Example 1 for /r/LLMPhysics

Simulations of Two Body Decays

Conquest★Ace

Simulating Two Body Decays

When an energetic beam of protons collides with a target material, it produces lots of particles, many of which are pions. We want to simulate the decay of $\pi^0 \to \gamma\gamma$.

1 Theory

We need to consider the relativistic kinematics and some quantum field theory.

Our neutral pion is a short-lived meson composed of a quantum superposition of up-antiup ($u\bar{u}$) and down-antidown ($d\bar{d}$) quarks.

$$\pi^0 = \frac{1}{\sqrt{2}}(u\bar{u} - d\bar{d})$$

It decays via the **electromagnetic interaction** into two photons:

$$\pi^0 \to \gamma\gamma$$

1.1 Conservation Laws of π^0 at rest

- ▷ Energy needs to be conserved: $E_{initial} = E_{final}$. The energy conservation of a pion at rest is then:

$$\begin{aligned} E &= pc \\ E_{\pi^0} &= m_{\pi^0}c^2 \\ E_{\pi^0} = E_\gamma + E_\gamma &= \frac{m_{\pi^0}c^2}{2} + \frac{m_{\pi^0}c^2}{2} \\ p_1 = p_2 &= \frac{m_{\pi^0}c}{2} \end{aligned}$$

- ▷ momentum needs to be conserved: In the pion's rest frame, the total momentum of the decay products, must be zero. So the two photons are emitted in opposite directions with equal magnitudes of momentum:

$$\vec{p}_{\pi^0} = 0 = \vec{p}_1 + \vec{p}_2 \implies \vec{p}_1 = -\vec{p}_2$$

- ▷ angular momentum needs to be conserved: pion has spin-0, so two photons must have angular momentum 0. Since photons have spin-1, their helicity is ± 1 , so the spins must be opposite to cancel each other out.
- ▷ parity needs to be conserved: Pion has negative parity, so the two photon system must match that.
- ▷ charge needs to be conserved: Photons are neutral, so charge is conserved.

1.2 Kinematics of Two Body Decay

In the rest frame of our pion, it has mass $m_{\pi^0} = 135$ MeV and no initial momentum. After decaying into the two photons, the photons travel at the speed of light, c just in opposite directions of one another.

In general for a particle at rest $p^\mu = (mc, 0, 0, 0)$ with energy $E = mc^2$ decaying to two particles with momenta p_1 and p_2:

$$E = E_1 + E_2$$
$$mc^2 = E_1 + E_2$$

We can solve for the momenta using the energy-momentum relation: $E^2 = p^2c^2 + m^2c^4$ for each particle.

$$E_1^2 = p_1^2c^2 + m_1^2c^4$$
$$E_2^2 = p_2^2c^2 + m_2^2c^4$$

Squaring both sides and using energy conservation:

$$m^2c^4 = (E_1 + E_2)^2 = E_1^2 + E_2^2 + 2E_1E_2$$

Substituting E_1^2 and E_2^2:

$$m^2c^4 = p_1^2c^2 + m_1^2c^4 + p_2^2c^2 + m_2^2c^4 + 2E_1E_2$$

And the following:

$$m^2c^4 - m_1^2c^4 - m_2^2c^4 = p_1^2c^2 + p_2^2c^2 + 2E_1E_2$$

$p_1 = p_2 = p$

$$m^2c^4 - m_1^2c^4 - m_2^2c^4 = 2p^2c^2 + 2E_1E_2$$

Now solve for E_1E_2:

$$E_1E_2 = \frac{(m^2 + m_1^2 - m_2^2(m^2 + m_2^2 - m_1^2)}{4m^2}c^4$$

Lastly we have the following for the magnitude of the momentum of the daughter particles of the decay:

$$p = \frac{\sqrt{(m^2 - (m_1 + m_2)^2)(m^2 - (m_1 - m_2)^2)}}{2m}c$$

When the pion is not at rest, i.e has momentum p_{π^0}, we can first analyze the decay in the rest frame of the pion like we did just have, then apply a Lorentz boost to the decay products to the lab frame.

Suppose pion has an initial 4-momenta: $p^\mu = (\gamma mc, \gamma vm, 0, 0)$ where $\gamma = \frac{1}{\sqrt{1-v^2}}$. Then, the decay products must be Lorentz-boosted to the lab frame.

$$E' = \gamma(E - vp_x)$$
$$p_x' = \gamma(p_x + vE)$$

1.3 Decay Rates

The decay $\pi^0 \to \gamma\gamma$ is given by:

$$\Gamma_{\pi\to\gamma\gamma} = \frac{(2\pi)^4}{2E_\pi} \int |M_{\pi\to\gamma\gamma}|^2 \delta(E_\pi - E_{\gamma1} - E_{\gamma2}) \delta^3(\mathbf{p}_\pi - \mathbf{p}_{\gamma1} - \mathbf{p}_{\gamma2}) \frac{d^3\mathbf{p}_{\gamma1}}{(2\pi)^3 2E_{\gamma1}} \frac{d^3\mathbf{p}_{\gamma2}}{(2\pi)^3 2E_{\gamma2}}$$

For this we need the matrix element, which we can get my analyzing the interaction of the specific coupling of the pion to photons. After we have that, we will need to perform the integration to get the total decay rate. This means we will need to know our detector's acceptance and other experimental constraints.

