UPPSALA
UNIVERSITET

A Monte-Carlo Generator for Hyperons

Facilitating the Search for an Explanation of the Asymmetric Universe

Benjamin Bucknall & Kamilla Faizullina
Supervised by Prof. Karin Schönning

Project in Computational Science: Report

February 2022

PROJECT REPORT

Institutionen för informationsteknologi

Abstract

One of the major outstanding problems in contemporary physics is that of *baryon asymmetry* - the question of why our universe contains far more matter than antimatter. The phenomena of *baryogenesis* could have lead to this observed asymmetry but depends on the violation of *charge-parity symmetry*. This project proposes and implements two Monte-Carlo methods, Hit-and-Miss Monte-Carlo and the Metropolis algorithm, that can be used to generate simulated data regarding the decays of a hyperon-antihyperon pair that could aid in the search for evidence for the violation of charge-parity symmetry. The efficiency and accuracy of these two Monte-Carlo methods are assessed and it is found that, there is marginal practical difference between the two methods.

Contents

1 Introduction

One of the fundamental mysteries in modern physics is that of why there is so much more matter than antimatter in our universe, known as the problem of *baryon asymmetry*. According to the current paradigm, matter and antimatter should have been created in equal amounts in the Big Bang. How then, is it possible that the universe could end up in a state where it is dominated by matter, with no naturally occurring antimatter?

One approach to resolving this apparent contradiction is to question the physical symmetries between matter and antimatter to try to find situations in which they do not hold. This would then indicate a pathway through which the early symmetric universe could have evolved into that which we observe today. This report is the product of a project that aims to assist in this line of research by developing computational tools that can be used to study subatomic particle decay reactions that may exhibit violation of the aforementioned matter-antimatter symmetries.

The report takes the following structure. Section 2 summarises the problem of baryon asymmetry in more detail before providing a brief introduction to the relevant physics background, including the Standard Model of particle physics, and the concept of *baryogenesis*, a potential solution to baryon asymmetry. Section 3 continues by detailing a particular particle reaction which is being studied with the hope of detecting asymmetries in the relative behaviour of matter and antimatter. It also motivates the use and utility of computational tools in studying this reaction. Section 4 then changes course, introducing two variants of Monte-Carlo methods which were applied to the problem at hand. Section 5 details the particular implementations of these Monte-Carlo methods. Section 6 discusses how information relating to the physics parameters governing the reaction can be recovered from the data generated by the Monte-Carlo methods. Section 7 presents results pertaining to the accuracy and computational efficiency of the computational tools developed, which Section 8 discusses. Section 9 concludes.

2 Motivation

In this section, we describe the physics background that motivates the project. We start from the open question of matter-antimatter asymmetry in the observable universe, before providing a very brief overview of the Standard Model of particle physics. We then describe three theoretical conditions which must be met for the currently asymmetric universe to have been able to develop from a symmetric initial state, particularly focusing on the condition of charge-parity violation.

As discussed in the introduction, a major unresolved question in modern physics is that of baryon asymmetry, that is, why there is so much more matter than antimatter in the observable universe. Our current understanding of the origin and subsequent development of our universe is given by the Big Bang theory. According to this theory, the universe came into existence approximately 13 billion years ago and subsequently underwent a period of rapid expansion, evidence of which can still be observed today in the form of fluctuations on the cosmic microwave background radiation. Crucially however, all presently established models of the Big Bang and the early universe predict that equal amounts of matter and antimatter were created. This is puzzling given that we are yet to find evidence for the existence of primordial, naturally occurring antimatter [Ber02]. This poses the key question motivating this report and a vast body of contemporary research of *'Where did all the antimatter go?!'*

2.1 The Standard Model of particle physics

In order to be able to fully understand this key question, and the way in which the research addressed in this report aims to answer it, we must first cover some background theory in particle physics. The Standard Model of elementary particle physics[1], as depicted in Figure

[1] Also referred to as simply 'the Standard Model' or SM.

