Sheet 4 Übung

Dienstag, 17. Mai 2022 21:32

9)

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

a)

$$\int_{-\infty}^{\infty} \mathrm{PDF}(\Delta\psi)\, d\Delta\psi \overset{!}{=} 1$$

$$= \int_{-\pi}^{\pi} N \cdot e^{-|\Delta\psi| \cdot k}\, d\Delta\psi = 2 \cdot N \cdot \int_{0}^{\pi} e^{-\Delta\psi \cdot k}\, d\Delta\psi$$

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

$$\Leftrightarrow N = \frac{k}{2} \frac{1}{1 - e^{-\pi k}}$$

b)

$$\mathrm{CDF}(\Delta\psi) = \int_{-\infty}^{\Delta\psi} \mathrm{PDF}\, d\Delta\psi = \begin{cases} \mathrm{I} & 0 < \Delta\psi < \pi \\ \mathrm{II} & -\pi \leq \Delta\psi \leq 0 \\ 0 & \text{sonst} \end{cases}$$

I)

$$\int_{-\pi}^{\Delta\psi} N \cdot e^{-|\Delta\psi'| \cdot k}\, d\Delta\psi' = N \cdot \left(\int_{0}^{\Delta\psi} e^{-\Delta\psi' k}\, d\Delta\psi' + \int_{-\pi}^{0} e^{\Delta\psi' k}\, d\Delta\psi'\right)$$

$$= N \cdot \left(\left[-\frac{1}{k} e^{-\Delta\psi' k}\right]_0^{\Delta\psi} + \left[\frac{1}{k} e^{\Delta\psi' k}\right]_{-\pi}^{0}\right)$$

$$= \frac{N}{k}\left(1 - e^{-\Delta\psi k} + 1 - e^{-\pi \cdot k}\right) = \frac{N}{k}\left(2 - e^{-\Delta\psi k} - e^{-\pi k}\right) = \mathrm{I}$$

II)

$$\int_{-\pi}^{\Delta\psi} N \cdot e^{\Delta\psi' \cdot k}\, d\Delta\psi' = \frac{N}{k}\left[e^{\Delta\psi' k}\right]_{-\pi}^{\Delta\psi} = \frac{N}{k}\left(e^{\Delta\psi \cdot k} - e^{-\pi k}\right) = \mathrm{II}$$

c)

inverse:

I) $u = \frac{N}{2} \frac{1}{1 - e^{-\pi k}} \left(2 - e^{-\Delta\psi k} - e^{-\pi k}\right)$

inverse:

I) $u = \frac{N}{2} \frac{1}{1-e^{-\pi k}} \left(2 - e^{-\Delta\varphi k} - e^{-\pi k}\right)$

$\Leftrightarrow -\ln\left(-\left(\frac{2u}{N}\left(1-e^{-\pi k}\right) - 2 + e^{-\pi k}\right)\right) \cdot \frac{1}{k} = \Delta\varphi = PPF(u)$