The neutral pion is composed of up and down quarks: $\pi^0 = \frac{1}{\sqrt{2}}(u\bar{u} - d\bar{d})$. We get an effective Lagrangian for the pion-photon interaction:

$$\mathcal{L}_{\pi^0\gamma\gamma} = \frac{\alpha}{8\pi f_\pi} \pi^0 F_{\mu\nu} \tilde{F}^{\mu\nu}$$

From this alongside with the Feynman rules, we can calculate the decay width:

$$\Gamma(\pi^0 \to \gamma\gamma) = \frac{\alpha^2 m_{\pi^0}}{64\pi^3 f_\pi^2} \approx 7.8 \text{ eV}$$

The branching ratio is:

$$\text{BR}(\pi^0 \to \gamma\gamma) = \frac{\Gamma(\pi^0 \to \gamma\gamma)}{\Gamma_{\text{total}}} \approx 98.8\%$$

In our simulation we are only considering this decay mode, and ignoring the other decays such as the Dalitz decay ($\pi^0 \to e^+e^-\gamma$) for example.

2 Lorentz product, invariant masses, and Lorentz Transformation

2.1 Lorentz product

The Lorentz product/Minkowski inner product of two four-vectors $A^\mu$and B^μ is:

$$A \cdot B = A^\mu B_\mu = A^0 B^0 - \vec{A} \cdot \vec{B} = A^0 B^0 - A^1 B^1 - A^2 B^2 - A^3 B^3$$

```python
def lorentz_product(A, B):
    """Computes the Lorentz product of two four-vectors."""
    return A[0] * B[0] - np.dot(A[1:], B[1:])
```

To test the Lorentz product, we can use some known properties and simple cases:

1. Self-product gives squared invariant mass:

$$P \cdot P = m^2$$

```python
#testing the functions
P = np.array([5, 3, 4, 0])  # E = 5, p = (3,4,0)
expected_mass_squared = 5**2 - (3**2 + 4**2)  # Should be 0 (massless
    case)
computed_mass_squared = lorentz_product(P, P)
print(computed_mass_squared, expected_mass_squared,
    expected_mass_squared-computed_mass_squared)
assert np.isclose(computed_mass_squared, expected_mass_squared), "
    Invariant mass test failed!"
```

Output:

```
        0 0 0
```

2. We can also check if two orthogonal four-vectors A^μ and B^μ have a Lorentz product of zero:

$$A \cdot B = 0$$

```python
A = np.array([2, 1, 1, 0])
B = np.array([1, 1, -1, 0])  # Manually chosen so that their Lorentz
    product is 0
assert np.isclose(lorentz_product(A, B), 0), "Orthogonality test failed
    !"
```

Output:

```
        0 0 0
```

and lastly, can check that the Lorentz product remains unchanged after a Lorentz boost:

```python
P = np.array([10, 3, 4, 0])
velocity = np.array([0.6, 0, 0])  # Boost along x-direction
P_boosted = lorentz_transform(P, velocity)
print(lorentz_product(P, P), lorentz_product(P_boosted, P_boosted))
assert np.isclose(lorentz_product(P, P), lorentz_product(P_boosted,
    P_boosted)), "Lorentz invariance test failed!"
```

Output:

```
        75 75.0
```

So, the functions are all behaving like we expect them to. We can then say, this product is invariant under Lorentz transformations meaning it has the same value in all inertial frames of references.

2.2 Invariant mass

The invariant mass of a particle with four-momentum vector $P^\mu = E, \vec{P}$ is defined as:

$$m^2c^2 = P \cdot P = P^\mu P_\mu = P^0P^0 - \vec{P} \cdot \vec{P} = E^2 - \vec{P}^2$$
$$m^2c^2 = E^2 - \vec{p} \cdot \vec{p}$$

and with $c = 1$: $m^2 = E^2 - \vec{p} \cdot \vec{p}$

For a pair of particles with four-momentum vectors P_1^μ and P_2^μ, with system mass M:

$$M^2 = (P_1 + P_2)^2 = (P_1 + P_2) \cdot (P_1 + P_2)$$
$$M^2 = P_1 \cdot P_1 + P_2 \cdot P_2 + 2P_1 \cdot P_2$$
$$M^2 = m_1^2 + m_2^2 + 2P_1 \cdot P_2$$

where m_1 and m_2 are the invariant masses of the two particles.

