Designing a particle physics experiment
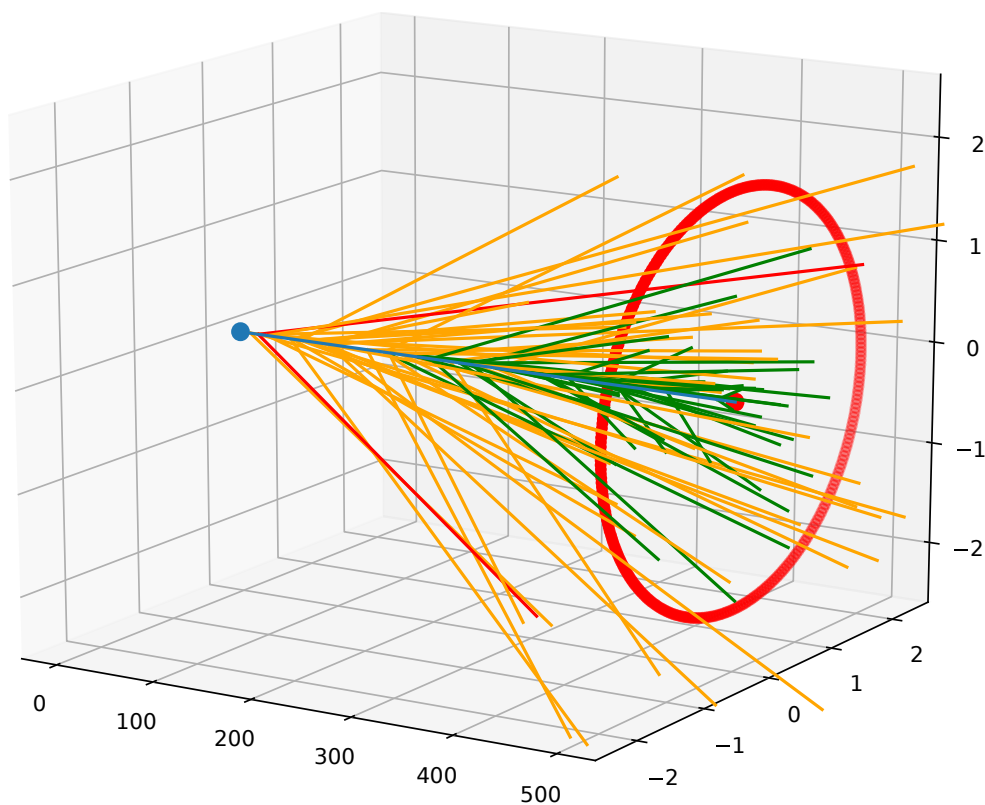
STEFANIE BRÜHLMEIER, BENI FRÖLICH, DAVE LINDER, ANDREAS WIEMEYER

supervised by
MICHELE ATZENI, ARNO GADOLA, OLAF STEINKAMP, ANDREAS WEIDEN

Physik-Institut der Universität Zürich

Contents

1 Introduction

In our project we are dealing with charged kaons which decay exponentially into charged and neutral pions:

$$K^+ \longrightarrow \pi^+ \pi^0$$

To simplify the calculations, we neglect the decay of the pions. The aim of this simulation is to determine the distance between a kaon source and a detector which maximizes the acceptance of the detector. The detectors are set up as follows:
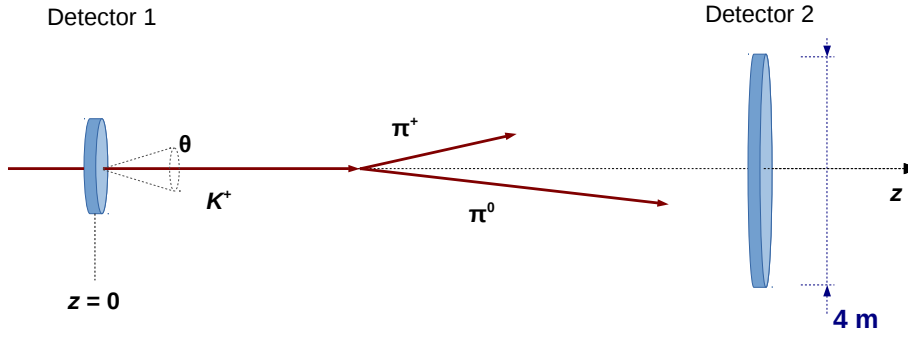