mit $u \in [0, 0.5)$

II) $v = \frac{N}{k}\left(e^{\Delta\varphi \cdot k} - e^{-\pi k}\right)$

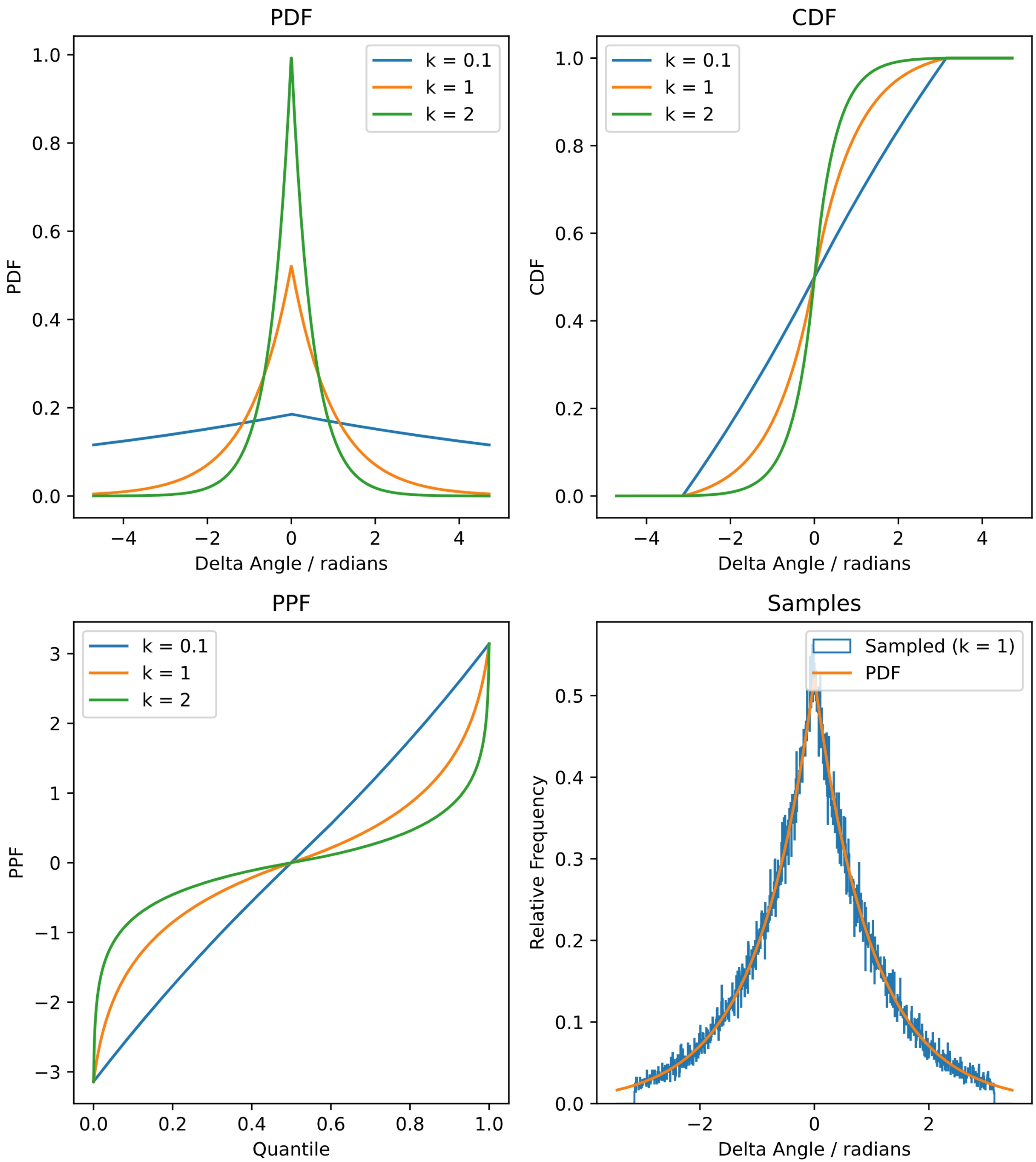
$\Leftrightarrow \ln\left(v \cdot \frac{k}{N} + e^{-\pi k}\right) \cdot \frac{1}{k} = \Delta\varphi = PPF(v)$

mit $v \in [0.5, 1]$

The code is on the following sides.

Interpretation of the results:

- smaller k → more equally distributed (as expected)
- the PDF curve fits the sample good



Information:

Exercise: Angle PDF

Random Seed: 1337

Runtime: 0.580s

Group name: project_c3

Tests:

Passed PDF test: True

Passed CDF test: True

Passed PPF test: True

Passed RVS test: True

PDF is normalized: True

Samples are reproducible: True

finished exercise 9
Lukas authored 3 days ago

143a4d7d

exp_angle_dist.py 6.66 KB

```python
from random import sample
import numpy as np


class ExponentialDeltaAngleDistribution(object):

    """Helper class for angular distribution PDF

    This is a helper class which defines the angular distribution pdf that
    may be used to describe the difference between the new and and old
    angle:
        new_angle = old_angle + delta_angle
        (plus accounting for 2-pi periodicity)

    The angle PDF used here peaks in forward direction (x=0) and drops off
    exponentially to bigger delta angles. It is defined by:

    f(x) = N * exp(-|x|k), for x in [-pi, pi)
           0, for x < -pi or x >= pi

        k: Parameter that defines how peaked the distribution is
        N: normalization constant
        N = k / (2*[1 - exp(-pi*k)])
    """

    def __init__(self, k=1.):
        """Initialize the distribution.

        Parameters
        ----------
        k : float, optional
            Defines how peaked in forward direction and how sharply the PDF
            alls off towards higher angles.
            The higher k, the sharper the PDF is peaked in forwards direction.
        """
        self.k = k
        self.N = N = self.k/2 *1/(1-np.exp(-np.pi*self.k))

    def pdf(self, delta_dir):
        """Angular distribution PDF.

        Parameters
        ----------
        delta_dir : float or array_like
            The angle between the direction of the energy deposition and the
            point at which to evaluate the angular distribution PDF.
            Value range: [-pi, pi)

        Returns
        -------
        float or array_like
            The PDF value of the angular distribution.
        """

        # ---------
        # Exercise:
        # ---------
        #   Angle PDf | part a) (exercises/angle_pdf.py)
        #
        # ---------------------------------------------------------------------
        # --- Replace Code Here
        # ---------------------------------------------------------------------

        # dummy solution to pass unit tests (this solution is not correct!)



        values = np.zeros_like(delta_dir)
        if np.isscalar(delta_dir):
            if delta_dir < -np.pi or delta_dir >= np.pi:
```

```
                None
            else:
                values = self.N*np.exp(-np.abs(delta_dir)*self.k)
        else:
            mask = np.logical_or(delta_dir > -np.pi, delta_dir < np.pi)
            values[mask] = self.N*np.exp(-np.abs(delta_dir[mask])*self.k)

        return values

    def cdf(self, delta_dir):
        """Angular distribution CDF.

        Parameters
        ----------
        delta_dir : float or array_like
            The angle between the direction of the energy deposition and the
            point at which to evaluate the angular distribution PDF.
            Value range: [-pi, pi)

        Returns
        -------
        float or array_like
            The PDF value of the angular distribution.
        """

        # ---------
        # Exercise:
        # ---------
        #   Angle PDf | part b) (exercises/angle_pdf.py)
        #
        # ---------------------------------------------------------------------
        # --- Replace Code Here
        # ---------------------------------------------------------------------

        # dummy solution to pass unit tests (this solution is not correct!)