The code for the invariant mass and system mass:

```python
def invariant_mass(P):
    """Computes the invariant mass of a four-momentum vector P."""
    return np.sqrt(P[0]**2 - np.dot(P[1:], P[1:]))

def system_mass(P1, P2):
    """Computes the invariant mass of a system of two particles."""
    P_total = P1 + P2
    return np.sqrt(lorentz_product(P_total, P_total))
```

```python
def test_invariant_mass():
    P = np.array([5, 3, 4, 0])
    expected_mass = np.sqrt(5**2 - (3**2 + 4**2))
    assert np.isclose(invariant_mass(P), expected_mass), "Invariant
        mass test failed!"
```

Output:

```
Invariant mass test passed!
```

2.3 Lorentz Transformation

The Lorentz transformation matrix for a boost in an arbitrary direction $\mathbf{v} = (v_x, v_y, v_z)$ is given by:

$$\Lambda^{\mu}{}_{\nu} = \begin{pmatrix} \gamma & -\gamma\beta n_x & -\gamma\beta n_y & -\gamma\beta n_z \\ -\gamma\beta n_x & 1+(\gamma-1)n_x^2 & (\gamma-1)n_x n_y & (\gamma-1)n_x n_z \\ -\gamma\beta n_y & (\gamma-1)n_y n_x & 1+(\gamma-1)n_y^2 & (\gamma-1)n_y n_z \\ -\gamma\beta n_z & (\gamma-1)n_z n_x & (\gamma-1)n_z n_y & 1+(\gamma-1)n_z^2 \end{pmatrix}$$

where $\beta = \frac{v}{c}$, $\gamma = \frac{1}{\sqrt{1-\beta^2}}$, and $\mathbf{n} = (n_x, n_y, n_z) = \frac{\mathbf{v}}{v}$.

The code for this is:

```python
def lorentz_transform(P, v):
    """Applies a Lorentz boost to a four-vector P in the direction of
        velocity v."""
    v = np.array(v)
    v_mag = np.linalg.norm(v)
    if v_mag == 0:
        return P  # No boost needed

    beta = v_mag / 1  # Assume c=1
    gamma = 1 / np.sqrt(1 - beta**2)
    n = v / v_mag  # Normalize direction

    # Lorentz transformation matrix
    Lambda = np.array([
        [gamma, -gamma * beta * n[0], -gamma * beta * n[1], -gamma *
           beta * n[2]],
        [-gamma * beta * n[0], 1 + (gamma - 1) * n[0]**2, (gamma - 1) *
            n[0] * n[1], (gamma - 1) * n[0] * n[2]],
        [-gamma * beta * n[1], (gamma - 1) * n[1] * n[0], 1 + (gamma -
           1) * n[1]**2, (gamma - 1) * n[1] * n[2]],
        [-gamma * beta * n[2], (gamma - 1) * n[2] * n[0], (gamma - 1) *
            n[2] * n[1], 1 + (gamma - 1) * n[2]**2]
    ])
```

```python

    return np.dot(Lambda, P)
```

We have already verified the Lorentz transform using our Lorentz product. So we can just refer to that for the check of this function.

3 Two Body Decay of a particle of mass M at rest

When a particle of mass M decays into two particles with mass m_1 and m_2, the total energy and momentum must be conserved. In the rest frame of the parent particle, the total energy before the decay is M and the total momentum before decay is 0 . As we discussed before, the decay products will move in opposite directions with equal and opposite momenta.

Using the energy-momentum relation: $E^2 - p^2 = m^2$, we can find the energy and momentum of the decay products.

$$E_1 = \frac{M^2 + m_1^2 - m_2^2}{2M}, \quad E_2 = \frac{M^2 + m_2^2 - m_1^2}{2M}$$

The magnitude of the momenta are then:

$$p = \sqrt{E_1^2 - m_1^2} = \sqrt{E_2^2 - m_2^2}$$

To ensure we get an isotropic decay (equal probability in all directions), we can sample the polar angle θ from a uniform distribution in $\cos\theta = 2\text{rand}() - 1$ (this gives us values from [-1,1]) and the azimuthal angle ϕ uniformly from 0 to 2π.

Then we convert to cartesian coordinates:

$$p_x = p\sin\theta\cos\phi, \quad p_y = p\sin\theta\sin\phi, \quad p_z = p\cos\theta$$

These are the momenta of the decay product of the first particle and the second particle is simply the negative of the first.

```python
def two_body_decay(M, m1, m2):
    """Generates two random four-momenta for a two-body decay in the
       rest frame of the decaying particle."""
    E1 = (M**2 + m1**2 - m2**2) / (2 * M)
    E2 = (M**2 + m2**2 - m1**2) / (2 * M)
    p_mag = np.sqrt(E1**2 - m1**2)  # Magnitude of the momentum

    # Generate a random isotropic direction
    theta = np.arccos(2 * np.random.rand() - 1)  # Uniform in cos(theta
       )
    phi = 2 * np.pi * np.random.rand()  # Uniform in phi

    p1 = np.array([E1, p_mag * np.sin(theta) * np.cos(phi), p_mag * np.
       sin(theta) * np.sin(phi), p_mag * np.cos(theta)])
    p2 = np.array([E2, -p1[1], -p1[2], -p1[3]])  # Opposite direction

    return p1, p2
```

We did not need to know the matrix element to compute the kinematics of the two-body decay because we are dealing with conservation laws and relativistic constraints. This allows to get the momentum distribution.