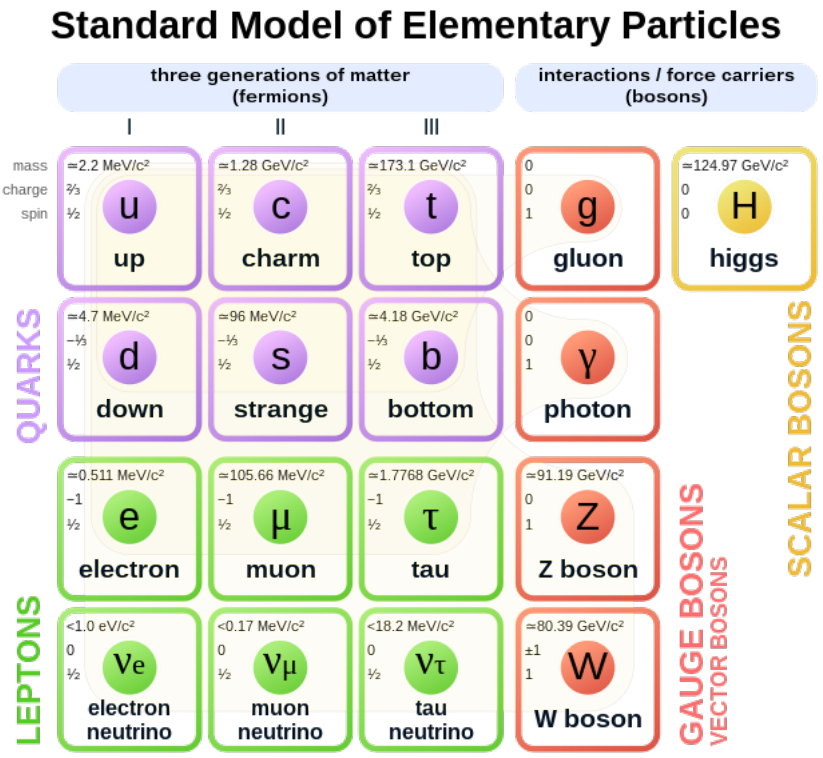

Figure 1: The Standard Model of Elementary Particles

1, is the theory governing the underlying elementary particles that constitute physical matter, as well as three of the four fundamental physical forces: electromagnetism, and the strong and weak nuclear forces[2]. More than 50 years of rigorous experimental particle physics research have produced results repeatedly confirming the Standard Model's theoretical predictions with extremely few exceptions. The Standard Model contains two classes of particles: *fermions*, which are generally thought of as being associated with matter; and *bosons* which generate the three fundamental forces mentioned above. Finally, the Higgs boson, the existence of which was predicted in 1964 by Peter Higgs and confirmed experimentally at the Large Hadron Collider at CERN in 2012, gives rise to the property of mass.

Within the fermion class of particles there is a further subdivision between *quarks* and *leptons*, the latter of which are distinguished by not being susceptible to the strong nuclear force. Furthermore, both quarks and leptons are separated into three *generations* of increasing mass. The first generation consists of the *up* and *down* quarks and the *electron* and *electron neutrino*; the second generation of the *charm* and *strange* quarks and the *muon* and *muon neutrino*; and the third generation contains the *top* and *bottom* quarks as well as the *tau* and *tau neutrino*. Each of the 12 fermions also have corresponding antiparticles, known as *antifermions* which have the same physical properties but with opposite *parity*, an intrinsic property of elementary particles.

Through the strong nuclear force, quarks are able to form composite particles known as *hadrons*, whereas leptons, being unaffected by the strong nuclear force, exist largely independently or, as in the case of the electron, as one of the key components of atoms. The final division that we will make is between two classes of hadron - *baryons*, consisting of a set of three quarks, and *mesons*, consisting of a quark and an antiquark pair. Baryons are what most matter consist of, most commonly through *protons*, made up of two up quarks and a down quark, and

[2]Consolidation of the Standard Model with the fourth fundamental force of gravity, as described by Einstein's theory of general relativity, is a major ongoing area of research.

neutrons, composed of one up and two down quarks, which together make up the nuclei of atoms. Less common baryons include quarks from higher generations and may be unstable, naturally tending to decay to lighter baryons, such as protons and neutrons. For example, Λ-hyperons consist of one up, one down, and one strange quark, and it is the decay of these hyperons into a proton and a type of meson known as a *pion* which is of central significance to this report. Note finally, that all baryons have corresponding antibaryons which are constructed by replacing each of the constituent quarks with their parallel antiquark.