        values = 0.5*np.ones_like(delta_dir)

        # values below range have a cdf of zero, over range have cdf of 1
        if np.isscalar(delta_dir):
            if delta_dir <= -np.pi:
                values = 0.
            elif delta_dir >= np.pi:
                values = 1.
            elif delta_dir > -np.pi and delta_dir < 0:
                values = self.N/self.k*(np.exp(self.k*delta_dir)-np.exp(-np.pi*self.k))
            elif delta_dir >= 0 and delta_dir < np.pi:
                values = self.N/self.k*(2 - np.exp(-np.pi*self.k) - np.exp(-delta_dir*self.k))
        else:
            values[delta_dir <= -np.pi] = 0.
            values[delta_dir >= np.pi] = 1.

            mask_lower = np.logical_and(delta_dir > -np.pi, delta_dir <0)
            values[mask_lower] = self.N/self.k*(np.exp(self.k*delta_dir[mask_lower])-np.exp(-np.pi*self.k))

            mask_upper = np.logical_and(delta_dir >= 0, delta_dir < np.pi)
            values[mask_upper] = self.N/self.k*(2 - np.exp(-np.pi*self.k) - np.exp(-delta_dir[mask_upper]*self.k))

        return values

    def ppf(self, q):
        """Percent point function (inverse of cdf).

        Parameters
        ----------
        q : float or array_like
            The percentile or quantile (lower tail probability) for which
            to compute the delta direction value.

        Returns
        -------
        float or array_like
            The delta direction value corresponding to the quantile `q`.
        """

        if np.any(q < 0.) or np.any(q > 1.):
            msg = 'Provided quantiles are out of allowed range of [0, 1]: {!r}'
            raise ValueError(msg.format(q))
```

```python
        # ---------
        # Exercise:
        # ---------
        #   Angle PDf | part c) (exercises/angle_pdf.py)
        #
        # ---------------------------------------------------------------------
        # --- Replace Code Here
        # ---------------------------------------------------------------------

        # dummy solution to pass unit tests (this solution is not correct!)
        values = np.zeros_like(q)
        if np.isscalar(q):
            if q < 0.5:
                values = 1/self.k*np.log(q*self.k/self.N+np.exp(-np.pi*self.k))
            elif q >= 0.5:
                values = -1/self.k*np.log(2-np.exp(-np.pi*self.k)-q*self.k/self.N)
        else:
            values[q < 0.5] = 1/self.k*np.log(q[q < 0.5]*self.k/self.N+np.exp(-np.pi*self.k))
            values[q >= 0.5] = -1/self.k*np.log(2-np.exp(-np.pi*self.k)-q[q >= 0.5]*self.k/self.N)


        return values

    def rvs(self, random_state, size=None):
        """Sample values from delta_dir PDF.

        Parameters
        ----------
        random_state : TYPE
            Description
        size : None, optional
            Number and shape of delta directions to sample.

        Returns
        -------
        float or array_like
            The sampled delta directions..
        """

        # ---------
        # Exercise:
        # ---------
        #   Angle PDf | part d) (exercises/angle_pdf.py)
        #
        # ---------------------------------------------------------------------
        # --- Replace Code Here
        # ---------------------------------------------------------------------
        # dummy solution to pass unit tests (this solution is not correct!)
        rng = np.random.default_rng(random_state)
        u = rng.random(size)
        samples = self.ppf(u)


        return samples
```