If we were to apply dynamics with the Lagrangian, we would need to find the matrix element. If we wanted to generate more correct physics-based events, we would need to consider the actual probabilistic distribution of the decays rather than assuming our events are uniform.

4 Mass M moving with velocity v

After calculating the momentum of the daughter particles in the rest frame, we need to boost to the lab frame to each daughters four-momentum.

$$P'^{\mu} = \Lambda^{\mu}_{n} u P^{\nu}$$

```
def two_body_decay(M, m1, m2, v):
    """Generates two random four-momenta for a two-body decay in the
       rest frame of the decaying particle,
    then boosts them to the lab frame where the decaying particle moves
        with velocity v."""
    E1 = (M**2 + m1**2 - m2**2) / (2 * M)
    E2 = (M**2 + m2**2 - m1**2) / (2 * M)
    p_mag = np.sqrt(E1**2 - m1**2)  # Magnitude of the momentum

    # Generate a random isotropic direction
    theta = np.arccos(2 * np.random.rand() - 1)  # Uniform in cos(theta
       )
    phi = 2 * np.pi * np.random.rand()  # Uniform in phi

    p1 = np.array([E1, p_mag * np.sin(theta) * np.cos(phi), p_mag * np.
       sin(theta) * np.sin(phi), p_mag * np.cos(theta)])
    p2 = np.array([E2, -p1[1], -p1[2], -p1[3]])  # Opposite direction

    # Boost to the lab frame
    p1_lab = lorentz_transform(p1, v)
    p2_lab = lorentz_transform(p2, v)

    return p1_lab, p2_lab
```

Some Validation of the Decays

We take some random 4-momenta decay the pion and check the conservation of energy and momentum of the products.

$$E_{\pi^0} = E_1 + E_2 \vec{p}_{\pi^0} = \vec{p}_1 + \vec{p}_2$$

To see how well these hold, compute the energy and momentum differences and plot their distributions:

$$E_{\pi^0} - (E_1 + E_2) |\vec{p}_{\pi^0} - (\vec{p}_1 + \vec{p}_2)|$$

```
# Conservation checks
df["energy_diff"] = df["E"] - (df["E1"] + df["E2"])
df["momentum_diff"] = np.sqrt(
    (df["px"] - (df["px1"] + df["px2"]))**2 +
    (df["py"] - (df["py1"] + df["py2"]))**2 +
    (df["pz"] - (df["pz1"] + df["pz2"]))**2
```

```
)

# Plot energy and momentum conservation
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].hist(df["energy_diff"], bins=100, color="blue", alpha=0.7)
axes[0].set_title("Energy Conservation: $E_{\pi^0} - (E_1 + E_2)$")
axes[0].set_xlabel("Energy Difference [GeV]")
axes[0].set_ylabel("Count")

axes[1].hist(df["momentum_diff"], bins=100, color="red", alpha=0.7)
axes[1].set_title("Momentum Conservation: $|\\vec{p}_{\pi^0} - (\\vec{p
    }_1 + \\vec{p}_2)|$")
axes[1].set_xlabel("Momentum Difference [GeV/c^2]")
axes[1].set_ylabel("Count")

plt.tight_layout()
plt.show()
```

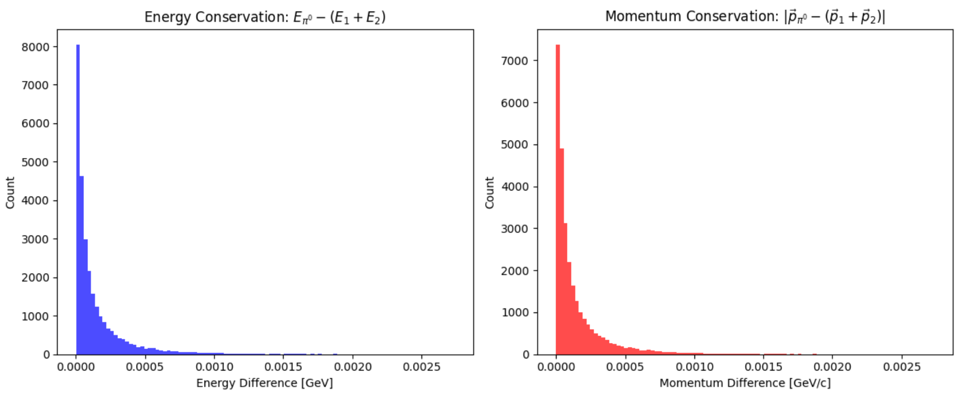

Figure 1: Histograms showing the distribution of energy and momentum differences for the two-body decay.

These graphs can be applied to both the decay of the particles at rest or in motion. The results are the same and we have conservation of energy and momentum (within the range of numerical precision). There is additional validation of almost all the functions and simulation in the code.