2.2 Baryogenesis and the Sakharov conditions

We now return to the question of baryon asymmetry by introducing the notion of *baryogenesis* - the term used for a physical process which could have lead to the observed matter-antimatter asymmetry. In 1967, the Russian physicist Andrei Sakharov showed that the occurrence of baryogenesis was dependent on three necessary conditions [Sak91]. These are [Ber02, RT99]:

1. Baryon number violation,
2. Departure from thermal equilibrium,
3. Charge-parity violation.

These conditions deserve a fair amount of clarification. Firstly, the baryon number B of a system is defined through the formula

$$B = \frac{1}{3}(n_q - n_{\bar{q}}), \tag{1}$$

where n_q and $n_{\bar{q}}$ are the number of quarks and antiquarks in the system respectively. For example, a baryon, having three quarks, has a baryon number of $+1$, whereas an antibaryon has a baryon number of -1. Mesons on the other hand, have a baryon number of 0 since they contain one quark and one antiquark. While there is currently no evidence of processes that do not conserve the baryon number of the system, if we consider the entire universe then this conservation must be clearly violated if we are to get from an initial state with $B = 0$, i.e., an equal number of quarks and antiquarks, to our currently highly asymmetric universe.

The second condition, that of departure from thermal equilibrium, is less intuitive and would require a somewhat more technical explanation than baryon number violation. At a high level however, this condition says that any baryon number-violating reaction must occur at a reaction rate that is less than that of the expansion of the universe. If this is not the case, then any products of the reaction will be able to annihilate each other and restore the system to one with conserved baryon number.

Finally, the third condition, and the one of most interest in this report, is that of the violation of charge-parity symmetry, referred to as *charge-parity violation*. Due to its significance to this project, we take the time to address this condition in more detail in the following subsection.

2.2.1 Charge-parity violation

Charge-parity symmetry (or simply CP-symmetry) is the physical symmetry claiming that particles and antiparticles obey identical yet opposite laws. Another way of looking at this is to say that the laws of physics should be identical under the combination of charge conjugation on the one hand, which replaces each particle with its corresponding antiparticle and vice-versa, with parity conjugation on the other. This latter transformation simply inverts the spatial coordinates of the system as if we were looking at its reflection in a mirror.

As an example, Figure 2 shows an electron moving with velocity $\vec{v}$ on the left, and a positron (the corresponding antiparticle of an electron) moving with velocity $-\vec{v}$ on the right. Thus, CP-symmetry states that these systems should evolve according to analogous physical laws in

such a way that, at any point, one can be mapped on to the other through performing both charge and parity-conjugation.

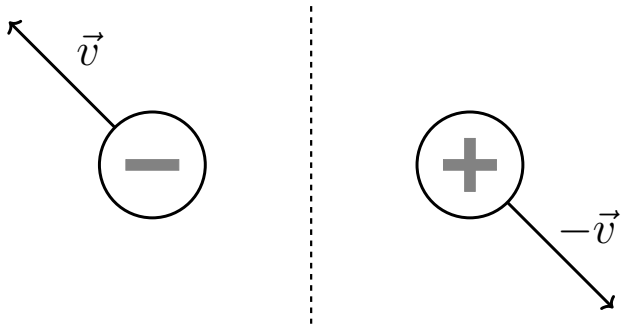

Figure 2: Charge-Parity symmetry states that the positron (right) should evolve according to the same physical laws as the electron (left).

We can then define charge-parity violation (CP-violation) as the violation of the law of CP-symmetry. This is intuitively necessary for baryogenesis to occur since, if CP-symmetry is indeed never violated, then any process that could lead to an excess of baryons over antibaryons would necessarily have a perfectly symmetric *'anti-process'* that would create an excess of antibaryons, cancelling out the initial process.

3 Formalism

With this background covered, we now introduce a specific particle reaction chain and describe how its study could provide evidence of CP-violation.