Figure 1: Experimental setup

As shown in figure 1 the initial beam of kaons is measured by detector 1 and the resulting pions are measured by detector 2. The second detector is a circular disc with a radius of 2 meters. Depending on the location of the decay and the decay angle (in the K^+ rest frame) the pions might miss the pion detector. Thus, the closer the second detector is to the first, the bigger the probability that the pions are detected. However, if the detector is too close to the first one, the kaons might decay after they have already passed the detector and will therefore not be detected. Given these two constraints, there is a unique optimal distance between the first and the second detector, which we compute with the following simulation. In the first part of this project, the average decay length of a K^+ is estimated based on a dataset containing the decay lengths of a mix of kaons and pions. Using the result of the first part we can simulate the location of the decay using a Monte-Carlo simulation for an exponential distribution. Further, some impulse vectors[1] (which are distributed isotropically in the K^+ rest frame) are generated, supplemented with an energy to form a four-vector and boosted into the laboratory frame. Finally, the percentage of decayed kaons that are detected (called 'acceptance of the downstream detector') can be determined. The whole simulation is conducted twice, once assuming that the particle beam is parallel to the z-axis and once including a Gaussian distributed deviation from the z-axis of the particle beam.

---

[1]For the derivation of the isotropic unit vector generator see appendix A.1

2 Determination of the average decay length of the K^+

In a first task we should estimate the average decay length of K^+ using data of an already conducted experiment in which $100'000$ measurements of decay lengths dl_i were generated. The particle's beam had a fixed momentum of $75\,\text{GeV}\,c_0{}^{-1}$. However, the beam consisted only of 16% of K^+. The remaining 84% consisted of π^+ with a known average decay length of $\beta\gamma c_0\tau_{\pi^+} = 4.188\text{km} = adl_{\pi^+}$.
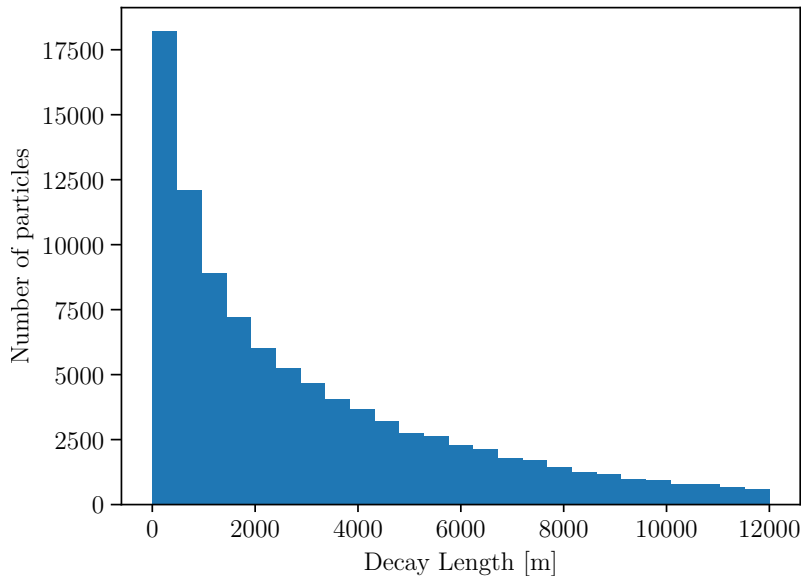

Figure 2: The particle distribution from a previous experiment with a fixed momentum of $75\,\text{GeV}$.