5 Simulation of the Two-Body Decay, $\pi^0 \to \gamma\gamma$

The following is a simulation of the two-body decay. It is a vectorized version of the previous code so it can handle a large number of events efficiently. The key steps are:

1. Initialization of variables and arrays
2. For each π^0 in pi0_momenta, perform a two-body decay to simulate the production of two photons using the two_body_decay function. We vectorized this function to handle a large number of events efficiently.
3. Boost to Lab Frame using the vectorzed_lorentz_transform function

4. Photon Check: an event is detected if at least one of the two photons hit the detector
5. Calculate Fraction

```
def vectorized_lorentz_transform(p, v):
    """
    Perform a Lorentz boost on a batch of 4-vectors.

    Parameters
    ----------
    p : numpy.ndarray
        Array of 4-vectors in the rest frame, shape (N, 4), with columns [E, px, py, pz].
    v : numpy.ndarray
        Array of boost velocities, shape (N, 3). (Assuming units where c=1.)

    Returns
    -------
    numpy.ndarray
        Boosted 4-vectors in the lab frame, shape (N, 4).
    """
    # Compute beta^2 for each event (v^2)
    beta2 = np.sum(v**2, axis=1, keepdims=True)  # shape: (N,1)
    # Calculate gamma factor for each event
    gamma = 1.0 / np.sqrt(1.0 - beta2)              # shape: (N,1)
    # Dot product between the spatial momentum and the boost velocity
    bp = np.sum(p[:, 1:] * v, axis=1, keepdims=True)  # shape: (N,1)

    # Calculate the factor (gamma - 1)/beta2 safely. Set to 0 when beta2 is nearly zero.
    with np.errstate(divide='ignore', invalid='ignore'):
        gamma2 = np.where(beta2 > 1e-12, (gamma - 1.0) / beta2, 0.0)  # shape: (N,1)

    # Boost the spatial components:
    # p[:, 1:] is (N,3) and v is (N,3); the operations are elementwise.
    spatial = p[:, 1:] + gamma2 * bp * v + gamma * p[:, :1] * v  # shape: (N,3)
    # Boost the energy component:
    energy = gamma * (p[:, :1] + bp)  # shape: (N,1)

    # Concatenate boosted energy and spatial parts along axis 1.
    boosted = np.hstack([energy, spatial])
    return boosted

def estimate_detection_fraction_vectorized(pi0_momenta, M, detector_size=1.0, distance=10.0):
    """
    Estimate the fraction of \pi^0 decays for which at least one photon hits the detector.

    The detector is assumed to be a square centered at (x, y) = (0, 0) in a plane
    located at z = distance. The size of the detector is detector_size (length of a side).

```

```
    Parameters
    ----------
    pi0_momenta : numpy.ndarray
        Lab frame 4-momenta of the \pi^0 particles (shape: [N,4] with
            columns [E, px, py, pz]).
    M : float
        Mass of the \pi^0 particle (in GeV/c^2).
    detector_size : float, optional
        Size of the detector (side length), by default 1.0.
    distance : float, optional
        z-distance to the detector plane, by default 10.0.

    Returns
    -------
    float
        The estimated geometric efficiency (fraction of decays detected
            ).
    """
    n_decays = len(pi0_momenta)

    # Generate random angles for isotropic decays in the pion rest
       frame.
    cos_theta = 2 * np.random.rand(n_decays) - 1
    theta = np.arccos(cos_theta)
    phi = 2 * np.pi * np.random.rand(n_decays)

    # In the rest frame of the \pi^0, energy = |p| for a photon. Each
       photon gets half the mass.
    p_mag = M / 2.0

    # Photon 1 in the rest frame: 4-vector [E, px, py, pz]
    p1_rest = np.column_stack([
        np.full(n_decays, p_mag),                # Energy
        p_mag * np.sin(theta) * np.cos(phi),
        p_mag * np.sin(theta) * np.sin(phi),
        p_mag * cos_theta
    ])

    # Photon 2 has exactly the opposite momentum.
    p2_rest = np.column_stack([
        np.full(n_decays, p_mag),                # Energy
        -p_mag * np.sin(theta) * np.cos(phi),
        -p_mag * np.sin(theta) * np.sin(phi),
        -p_mag * cos_theta
    ])

    # Compute the boost velocity of the \pi^0: v = p/E
    # pi0_momenta has columns [E, px, py, pz]
    v = pi0_momenta[:, 1:] / pi0_momenta[:, :1]  # shape: (N,3)

    # Boost both photons to the lab frame.
    p1_lab = vectorized_lorentz_transform(p1_rest, v)
    p2_lab = vectorized_lorentz_transform(p2_rest, v)

    def check_hits(p):
        """
        Check if the boosted photon (p) hits the detector.