3.1 The reaction

The reaction in question, detailed in [FK17], is the head-on collision between an electron and a positron (e^-, e^+), producing a Λ-hyperon/antihyperon pair $(\Lambda/\bar{\Lambda})$, and the subsequent decays of the hyperon into a proton and a negative pion (p, π^-), and the antihyperon into an antiproton and a positive pion $(\bar{p}, \pi^+)$. These decays occur naturally on the order of 10^{-10} seconds after the creation of the hyperon-antihyperon pair. By colliding an electron with a positron in a controlled environment we are able to create the Λ-hyperon and $\bar{\Lambda}$-antihyperon in such a way that they move with equal and opposite velocities. Once this hyperon-antihyperon pair has been produced, each particle quickly decays into its respective daughter products. CP-symmetry should manifest itself in the resulting decay pattern, i.e., the direction of emission of these daughter products, due to the fact that the hyperon and antihyperon had opposite parities and momenta in the centre of mass reference frame. By inferring *'decay parameters'* that quantify the decay patterns for each of the hyperon and antihyperon we can test CP-symmetry; if the decay parameters are equal and opposite, then CP-symmetry holds, whereas if the parameters differ then this would imply CP-violation. Figure 3 shows a pictorial representation of the reaction in question which we will now discuss in more detail.

In Figure 3 we see the incoming electron and positron beams along the x-axis colliding in the centre of the figure, producing a Λ-hyperon/antihyperon pair which scatter with equal and opposite velocities. The scattering angle of the Λ-hyperon with respect to the incoming positron beam is denoted θ, which is identical to the angle between the Λ-antihyperon's velocity and the electron beam due to conservation of momentum. The Λ-hyperon then rapidly decays into a proton and negative pion, while the $\bar{\Lambda}$-antihyperon decays into an antiproton and a positive pion. Table 1 shows information regarding the particles involved in the reaction.

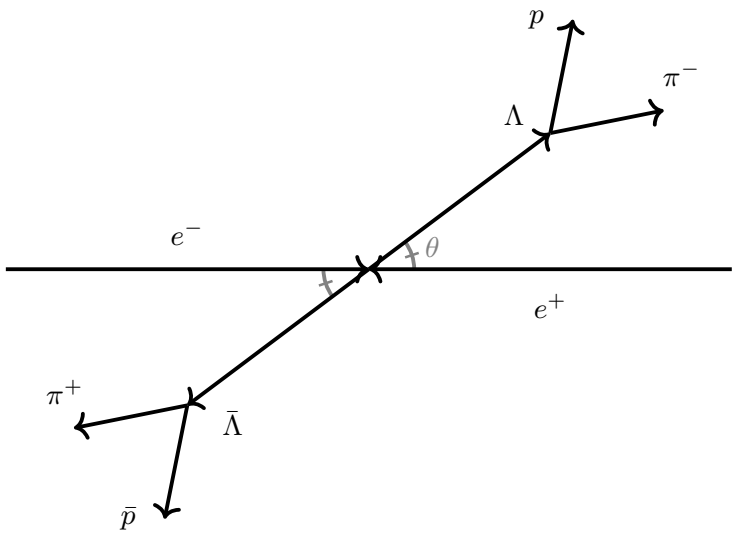

Figure 3: Graphical representation of the $e^+e^- \to \Lambda\bar{\Lambda}$ reaction. The collision of an e^- with a e^+ produces a $\Lambda/\bar{\Lambda}$ pair, which decay into (p, π^-) and $(\bar{p}, \pi^+)$ pairs respectively.

Particle	Symbol	Quark composition	Charge	Mass (MeV)	Baryon number	Spin	Parity
Electron	e^-	N/A	-1	0.511	0	$+\frac{1}{2}$	$+$
Positron	e^+	N/A	$+1$	0.511	0	$+\frac{1}{2}$	$-$
Λ-hyperon	Λ	uds	0	1115.683	$+1$	$+\frac{1}{2}$	$+$
Λ-antihyperon	$\bar{\Lambda}$	$\bar{u}\bar{d}\bar{s}$	0	1115.683	-1	$+\frac{1}{2}$	$-$
Proton	p	uud	$+1$	938.272	$+1$	$+\frac{1}{2}$	$+$
Antiproton	$\bar{p}$	$\bar{u}\bar{u}\bar{d}$	-1	938.272	-1	$+\frac{1}{2}$	$-$
Positive pion	π^+	$u\bar{d}$	$+1$	139.570	0	0	$-$
Negative pion	π^-	$\bar{u}d$	-1	139.570	0	0	$-$

Table 1: Physical properties of the particles involved in the reaction. u, d, and s represent up, down, and strange quarks respectively. Bars denote antiquarks. Information extracted from [Z$^+$20].