To estimate the average decay length of the Kaons adl_{K+} we used the maximum likelihood method with the following equation, which was subsequently maximized:

$$\ln L(dl_i|adl_{K+}) = \sum_{i=1}^{N} \ln \left(\frac{0.84}{adl_{\pi^+}} \cdot e^{\frac{-dl_i}{adl_{\pi^+}}} + \frac{0.16}{adl_{K+}} \cdot e^{\frac{-dl_i}{adl_{K+}}} \right) \tag{1}$$

The equation is composed of two exponential distributions, weighted with the corresponding probability that a particle belongs to this distribution. Implementing this in python[2] resulted in a estimator for the average decay lenght of

$$\widehat{adl}_{K+} = (562 \pm 10)\,\text{m} \tag{2}$$

---

[2]for the python code see appendix in section A.2.1

as shown in figure 3
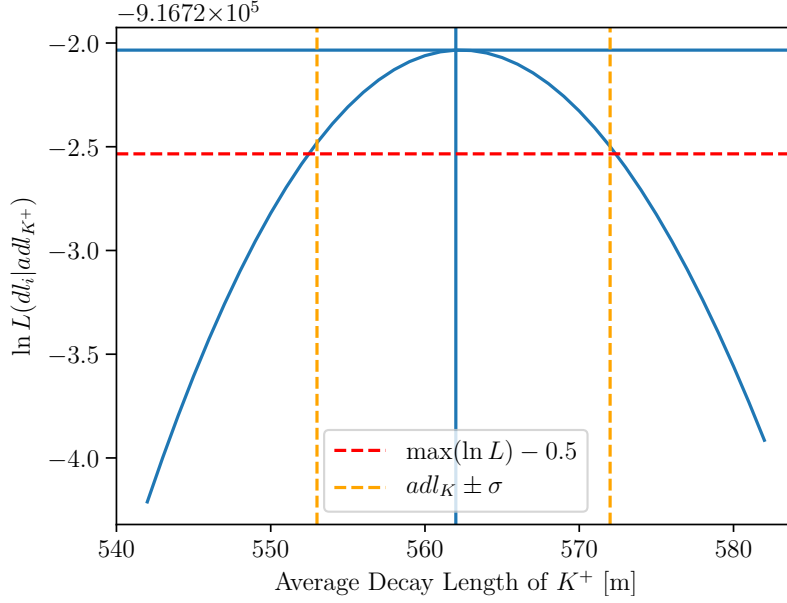
Figure 3: The log-likelihood function around its maximum. With the yellow lines representing the uncertainty.

The error $\sigma_{adl_{K+}}$ is calculated by

$$\ln L(adl_{k+} \pm \sigma_{adl_{K+}}) = \ln L(adl_{K+}) - \frac{1}{2} \tag{3}$$

Now we can calculate the average decay time

$$\tau_{K+} = \frac{adl_{k+}}{\beta \gamma c} \tag{4}$$

Where

$$\beta = \frac{|p_k|}{E_K} \tag{5}$$

$$\gamma = \frac{E_K}{m_K} \tag{6}$$

$$E = \sqrt{m^2 + |p|^2} \tag{7}$$

$$|p| = \sqrt{p_x^2 + p_y^2 + p_z^2} = 75 \, \text{GeV} \, c_0^{-1} \tag{8}$$

$$m_{K+} = (493.677 \pm 0.016) \times 10^6 \, \text{eV} \tag{9}$$

4

Then we get the result

$$\tau_{K+} = (1.235 \pm 0.022) \times 10^{-8}\,\text{s} \tag{10}$$

Compared with the literature value of $(1.2380 \pm 0.0020) \times 10^{-8}\,\text{s}$ this result seems reasonable because τ_{K+} is within the uncertainty of the pdg value.