```

```
        p is expected to be a numpy array of shape (N,4) for lab-frame
            4-vectors.
        """
        # We are interested in the photon's z-component; if it is
            nearly zero or negative (moving backward), it won't hit.
        pz = p[:, 3]
        nonzero = np.abs(pz) > 1e-8
        # For photons moving toward the detector (i.e. pz > 0), compute
             the parametric distance t such that z = distance.
        # For non-moving or backward-moving photons, set t = infinity.
        t = np.where((pz > 0) & nonzero, distance / pz, np.inf)
        # The photon hits at positions (x_hit, y_hit)
        x_hit = p[:, 1] * t
        y_hit = p[:, 2] * t
        # Check if (x_hit, y_hit) fall within half the detector size.
        hit = (np.abs(x_hit) <= detector_size / 2.0) & (np.abs(y_hit)
            <= detector_size / 2.0)
        # Only count if the photon is moving forward.
        hit = hit & (pz > 0)
        return hit

    hits1 = check_hits(p1_lab)
    hits2 = check_hits(p2_lab)

    # An event is detected if at least one of the two photons hits the
        detector.
    detected = hits1 | hits2
    fraction_detected = np.mean(detected)
    return fraction_detected
```

Here we apply the functions defined above for our 120 GeV momenta array.

```
if __name__ == '__main__':
    # Define the \pi^0 mass (in GeV/c^2)
    M_pi0 = 0.1349768
    n_events = 100000  # Number of decays to simulate

    # Generate random lab frame \pi^0 momenta.
    # We'll sample the spatial momentum components from a normal
        distribution.
    lab_momenta_spatial = np.random.normal(0, 1, (n_events, 3))
    # Compute the energy using E = sqrt(p^2 + m^2)
    energies = np.sqrt(np.sum(lab_momenta_spatial**2, axis=1) + M_pi0
        **2)
    # Construct the \pi^0 momenta array with columns [E, px, py, pz]
    pi0_momenta = np.column_stack([energies, lab_momenta_spatial])

    # Estimate the geometric efficiency:
    fraction = estimate_detection_fraction_vectorized(momenta_array,
        M_pi0,
                                                        detector_size
                                                            =1.0,
                                                        distance=10.0)
    print(f"Geometric efficiency: {fraction:.6f}")
    print(f"Expected number of detected events: {fraction * n_events:.0
        f}")
```

We get outputs of around 0.36 or 36%.

We also apply the functions for a isotropic distribution of momenta and get a similar result of 0.16%

```python
if __name__ == '__main__':
    # Define the \pi^0 mass (in GeV/c^2)
    M_pi0 = 0.1349768
    n_events = 100000  # Number of decays to simulate

    # Generate random lab frame \pi^0 momenta.
    # We'll sample the spatial momentum components from a normal
        distribution.
    lab_momenta_spatial = np.random.normal(0, 1, (n_events, 3))
    # Compute the energy using E = sqrt(p^2 + m^2)
    energies = np.sqrt(np.sum(lab_momenta_spatial**2, axis=1) + M_pi0
        **2)
    # Construct the \pi^0 momenta array with columns [E, px, py, pz]
    pi0_momenta = np.column_stack([energies, lab_momenta_spatial])

    # Estimate the geometric efficiency:
    fraction = estimate_detection_fraction_vectorized(pi0_momenta,
        M_pi0,
                                                     detector_size
                                                        =1.0,
                                                     distance=10.0)
    print(f"Geometric efficiency: {fraction:.6f}")
    print(f"Expected number of detected events: {fraction * n_events:.0
        f}")
```

5.1 Changing the detector coverage

The probability a photon hits the detector is determined by:

$$P_{\text{hit}\approx\frac{\text{Detector Area}}{4\pi\times(\text{Distance})^2}}$$

Our original setup had a detector of $1\,m \times 1\,m$ at $10\,m$ which had a solid angle $\Delta\Omega = \frac{1}{10^2} = 0.01\,sr$ the total solid angle is then $4\pi \approx 12.566\,sr$ This gives us a probability per photon of :

$$\frac{0.01}{12.566} \approx 0.000796\,(0.08\%)$$

- ▷ Probability for 2 photons:

$$1 - (1 - 0.000796)^2 \approx 0.00159\,(0.16\%)$$

Effect of Detector Size

1. Smaller Detector (e.g., $0.1\,m \times 0.1\,m$):
   - ▷ Area: $0.01\,m^2 \rightarrow \Delta\Omega = \frac{0.01}{100} = 0.0001$ sr
   - ▷ Probability per photon: $\frac{0.0001}{12.566} \approx 8 \times 10^{-6}$
   - ▷ Expected fraction: $\approx 1.6 \times 10^{-5}\,(0.0016\%)$
2. Larger Detector (e.g., $10\,m \times 10\,m$):
   - ▷ Area: $100\,m^2 \rightarrow \Delta\Omega = \frac{100}{100} = 1\,sr$
   - ▷ Probability per photon: $\frac{1}{12.566} \approx 0.0796$
   - ▷ Expected fraction: $\approx 1 - (1 - 0.0796)^2 \approx 0.153\,(15.3\%)$

5.2 Simulation Results

To estimate the geometric acceptance, also known as geometric efficiency, of a hypothetical detector for detecting photons resulting from the decay of neutral pions ($\pi^0 \to \gamma\gamma$). The detector is positioned 10 meters downstream from the target, featuring a 1m x 1m front face.

