Simulation of ALICE Project

Coli Simone

November 24th, 2021

1 Introduction

ALICE_Simulation is a computer program that simulates the ALICE experiment conducted at CERN since 1993, which consists of analyzing the collisions occurring between particles at very high energies$^{(1)}$. Those would create different particles following a probability distribution (Table1).

Particle Type	Probability
π^+	40%
π^-	40%
k^+	5%
k^-	5%
P^+	4.5%
P^-	4.5%
k^*	1%

Table 1: *The table shows the probability of obtaining a specific particle from a collision of high energy particles*

In the simulation, we generated a finite amount of particles resulting from the collision of the flux in the particle accelerator, each with a proper momentum, mass, and resulting energy to maintain the conservation of those properties.The goal of the experiment is to detect the existence of the Kaon 0 (k^*), a very rare particle that decays into either a Pion plus (π^+) and Kaon minus (k^-) or a Pion minus (π^-) and Kaon plus (k^+), after only $5.2 \times 10^{-8}s$. We so stored the data of momentum, energy, charge, and invariant mass of all the particles to detect the presence of differences that could indicate the existence of the Kaon.

The program we implemented stores the information about the particles and generates histograms from which we studied the system.

2 Code Structure

The code's division into different files and folders has the background idea of making it more orderly. There are eight files for the simulation program (one main file:

mainE.cpp, one libraries file: library.hpp, three header files: ParticleType.hpp, ResonanceType.hpp, Particle.hpp, and three source files: ParticleType.cpp, ResonanceType.cpp, Particle.cpp) and one for the data analysis (analysis.C). The header files contain the classes implemented for the proper functioning of the simulation. Meanwhile, the source files contain the implementation of the methods defined in the headers.

ParticleType Class

The class `ParticleType` creates a homonymous type that contains the name, the mass, and the charge of a particular particle, respectively as a `std::string`, a `double`, and an `integer`. This class has five methods, two of which are virtual.

ResonanceType Class

The `ResonanceType` class is a derived class from `ParticleType` and, in addition to the base class items, it contains the information about the width of the particle as a double. The type defined with the name of this class creates a particle with a width. Contrarily to what happens for a `ParticleType` object, in which the width of the particle is always zero. This class has two methods, both of them are the override of the already existing virtual methods in the base class.

Particle Class

The class `Particle` is the one that allows creating a particle giving it a random momentum and making it decay into other particles if necessary. It also creates a set of particles type, each with a proper index as an identifier. The variables in the class are three momentum components (`fPx`, `fPy`, `fPz`), an array of ParticleType and its dimension (`fParticleType`, `fMaxNumParticleType`), an index of particle type (`fNParticleType`), and a numeric code proper of each particle type `fIndex`). This class has, also, several methods, including some static, which means they are accessible from the main without defining an object.

3 Generation

In the simulation, there had been 10000 collision events, each using a set of 100 particles. The particles resulting from the collisions were Positive Pions (π^+), Negative Pions (π^-), Positive Kaons (k^+), Negative Kaons (k^-), Protons (p^+), Antiprotons (p^-), and Resonance Kaons (k^*), generated randomly using a uniform distribution and the probability shown in Table 1. The module of the momentum of the particles comes from an exponential distribution with a mean of 1. While its direction drives from the cartesian components:

$$\begin{cases} p_x = |p| \cdot \cos\theta \cdot \cos\phi \\ p_y = |p| \cdot \cos\theta \cdot \sin\phi \\ p_z = |p| \cdot \sin\theta \end{cases} \tag{1}$$

Where the azimuthal angle theta (θ) and the polar angle phi (ϕ) are generated using a uniform random distribution respectively from 0 to π and the second from 0 to 2π. In the case that a Resonance Kaon is created from the collision of particles, it deacis into eather a Positive Pion and a Negative Kaon or a Negative Pion and a Positive Kaon with the same probability. The momentum of this new two particles comes from a normal distribution.

4 Analysis

The generation of particles types is compatible with the theoretical calculation as shown in Table. 2

Particle Type	Entries	Error	Theo. Ent.
π^+	3997740	1999	$4 \cdot 10^6$
π^-	4002160	2001	$4 \cdot 10^6$
k^+	500290	707	$5 \cdot 10^5$
k^-	499781	707	$5 \cdot 10^5$
P^+	449544	671	$4.5 \cdot 10^5$
P^-	450121	671	$4.5 \cdot 10^5$
k^*	100366	317	$1 \cdot 10^5$

Table 2: *The table shows the entries got from the simulation with the respective error calculated using root, and the theoretical entries calculated taking the percentage of each particle type from the total amount of particles.*

As well as the angles and the momentum distributios are fitting the relative uniform and exponential distribution as shown in the images 1 and 2.

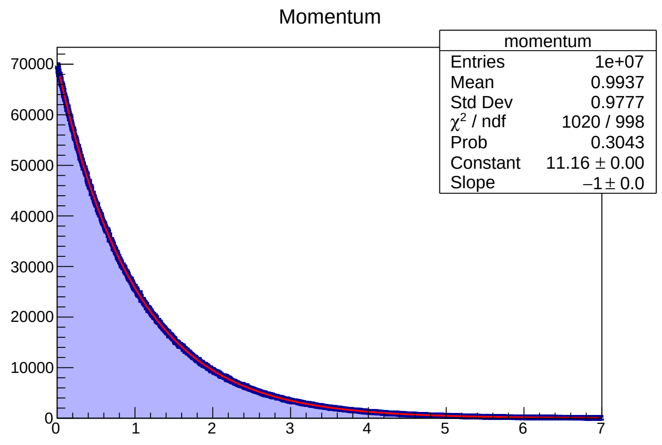

Figure 1: *The image shows the fit between the exponential distribution compared with the momentum histogram with occurrences randomly generated. The Chi-square divided by the number of degrees of freedom is very close to one, meaning that the histogram very well approximates an exponential distribution with a mean of 1.*

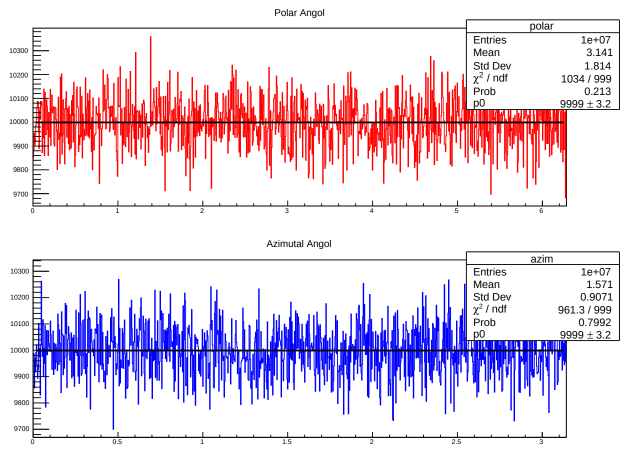

Figure 2: *The image shows the fit between the uniform distribution compared with the histogram of the angles randomly generated. The Chi-square divided by the number of degrees of freedom is very close to one, meanning that the histograms very well approximate an uniform distribution.*

Word Citacion

1. Alice Experiment. CERN. `https://home.cern/science/experiments/alice` . November 29th, 2021.