cdf[i] = 1
            else:
                cdf[i] = 0
return cdf

```

c) Unterscheidung in 4 Fälle

$$\text{I)} q \leq 0, \text{ II)} 0 \leq q < 0.5, \text{ III)} 0.5 \leq q < 1, \text{ IV)} q > 1$$

$$\text{I)} PPF(q) = -\Delta\psi$$

$$\text{IV)} PPF(q) = \Delta\psi$$

$$\text{II)} CDF(\Delta\psi) = q = \frac{e^{-\Delta\psi k} - e^{\Delta\psi k}}{2e^{-\Delta\psi k} - 2}$$

$$\Leftrightarrow q(2e^{-\Delta\psi k} - 2) = e^{-\Delta\psi k} - e^{\Delta\psi k}$$

$$\Leftrightarrow q(2e^{-\Delta\psi k} - 2) + e^{-\Delta\psi k} = -e^{\Delta\psi k}$$

$$\Leftrightarrow \ln(q(2e^{-\Delta\psi k} - 2) + e^{-\Delta\psi k}) = -\Delta\psi k$$

$$PPF(q) = \Delta\psi = -\frac{\ln\{q(2e^{-\Delta\psi k} - 2) + e^{-\Delta\psi k}\}}{k}$$

III)

$$CDF(\Delta\psi) = q = \frac{e^{-\Delta\psi k} - 1}{2e^{-\Delta\psi k} - 2} + \frac{1}{2}$$

$$\Leftrightarrow (q + \frac{1}{2})(2e^{-\Delta\psi k} - 2) + 1 = e^{-\Delta\psi k}$$

$$\Leftrightarrow PPF(q) = \Delta\psi = -\frac{\ln\{(q + \frac{1}{2})(2e^{-\Delta\psi k} - 2) + 1\}}{k}$$

Die Implementierung sieht dann folgend aus:

```

ppf = np.zeros_like(q)
if np.isscalar(q):
    if q == 0:
        ppf = -np.pi
    elif q == 0.5:
        ppf = 0
    elif q == 1:
        ppf = np.pi
    elif q > 0 and q < 0.5:
        ppf = np.log(self.k * q / ( N ) + np.exp(- np.pi * self.k)) / self.k
    elif q > 0.5 and q < 1:
        ppf = - np.log(1 - (self.k/N) * (q - 1/2)) /self.k
    else:
        for i in range (0,len(q)):
            if q[i] == 0:
                ppf[i] = -np.pi
            elif q[i] == 0.5:
                ppf[i] = 0
            elif q[i] == 1:
                ppf[i] = np.pi
            elif q[i] > 0 and q[i] < 0.5:
                ppf[i] = np.log(self.k * q[i] / ( N ) + np.exp(- np.pi * self.k)) / self.k
            elif q[i] > 0.5 and q[i] < 1:
                ppf[i] = - np.log(1 - (self.k/N) * (q[i] - 1/2)) /self.k
return ppf

```

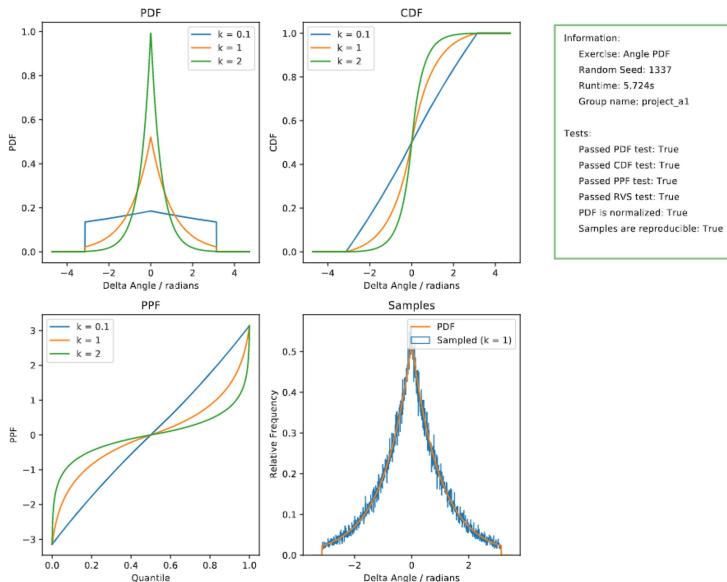
d) rvs kann dann einfach mit der aus c) geschriebenen Funktion implementiert werden.