This reaction is described succinctly by the following angular distribution $\mathcal{W}$ that governs the five angles $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$, and is parameterised by four physical parameters $\beta = (\eta, \Delta\Phi, \alpha_1, \alpha_2)$ [FK17]:

$$\begin{aligned}\mathcal{W}(\xi|\beta) =& \mathcal{F}_0(\xi) + \eta\mathcal{F}_5(\xi) \\ &+ \alpha_1\alpha_2\Big(\mathcal{F}_1(\xi) + \sqrt{1-\eta^2}\cos(\Delta\Phi)\mathcal{F}_2(\xi) + \eta\mathcal{F}_6(\xi)\Big) \\ &+ \sqrt{1-\eta^2}\sin(\Delta\Phi)\big(\alpha_1\mathcal{F}_3(\xi) + \alpha_2\mathcal{F}_4(\xi)\big),\end{aligned} \tag{2}$$

where the functions $\mathcal{F}_i(\xi)$, $i = 0, \cdots, 6$ are given by:

$$\begin{aligned}
\mathcal{F}_0(\xi) &= 1 \\
\mathcal{F}_1(\xi) &= \sin^2\theta \sin\theta_1 \sin\theta_2 \cos\phi_1 \cos\phi_2 + \cos^2\theta \cos\theta_1 \cos\theta_2 \\
\mathcal{F}_2(\xi) &= \sin\theta \cos\theta (\sin\theta_1 \cos\theta_2 \cos\phi_1 + \cos\theta_1 \sin\theta_2 \cos\phi_2) \\
\mathcal{F}_3(\xi) &= \sin\theta \cos\theta \sin\theta_1 \sin\phi_1 \\
\mathcal{F}_4(\xi) &= \sin\theta \cos\theta \sin\theta_2 \sin\phi_2 \\
\mathcal{F}_5(\xi) &= \cos^2\theta \\
\mathcal{F}_6(\xi) &= \cos\theta_1 \cos\theta_2 - \sin^2\theta \sin\theta_1 \sin\theta_2 \sin\phi_1 \sin\phi_2.
\end{aligned} \tag{3}$$

The angles and parameters that appear in this distribution are presented in Tables 2 and 3 respectively and are worthy of some explanation.

Angle	Description	Range
θ	Angle between incoming e^+e^- beam and outgoing $\Lambda\bar{\Lambda}$ beam	$[0, \pi]$
θ_1	Polar altitude of the velocity of p	$[0, \pi]$
ϕ_1	Polar azimuth of the velocity of p	$[0, 2\pi]$
θ_2	Polar altitude of the velocity of $\bar{p}$	$[0, \pi]$
ϕ_2	Polar azimuth of the velocity of $\bar{p}$	$[0, 2\pi]$

Table 2: Description of the angles characterising the reaction.

Parameter	Description	Range
η	Production parameter	$(-1, 1)$
$\Delta\Phi$	Phase parameter	$[0, 2\pi]$
α_1	Λ decay parameter	$[-1, 1]$
α_2	$\bar{\Lambda}$ decay parameter	$[-1, 1]$

Table 3: Description of the parameters that govern the reaction.

Firstly, the angles $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$ are what are measured when the reaction is physically carried out. The angle θ, as shown in Figure 3 and described above, is the angle between the incoming positron beam, and the trajectory of the outgoing Λ-hyperon. Furthermore, the pair of angles (θ_1, ϕ_1) are the polar angles describing the trajectory of the proton produced by the decay of the Λ-hyperon, relative to the direction of motion of the Λ-hyperon itself. This is shown diagramatically in Figure 4.

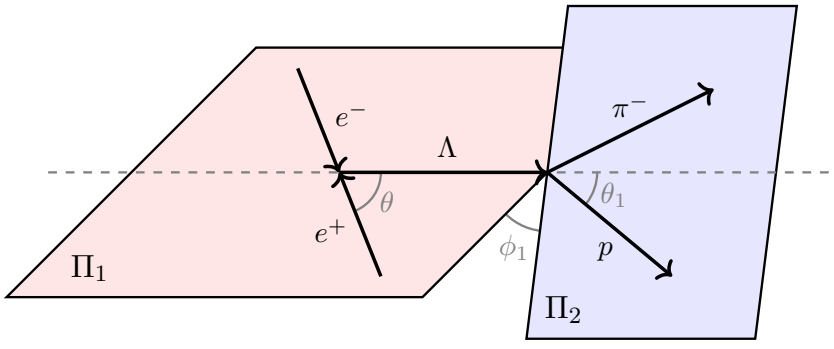

Figure 4: Graphical representation of production angle θ and decay angles (θ_1, ϕ_1).