Simulation Setup

The simulation was conducted using a vectorized process to model the two-body decay of π^0 particles into two photons. Initially, the momenta of the π^0 were sourced from a dataset simulating proton-target collisions. These momenta were used as inputs to model the decay events in the rest frame of the π^0 particles.

Lorentz Boost to the Lab frame

Photons arising from these decays were subject to relativistic transformations, as each photon pair was translated from the π^0 rest frame to the lab frame using Lorentz boosts. A vectorized method was employed for the transformations, allowing efficient computation across the entire dataset.

Detection Criteria

For each photon, its trajectory was assessed to determine whether it intersected with the detector's plane at $z = 10m$. An intersection was counted as a 'hit' if the coordinates at intersection remained within the $[-0.5m, 0.5m]$ bounds of the detector face in both x and y dimensions.

The fraction of decays wherein at least one photon intersected with the detector plane defines the geometric acceptance. Across the simulations, the geometric efficiency was calculated as follows:

$$\text{Geometric Efficiency} = \frac{\text{Number of Hits}}{\text{Total Number of Decays}}$$

Results and Interpretation

The simulation resulted with varying output due to random chance. In the table below, are the results of the geometric efficiency for different outputs with random seeds.

Geometric Efficiency
0.367154
0.366336
0.367865

Table 1: Geometric Efficiency for 120 GeV decays provided by the data file

Geometric Efficiency
0.001670
0.001480
0.001540

Table 2: Geometric Efficiency for Isotropic Decays

When using random isotropic decays, we find the geometric efficiency to be approximately 0.0016. This aligns with the theoretical prediction of 0.16%.

However, when using the data file provided, which came from π^0 with average momenta of 120 GeV, we find the geometric efficiency to be approximately 0.367. This is significantly higher than the theoretical prediction of 0.16%. This discrepancy is likely due to the fact that the data file provided is not isotropic, and thus the geometric efficiency is higher.

We can see that the data is not isotropic by simply seeing that the average energy of the photons is 6 GeV and has maximum energy of 108 GeV.

```
print(E.mean())
print(E.max())
```