3 Optimal detector distance

We will not follow the order of the exercises from the info sheet, since it is easier to explain our solutions if we look at exercise four and three together. As we will see, exercise three can be solved by simply skipping a step in the solution of exercise four. So what exactly had to be done to solve exercises three and four? The problem can be roughly divided into three parts:

1. Simulate the path of the kaons from the source to the point where they decay.

2. Calculate the momenta of the pions in the lab frame after the decay.

3. Find the optimal distance by checking how many pions hit the detector for each distance.

3.1 Simulation of the K^+ source

We can use the results of the first exercise to generate the distance the kaons will travel. For this we use a Monte-Carlo simulation of an exponential decay with $\lambda = 1/adl$. Now that we have the distance travelled, we need the direction of flight to determine the location of the decay. For exercise three the direction of flight is always along the z-axis, so the locations of the decay can be calculated simply by multiplying the distances travelled with the vector $[0; 0; 1]$. For exercise four we will need to rotate these vectors such that the angles between them and the z-axis match a normal distribution. To do this we generate an angle α according to a normal distribution (with a σ of $1\,\text{mrad}$) and a second angle β according to a uniform distribution (from 0 to π). We then rotate the vectors around the x-axis by α and then by β around the z-axis. With the first part done we can move on to the second.

3.2 Simulating the decay

For the calculations done in this part of the simulation we will need (in addition to the list values and equations 5 - 9) the following list values from the Particle Data Group:[3]

$$m_{pi+} = (139.570\,61 \pm 0.000\,24) \times 10^6 \text{ eV}$$
$$m_{pi0} = (134.9770 \pm 0.0005) \times 10^6 \text{ eV}$$

In the K^+ rest frame we now have a momentum of 0 and a rest energy m_K^+. The K^+ now decays into a π^+ and a π^0 with momenta $p_{\pi+}$, $p_{\pi0}$ in oposing directions. The momenta must have the same magnitude and $p_{\pi+} = -p_{\pi0}$.

The magnitude of the momentum is calculated with the energy of lost mass in the decay. In the following equations p is used as absolute scalar.

$$E_{\pi+} = \frac{m_{K+}^2 + m_{pi+}^2 - m_{pi0}^2}{2m_{K+}} \tag{11}$$

$$p_{\pi+} = \sqrt{E_{\pi+}^2 - m_{pi+}^2} \tag{12}$$

$$p_{\pi+} = p_{\pi0} \tag{13}$$

We use isotropically distributed unit vetors[4] in the simulation . We multiply them with $p_{\pi0}$ to get the momenta for π^0 in the K^+ rest frame. Tho get the momenta of π^+ we take the same distribution and flip signs. We then build the four vectors P_π^*.

$$P_\pi^* = \begin{bmatrix} E_\pi^* \\ p_{\pi,x}^* \\ p_{\pi,y}^* \\ p_{\pi,z}^* \end{bmatrix} = \begin{bmatrix} \sqrt{m_\pi^2 + ||p_\pi||^2} \\ p_{\pi,x}^* \\ p_{\pi,y}^* \\ p_{\pi,z}^* \end{bmatrix} \tag{14}$$

Back to the lab rest frame, we boost them with the boost matrix. This boost matrix performs a boost along the z-axis.

---

[3]Values from Particle Data Group (m_{K+}), (m_{pi+}), (m_{pi0})

[4]For the derivation see appendixA.1

$$P_\pi = \begin{bmatrix} \gamma & 0 & 0 & \beta\gamma \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \beta\gamma & 0 & 0 & \gamma \end{bmatrix} \cdot P_\pi^* \qquad (15)$$

P_π is the four-vector in the lab frame. P_π^* is the four-vector in the kaon rest frame.

3.2.1 With scattering

When we use a beam that is not parallel to the z-axis, we need to perform a rotation to the momentum of the four-vector before and after the boost. The rotation after the boost is the same that would rotate e_z to align with the position vector. The one before is the inverse rotation.

Since the momentum of the four-vectors P_π of the pions are isotropically distributed in the K^+ rest frame, we can omit the first rotation and still end up with a an isotropical distribution.

The operation performed with the angle α around y-axis and β around x-axis:

$$R(\alpha, \beta) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \qquad (16)$$

We apply $R(\alpha, \beta)$ to the momentum after we have extracted the momentum of the four-vector P_π.