fixed correlation
Lukas authored 20 minutes ago

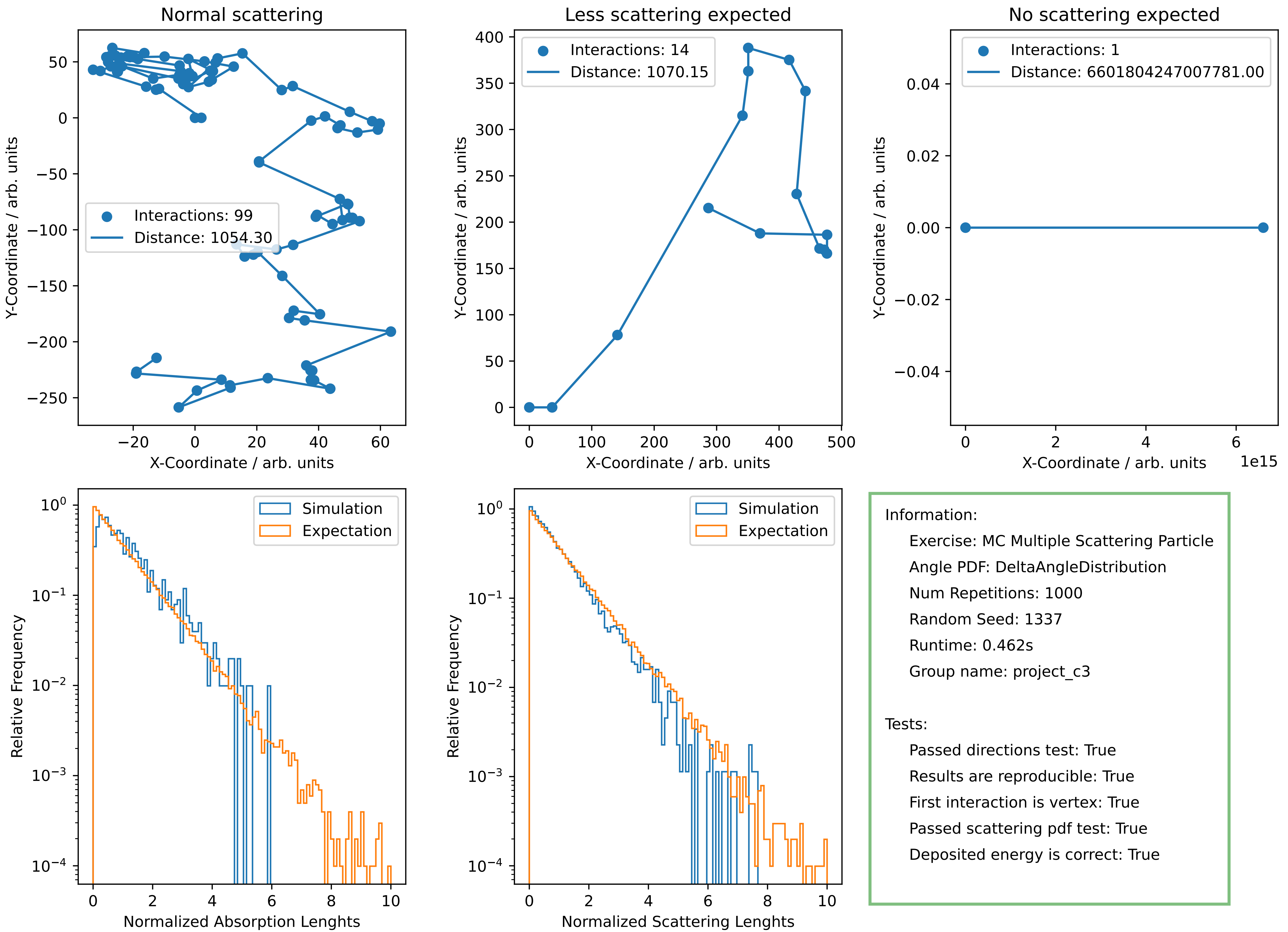
9e140fbd

multiple_scattering.py 6.28 KB

```
import numpy as np

from project_c3.simulation.particle import BaseParticle
from project_c3.simulation.detector.angle_dist import DeltaAngleDistribution
from project_c3.random import Generator


class MultipleScatteringParticle(BaseParticle):

    """Class implements a particle with multiple scattering and absorption.

    The particle can scatter and be absorbed according the defined
    scattering and absorption lengths. The total propagation length is drawn
    from an exponential with the absorption length as decay parameter.
    The next scattering point is also drawn form an exponential with the
    scattering length as decay parameter.
    The scattering is fully elastic, e.g. the amount of energy deposited at
    a scattering point is zero. At the point of absorption the particle
    deposits all of its energy `energy`.

    Note: this is how Photons are propageted in many particle experiments.

    Attributes
    ----------
    direction : float
        The direction of the particle in radians.
        This is an angle inbetween [0, 2pi).
    energy : float
        The energy of the particle. The energy uses arbitrary units.
    x : float
        The x-coordinate of the track anchor-point in detector units.
    y : TYPE
        The y-coordinate of the track anchor-point in detector units.
    """

    def __init__(self, energy, direction, x, y,
                 particle_id=0,
                 name='MultipleScatteringParticle',
                 scattering_length=10.,
                 absorption_length=150.,
                 angle_distribution=DeltaAngleDistribution(),
                 ):
        """Initialize the track particle.

        Parameters
        ----------
        energy : float
            The energy of the particle in arbitrary units. This must be greater
            equal zero.
        direction : float
            The direction of the particle in radians. The direction must be
            within [0, 2pi).
        x : float
            The x-coordinate of the track anchor-point in detector units.
        y : float
            The y-coordinate of the track anchor-point in detector units.
        particle_id : int, optional
            An optional particle identification number.
        name : str, optional
            An optional name of the particle.
        scattering_length : float, optional
            This parameter controls the distance to the next scattering point.
            The distance to the next scattering point is sampled from an
            exponential with `scattering_length` as the decay parameter.
        absorption_length : float, optional
            This parameter controls the total propagation distance until the
            particle is absorbed by the medium. The propagation distance is
            sampled from an exponential with `absorption_length` as the decay
            parameter.
        angle_distribution : DeltaAnglePDF, optional
```

```
            The delta angle distribution to use to sample new delta direction
            vectors. Default distribution is: `DeltaAngleDistribution`.
        """

        # call init from base class: this will assign the energy and direction
        super().__init__(
            energy=energy,
            direction=direction,
            particle_id=particle_id,
            name=name,
        )

        # assign values to object
        self.x = x
        self.y = y
        self.scattering_length = scattering_length
        self.absorption_length = absorption_length
        self.angle_distribution = angle_distribution

    def propagate(self, random_state, **kwargs):
        """Propagate the particle.

        This method propagates the particle and creates energy losses.