```

n = random_state.uniform(0,1, size)
rvs = self.ppf(n)

return rvs

```



Die mittels der PPF gesammelten Daten beschreiben gut die zugehörige Kurve der PDF. Die PPF ist also eine geeignete Methode.

Blatt_4,Dueser,Sander,Ulbrich

May 25, 2021

1 Aufgabe 10

Hier in der Datei soll nur der Anwendungsweg des Codes erklärt werden. Sie ist hier nicht ausführbar.

Der ausführbare Code befindet sich in “multiple_scattering.py”

Für die Teilaufgabe a) soll ein Weg implementiert werden bei dem mit der Instanz “random_state”, welche hier “default_rng()” entspricht, zufällige “absorbtion_length” Werte gezogen werden. Diese werden hier im float “distance” gespeichert.

```
[ ]: distance= random_state.exponential(scale=self.absorption_length)
```

Für die b) werden zuerst die Startzeitpunkte in die Liste “energy_depositions” gepackt, welche am Ende der Funktion returned werden soll.

Nun ist der Gedanke für das Teilchen die Wege zwischen zwei Streuungen zu ziehen, die floats, die den Ort des Teilchens beschreiben, die Richtung berücksichtigend, um diesen Wert zu erhöhen. Anschließend wird dieser Wert von “distance” subtrahiert, so dass “distance” die Reststrecke des Teilchens beschreibt.

Dabei muss die Fallunterscheidung gemacht werden ob die gezogene Länge größer oder kleiner als die Reststrecke ist.

Fall 1 segment > distance

“segment” wird auf die Ortsvariablen addiert. Dabei werden die Komponenten in die Richtung über sinus und cosinus und den Winkel “direction” bestimmt.

Nun wird “distance” dekrementiert, und die neue Richtung mit der gegebenen Funktion “angle_distribution.rvs(random_state)” neu gesetzt.

Dabei wird der neue Wert auf den alten addiert. Auf einen overflow über 2π muss dabei nicht geachtet werden Außerdem werden die Variablen mit E=0 an “energy_depositions” angehängt.

Fall 2 segment \leq distance

Vorgehen ist identisch nur, dass “distance” auf 0 gesetzt wird, die Energie mit angehängt wird und der Winkel nicht neu gewürfelt wird.

Das ganze wird so lange durchlaufen bis Fall2 einmal eintritt.

```
[ ]: energy_depositions = [(self.x, self.y, 0., self.direction)]
```

```
while distance > 0:
```

```

segment= random_state.exponential(scale=self.scattering_length)

if segment>=distance:

    self.x+=distance * np.cos(self.direction)
    self.y+=distance * np.sin(self.direction)