3.3 Counting double hits

To determine the number of decays for which both pions hit the detector, we first define a distance z_d from source to detector. And a radius r of the detector. The impulses in the lab frame are $\overrightarrow{p_{\pi+}}$ and $\overrightarrow{p_{\pi 0}}$. And decay position $\overrightarrow{x} = (x, y, z)$. We want to find the number of double hits[5]

1. Drop all the decay entries where $z > z_d$.

---

[5]See Appendix A.2.2 for single hits and A.2.3 for double hits implementation in Python.

2. Do the following for $\overrightarrow{p} = \overrightarrow{p_{\pi^+}}$ or $\overrightarrow{p_{\pi^0}}$.

   (a) Calculate s such that $\overrightarrow{q} = \overrightarrow{x} + s \cdot \overrightarrow{p}$ lands in the plane of z_d.

   (b) Drop the pion if it's outside the detector. $(x + q_x)^2 + (y + q_y)^2 > r^2$.

3. Drop the decay if at least one of the two pions was dropped.

4. Count the number of decays left.

3.3.1 Determine optimal detector distance z_d

We will now determine the optimal distance such that the acceptance of is maximized. Acceptance is defined as follows:

$$P(z_d) = \frac{n \text{ decays for which both pions hit at } z_d}{n \text{ decays simulated}} \tag{17}$$

To determine the optimal z-distance, we implemented the algorithm described in 3.3. We let it run for z values in a range from 215 to 350 meters in 85 evenly sized steps. The algorithm generates 85 double hit counts in an array. We divide by N to get the probabilities. Then we fit it with a polynom of degree 3 and determine the maximum of that polynom. To show it graphically, we will show a plot. The simulation was run once with $N = 10^6$ decays.

To maximize the acceptance we run the experiment about 500 times with $N = 10^6$ kaons generated. First we simulate a beam aligned to the z-axis. Second we generate a beam that scatters at the source in a gaussian distribution with $\sigma_\theta = 0.001$ rad.

We plot the data for each experiment done and fit a maximum. As an example we show one plot and fit here 4. The maxima are then put into a histogram. the average and standard deviation of the maxima are listed below. The standard deviation gives a measure of the statistical uncertainty of the simulation.

Optimal z for aligned Beam

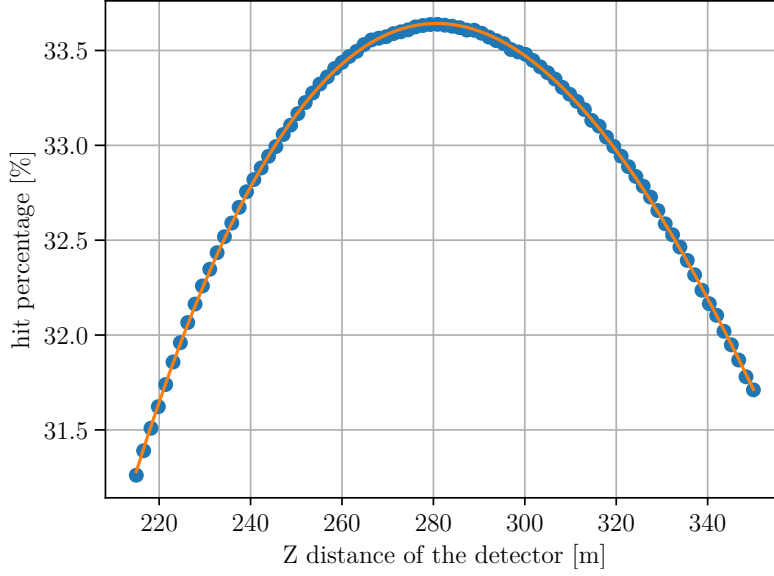$$z_{\text{aligned}} = 285.5 \, \text{m}$$
$$\sigma_z : 0.7 \, \text{m}$$

Figure 4: Sample output of decay_simulation_optimized.py. Changing the z position of the detector simulated with 10^6 decays, $\sigma_\theta = 0.001\,\mathrm{rad}$.