        The energy losses can be passed on to a Detector instance to generate
        an event.

        The first energy deposition should consist of the vertex of the
        particle, e.g. it should be:
            (self.x, self.y, 0., self.direction)
        The last energy deposition should be the point of absorption and the
        deposited energy should be equal to the particle's energy.

        Parameters
        ----------
        random_state : RNG object
            The random number generator to use.
        **kwargs
            Additional keyword arguments.

        Returns
        -------
        array_like
            The list of energy losses. Each energy loss consists of a tuple of
            [x-position, y-position, deposited Energy, direction].
            Shape: [num_losses, 4]
        """

        # ---------
        # Exercise:
        # ---------
        #   MC Multiple Scattering (exercises/mc_multiple_scattering.py)
        #
        # --------------------------------------------------------------------
        # --- Replace Code Here
        # --------------------------------------------------------------------

        # dummy solution to pass unit tests (this solution is not correct!)
        # this is just a dummy energy deposition list with the starting vertex
        # and one interaction point at a distance of 10 units in x-direction

        u = random_state.random()

        absorption_distance = -self.absorption_length*np.log(1-u)

        distance_travelled = 0

        energy_depositions = []

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

        while distance_travelled < absorption_distance:

            u = random_state.random()

            scattering_distance = -self.scattering_length*np.log(1-u)

            if scattering_distance > 10e10:
                print(self.scattering_length,u, scattering_distance)

            prev_pos_x = self.x
```

```
            prev_pos_x = self.x
            prev_pos_y = self.y

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


            distance_travelled += np.sqrt((self.x-prev_pos_x)**2 + (self.y-prev_pos_y)**2).item()

            if distance_travelled >= absorption_distance:
                deposited_energy = self.energy

            else:
                deposited_energy = 0

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

            self.direction = (self.direction + self.angle_distribution.rvs(random_state = random_state, size=1)).item()

        #print('finished loop')
        return energy_depositions
```



Normal scattering
Interactions: 99
Distance: 1054.30
Y-Coordinate / arb. units
X-Coordinate / arb. units
Less scattering expected
Interactions: 14
Distance: 1070.15
No scattering expected
Interactions: 1
Distance: 6601804247007781.00
1e15
Simulation
Expectation
Relative Frequency
Normalized Absorption Lenghts
Normalized Scattering Lenghts
Information:
Exercise: MC Multiple Scattering Particle
Angle PDF: DeltaAngleDistribution
Num Repetitions: 1000
Random Seed: 1337
Runtime: 0.462s
Group name: project_c3
Tests:
Passed directions test: True
Results are reproducible: True
First interaction is vertex: True
Passed scattering pdf test: True
Deposited energy is correct: True

Sheeeeesh 420

Dienstag, 17. Mai 2022 22:19

10 c)

(c) The resulting distances between two scattering points deviate slightly from the exponential distribution. There are distances shorter than expected. Why does this happen?

The reason it is shorter than expected, is because we multiply an exponential distribution with a non-exponential distribution.

Interpretation:

We are confused of the cut off of the absorbtion length, since it is getting cutted of at 6 and not at 10.

At the beginning from 0 - 5 the simulation follows really good the expectation.

The first row of diagrams also fits the expectation really well.