    distance=0

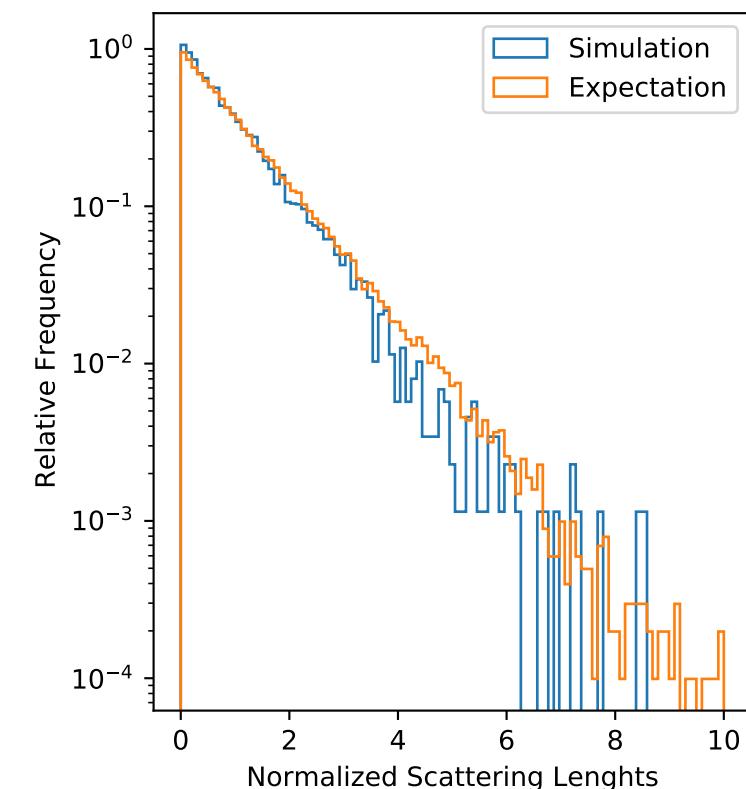
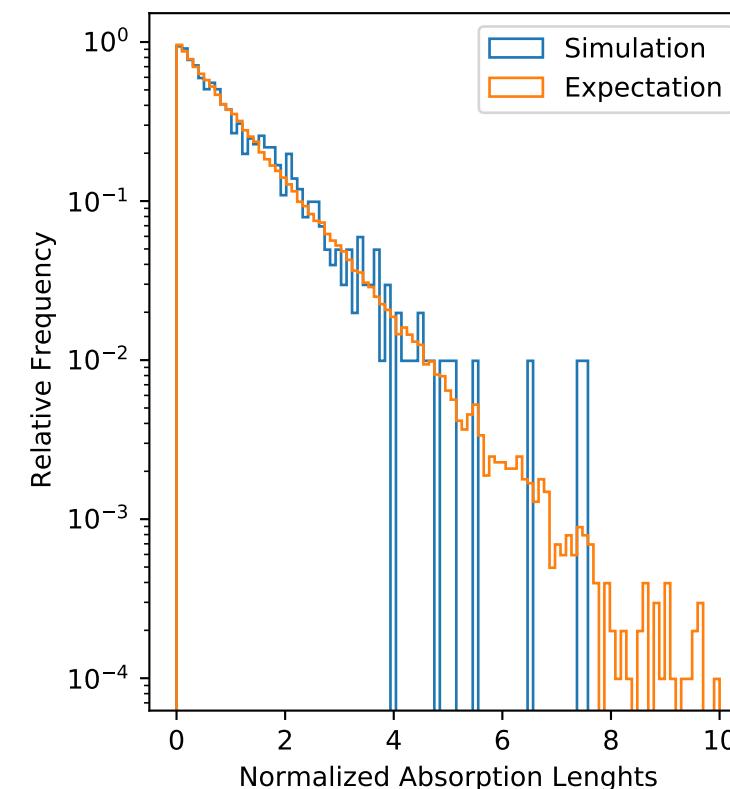
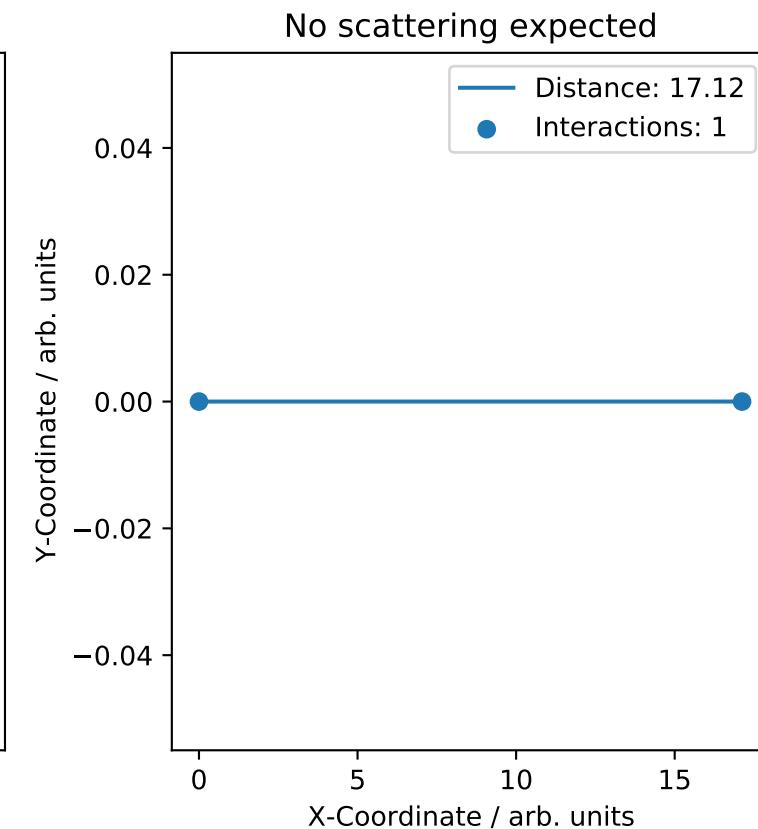
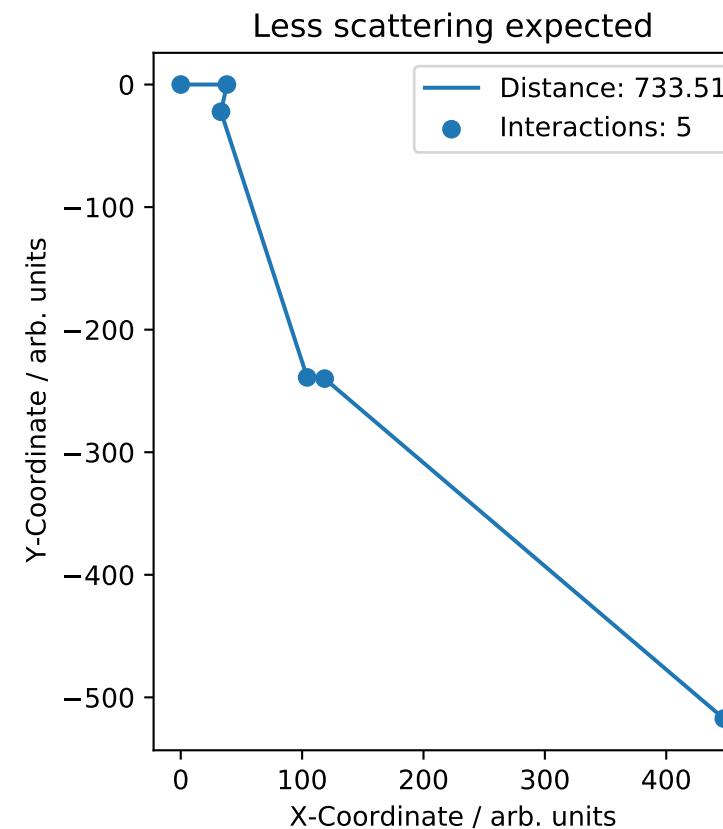
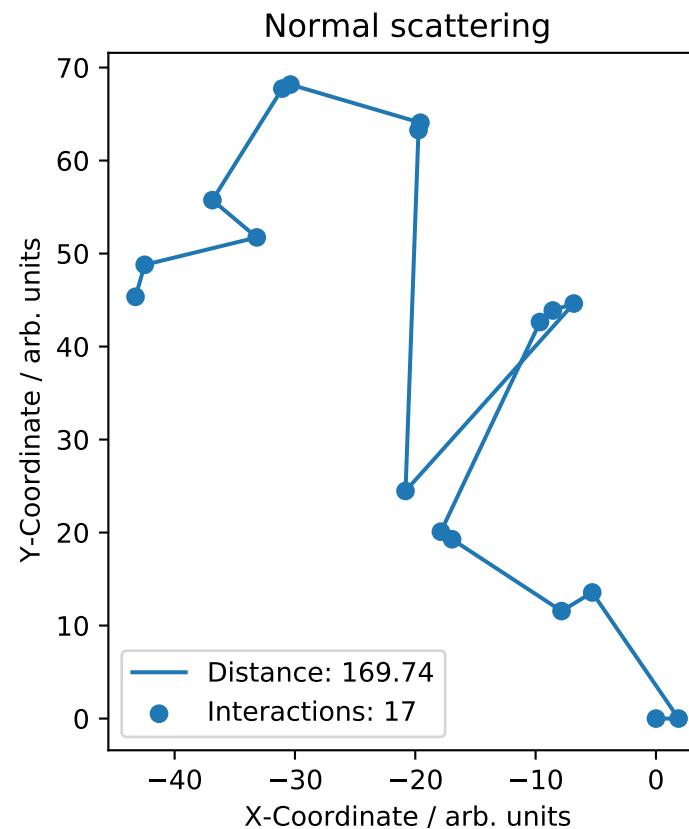
    energy_depositions.append((self.x,self.y,self.energy,self.
    ↪direction ))
else:
    self.x+=segment* np.cos(self.direction)
    self.y+=segment* np.sin(self.direction)

    distance-=segment
    self.direction+=self.angle_distribution.rvs(random_state)

    energy_depositions.append((self.x,self.y,0.,self.direction ))

```

Das Ergebnis ist zufriedenstellend. Allerdings haben wir keine Ahnung warum der “TestPassed directions test: False” nicht bestanden wird.



Information:

- Exercise: MC Multiple Scattering Particle
- Angle PDF: DeltaAngleDistribution
- Num Repetitions: 1000
- Random Seed: 1337
- Runtime: 1.920s
- Group name: project_a1

Tests:

- Passed directions test: False
- Results are reproducible: True
- First interaction is vertex: True
- Passed scattering pdf test: True
- Deposited energy is correct: True