Optimal z for scattered Beam

$$z_{\text{scattered}} = 280.7\,\mathrm{m}$$
$$\sigma_z : 0.7\,\mathrm{m}$$

To calculate the final uncertainty we used the relative error on the average decay length from chapter 2. The relative errors get quadratically added.

$$m_z = \sigma_z \cdot \sqrt{\left(\frac{\sigma_{adl_{K+}}}{adl}\right)^2 + \left(\frac{\sigma_z}{z}\right)^2} \tag{18}$$

The final optimal values for z are:

$$z_{\text{aligned}} = (286 \pm 5)\,\mathrm{m}$$
$$z_{\text{scattered}} = (281 \pm 5)\,\mathrm{m}$$

The error does not matter that much, since a change of 5 meters around the maximum does maximally change the acceptance by 0.5% as seen in figure 5.
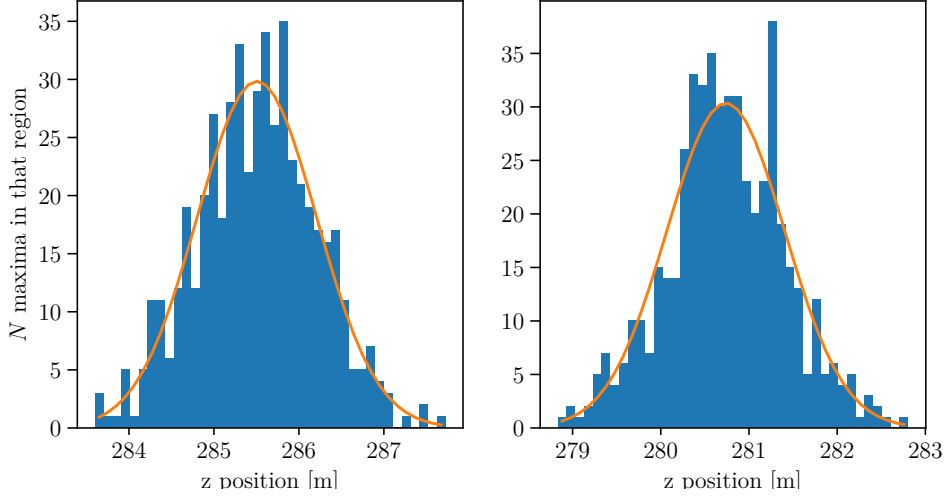
9

Figure 5: Left: Optimal z for aligned beam, 511 simulations. Right: Optimal z for scattered beam, 512 simulations.

4 Conclusion

As we have seen the optimal distance between the detector and the source is around 280 meters (for the precise values check section 3.3.1). Therefore the experiment can be conduced optimally without expanding the provided area.

For the beam with gaussian scattering around the z-axis, the optimal distance should be shorter than without scattering, because the deviation from the z-axis leads to a higher probability of the pions missing the detector. Therefore our results are in line with what's to be expected.

It's also worth mentioning that if we were to account for the pion decay, the optimal distance would tend to be shorter, since the decays of pions can only reduce the acceptance at longer distances.

A Appendix

A.1 Derivation of the isotropic unit vector generator

For exercise 3 we need to generate vectors that are isotropically distributed in three-dimensional space. That is, we need to generate vectors that have an equal chance to point in any direction. This is not as simple as calculating two uniformly distributed angles φ and ψ and rotating a vector around two different axes by these angles. To see this one can do the following:

Imagine a sphere whose centre is, for sake of simplicity, situated at the centre of the coordinate system and whose radius is one. Now imagine a vector that reaches from its centre to its surface. Again, for sake of simplicity, let's say that this vector is aligned with one axis (let's say the z-axis to match our code). We are interested in the position of the end-point of the vector. The vector is now rotated by φ along one axis orthogonal to its starting position (say the x-axis) and then rotated again by psi around the other axis orthogonal to its starting position (the y-axis). We can see that the first rotation determines a limited space through which the end-point can move by rotating the vector around the y-axis. More precisely, this space is a circle on the surface of the sphere with its centre on the y-axis. The probability p of the angle corresponding to each of these circles will determine how probable it is to have the end-point land somewhere on them. Since the distribution is supposed to be isotropical, we want the probability for the end-point to land on any point of the circle to be constant.
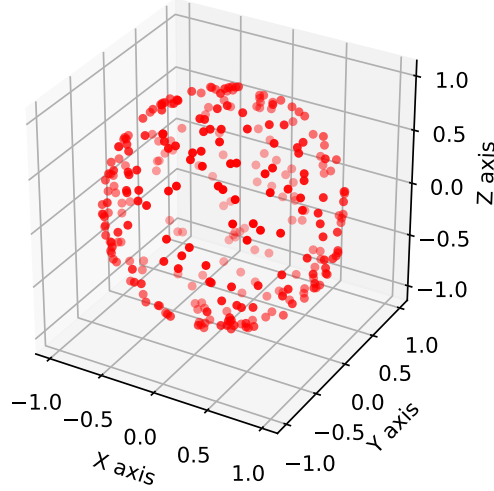
Figure 6: Randomly distributed three vector on a unit sphere.

From this follows that the probability to land on each circle should be proportional to its circumference. Since the circumference of the circle is a non-constant function of φ and since (as we have stated) the probability to land on one circle is equal to the probability of the corresponding φ, the probability for all angles φ mustn't be constant. We can calculate the probability for each angle as follows:

$$p(\varphi) \propto \text{circumference} \implies \frac{p(\varphi)}{\text{circumference}} = k \quad k \in \mathbb{R} \tag{19}$$

And the circumference is given by the following equation:

$$\text{circumference} = 2\pi \cdot \cos(\varphi) \tag{20}$$

Using this we get an equation for $p(\varphi)$:

$$\frac{p(\varphi)}{2\pi \cdot \cos(\varphi)} = k \iff p(\varphi) = k \cdot 2\pi \cdot \cos(\varphi) \tag{21}$$

Since $\varphi \in (\frac{-\pi}{2}, \frac{\pi}{2}]$ we want the function to be normalized over this range:

$$\int_{-\pi/2}^{\pi/2} k \cdot 2\pi \cdot \cos(\varphi) \overset{!}{=} 1 \implies k \cdot 4\pi = 1 \implies k = \frac{1}{4\pi} \tag{22}$$

And we get the final equation:

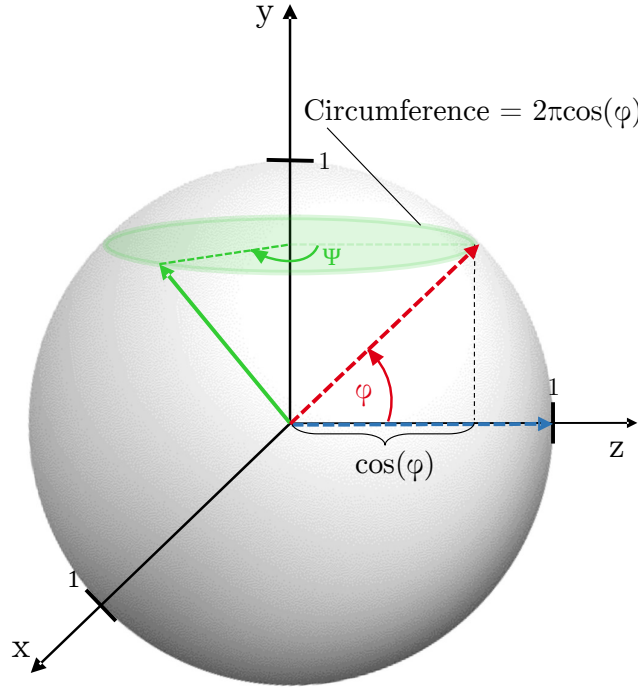$$p(\varphi) = \frac{1}{2} \cos(\varphi) \tag{23}$$

12

Figure 7: Visualisation of the parameters used in the derivation of the isotropic unit vector generator.