output

```
6.031258107431741
108.53296975773868
```

compared to

```
1.602811772293395
5.149393972687484
```

of the isotropic data

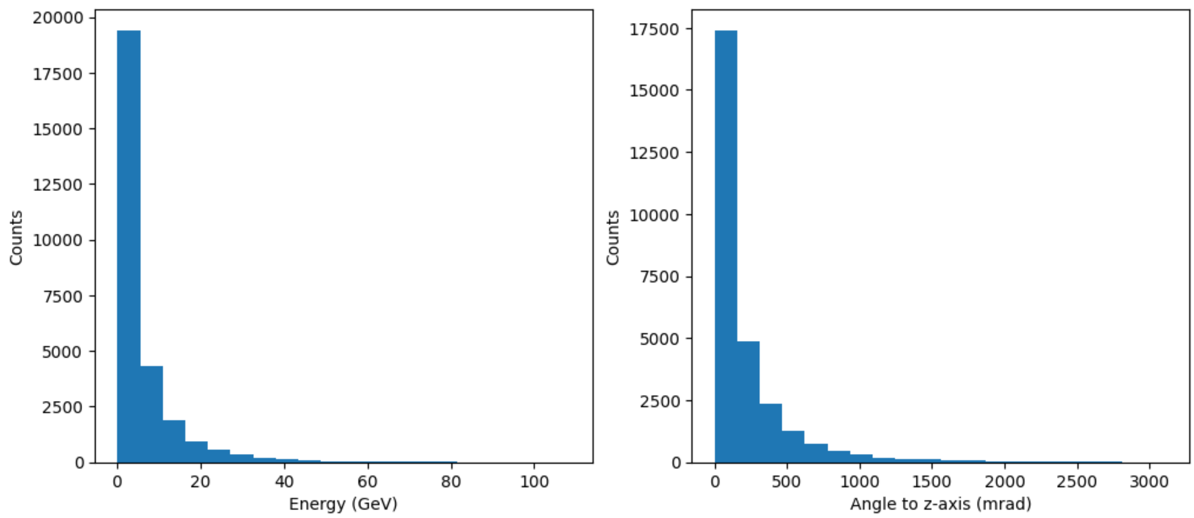

Figure 2: Distribution of the 120 GeV Decay

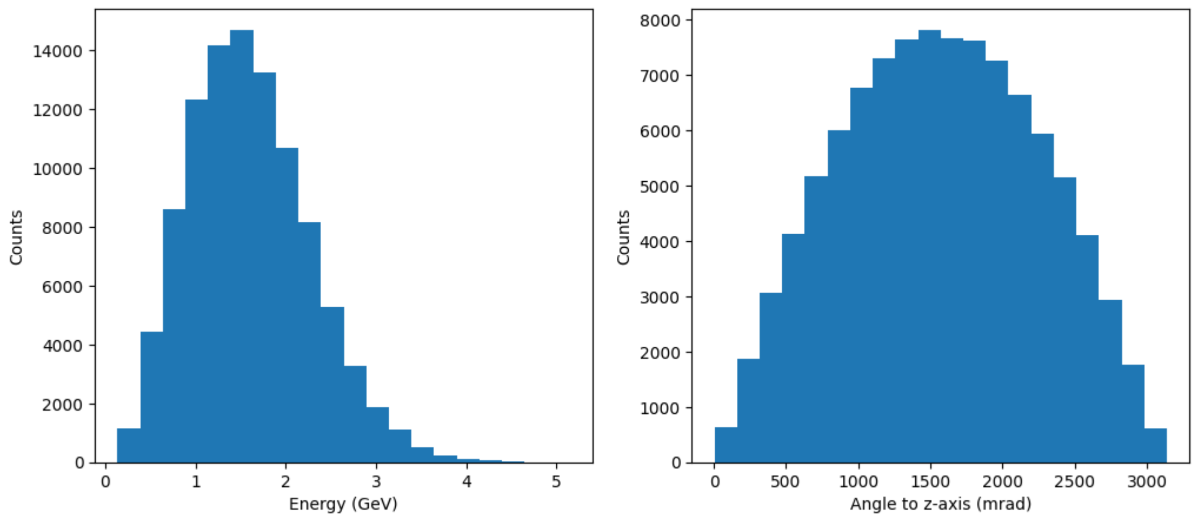

Figure 3: Distribution of the Isotropic Decay

It is clear that the initial momenta and angles of the π^0 particles significantly influenced the likelihood of detection, demonstrating the sensitivity of geometric acceptance to the kinematic conditions of particle decays.

5.3 Changing Detector Sizes

```
results = []
configurations = [
    {"detector_size": 10.0, "distance": 1.0},
    {"detector_size": 10.0, "distance": 10.0},
    {"detector_size": 10.0, "distance": 100.0},
    {"detector_size": 0.1, "distance": 1.0},
    {"detector_size": 0.1, "distance": 10.0},
    {"detector_size": 0.1, "distance": 100.0},
]

for config in configurations:
    detector_size = config["detector_size"]
    distance = config["distance"]
    fraction = estimate_detection_fraction_vectorized(pi0_momenta, M_pi0,
                                                      detector_size=detector_size,
                                                      distance=distance)
    results.append({
        "detector_size": f"{detector_size:.1f}m x {detector_size:.1f}m",
        "distance": f"{distance:.1f}m",
        "efficiency": f"{fraction:.6f}"
    })

# Display results in a nice table
print("| Detector Size      | Distance | Geometric Efficiency |")
```

```python
print("|----------------------|----------|----------------------|")
for result in results:
    print(f"| {result['detector_size']:<20} | {result['distance']:<8} | {result['efficiency']:<20} |")
```

Detector Size	Distance (m)	Geometric Efficiency
1.0m x 1.0m	10.0	0.369180980831
10.0m x 10.0m	1.0	0.991713787830
10.0m x 10.0m	10.0	0.912016785803
10.0m x 10.0m	100.0	0.370852448522
0.1m x 0.1m	1.0	0.368398591700
0.1m x 0.1m	10.0	0.021551264270
0.1m x 0.1m	100.0	0.000284505139

Table 3: Geometric Efficiency for Different Detector Configurations

and for the isotropic data:

Detector Size	Distance (m)	Geometric Efficiency
1.0m x 1.0m	10.0	0.001630000000
10.0m x 10.0m	1.0	0.458140000000
10.0m x 10.0m	10.0	0.086440000000
10.0m x 10.0m	100.0	0.001460000000
0.1m x 0.1m	1.0	0.001510000000
0.1m x 0.1m	10.0	0.000010000000
0.1m x 0.1m	100.0	0.000000000000

Table 4: Geometric Efficiency for Different Detector Configurations

References

[1] David J. Griffiths. *Introduction to Elementary Particles*. Wiley-VCH, 2 edition, 2008.

[2] Particle Data Group. Api. Accessed: 2025-04-09.

[3] Pasayten Institute. Neutral pion — the field guide to particle physics, 2023. Accessed: 2025-04-09.

[4] Mark Thomson. *Modern Particle Physics*. Cambridge University Press, 2013.

[5] Esther Weil, Gernot Eichmann, Christian S. Fischer, and Richard Williams. Electromagnetic decays of the neutral pion. *Physical Review D*, 96(1), July 2017.

[6] Wikipedia contributors. Pion — wikipedia, the free encyclopedia, 2025. Accessed: 2025-04-09.

[7] Simon Zimmermann. *Study of neutral pion production in high energy proton-proton collisions at the LHC*. Phd thesis, University of Heidelberg, 2013.