Now we need to somehow get a value between $(\frac{-\pi}{2}, \frac{\pi}{2}]$ that is distributed according to this function implemented in python. We weren't aware of a built-in function for this job, so we needed to get there on our own, using a uniform distribution as a starting point. To make our solution evident one can do the following. Think of the area under the probability curve $p(\varphi)$. If we scatter dots evenly across this area, the amount of dots under an infinitely small segment of the curve will be proportional to the area under the curve and thus the value of the curve at the segment. In other words: the amount of dots under the curve at φ will be proportional to $p(\varphi)$. Now Imagine the Integral of the probability curve $p(\varphi)$ as a sum of thin slices. Instead of having the pillars standing next to each other in the graph, imagine stacking them on top of each other. The leftmost pillar at the bottom, the rightmost pillar at the top. The scattered dots would now be uniformly distributed across the height of the stacked pillars. We can generate this distribution. All we need is a function to calculate back from the height in the stack to the angle φ. To find this function we take the Integral of $p(\varphi)$ from $-\pi/2$ to an undetermined value $a \in (-\frac{\pi}{2}, \frac{pi}{2}]$:

$$\int_{-\pi/2}^{a} \frac{1}{2} \cos(\varphi) = \frac{1}{2} \sin(\varphi) \Big|_{-\pi/2}^{a} = \frac{\sin(a) + 1}{2} \tag{24}$$

Speaking in our previous pseudomathematic language, this function assigns a height

13

in the stack of pillars to any value a. The absolute height of the pillar is unimportant for this to work. All we need to insure is that the range of the uniformly distributed values is equal to the range of the integral. Since the Integral ranges from 0 to 1, we will uniformly distribute from 0 to 1 as well. Now we just need to invert the previous Integral to get the function we were looking for.

$$y = \frac{\sin(a) + 1}{2} \implies a = \arcsin(2y - 1) \tag{25}$$

With y being uniformly distributed values from 0 to 1 we can use this equation to get the correct $\varphi(a)$. Now all that's left to do is to rotate the vector $[0; 0; 1]$ first by φ around the x-axis and then by ψ around the y-axis.

A.2 Python code

Here we provide a small Python code insight. The full code we released on github[6]

A.2.1 Implementation of the weighted exponential distributions to compute the estimator $\widehat{adl}_{K+}$

```python
def f(length, length_K):
    return np.log((0.16/ length_K) * np.exp(-length / length_K) + (0.84/
        length_pi) * np.exp(-length/ length_pi))

def ll(length_K):
    total = 0
    for length in lengths:
        total += f(length, length_K)
    return total

LLs = ll(length_Ks)

length_K_hut = length_Ks[list(LLs).indadx(max(LLs))]
```

A.2.2 Implementation of the single_hit function

```python
def single_hit(decay_position, four_vecs, z_detector, detector_radius):
"""check if one particle hit the detector"""
    p = four_vecs[:, 1:]
    x, y, z = decay_position.T

    # q is the position the particle will pass in the plane through
        z_detector
    q = p.T * (-z + z_detector) / p[:,2]
    qx, qy, qz = q

    # return whether the particles hit the circular disk
    return (x + qx)**2 + (y + qy)**2 < detector_radius**2
```

A.2.3 Implementation of the count_double_hits function

```python
def count_double_hits(decay_positions, particle1_4vec, particle2_4vec, r,
   z_detector):
        """count for how many of the decays, particle1 and particle2 hit the
            detector"""

        # numpy filter to filter the ones that decay in front of the
            detector
        in_front = decay_positions[:,2] < z_detector

        # drop the decays that are behind
        decay_positions = decay_positions[in_front]
        particle1_4vec = particle1_4vec[in_front]
        particle2_4vec = particle2_4vec[in_front]

        # boolean array. True if hit, False if miss
        hits1 = single_hit(decay_positions, particle1_4vec, z_detector, r)
        hits2 = single_hit(decay_positions, particle2_4vec, z_detector, r)

        # combine the with elementwise and.
        hits = hits1 & hits2
        return len(decay_positions[hits])
```