As is shown in Figure 4, θ_1 is the polar angle between the directions of motion of the Λ-hyperon and the proton, whereas ϕ_1 is the azimuthal angle between the *production plane* Π_1, spanned by the momenta of the positron and the Λ-hyperon, and the *decay plane* Π_2, spanned by the momenta of the proton and the negative pion. The pair of polar angles (θ_2, ϕ_2) are defined analogously to (θ_1, ϕ_1) for the decay of the $\bar{\Lambda}$-antihyperon.

Secondly the parameters $\beta = (\eta, \Delta\Phi, \alpha_1, \alpha_2)$ as given in Table 3 are physical parameters that govern the overall shape of the distribution that models the measured angles ξ. The parameter η, known as the Λ *production parameter* governs distribution of the measured angle θ. The parameter η is derived from the quantity $R = \frac{|G_E|}{|G_M|}$, the ratio of the absolute values of the electric and magnetic form factors G_E and G_M, through the following equation:

$$\eta = \frac{\tau - R^2}{\tau + R^2}, \tag{4}$$

where τ is a physical parameter quantifying the energy of the incoming electron-positron beam. Broadly speaking the electric form factor G_E informs us of the distribution of charge within the hyperon. Overall Λ-hyperons have null charge, but the constituent up, down, and strange quarks have charges of $\frac{2}{3}$, $-\frac{1}{3}$, and $-\frac{1}{3}$ respectively. Therefore, within the hyperon, charge will be non-uniformly distributed. In particular, due to the greater mass of the strange quark, it is often located nearer the centre of the hyperon, thus making its charge more negative towards the centre. Secondly, the magnetic form factor G_M tells us about the magnetic properties of the hyperon produced by the motion of these same charged quarks. The *phase parameter* $\Delta\Phi$ concerns the phase between the electric and magnetic form factors and effects the distribution of the decay angles $(\theta_1, \phi_1, \theta_2, \phi_2)$. Finally, there are two *decay parameters* α_1 and α_2 that characterise the decays of the Λ-hyperon and $\bar{\Lambda}$-antihyperon respectively. More precisely, these decay parameters govern the emission directions of the proton and antiproton. Occasionally, the decays involve a *parity-violating amplitude* that give the produced (anti)proton a greater tendency to be emitted in the direction corresponding to the axis of spin of the Λ-(anti)hyperon. In other cases, with *parity-conserving amplitude*, there is no preference for the decay products to be emitted in any particular direction. The decay parameters α_1 and α_2 quantify the interference between the parity-violating and parity-conserving amplitudes. Thus, if CP-symmetry is strictly conserved in this reaction we should always find that $\alpha_1 = -\alpha_2$.

Finally, it is worth noting that in this reaction we are able to observe the conservation of both the overall charge and baryon number of the system. However, due to relativistic effects, we do not have conservation of mass, some of which is converted into the kinetic energy of the daughter particles.

3.2 The 3D formalism

In many cases, the full five-dimensional distribution as given in (2) may be too complex and unwieldy for theoretical or experimental analysis. In such cases it is therefore instructive to consider simplifications to the collision and decay events by focusing only on a particular subset of the particles and the angles between their trajectories. In the present and following subsections of this report we briefly explain two simplifications of the distribution in (2) that are useful in this regard. Throughout this discussion, the functions $\left\{\mathcal{F}_i(\xi)\right\}_{i=0}^{6}$ refer to those specified in (3).

Firstly, we consider the *'three-dimensional'* case shown in blue and red in Figure 5. In this simplification we consider both the collision of the electron-positron pair and the creation of the Λ-hyperon, as well as the subsequent decay of the hyperon, but ignore the creation and decay of the $\bar{\Lambda}$-antihyperon. Thus, the system depends on the parameters $(\eta, \Delta\Phi, \alpha_1)$ and we wish to measure the angles $(\theta, \theta_1, \phi_1)$. The distribution for this case is given as $\mathcal{W}(\xi) = \mathcal{F}_0(\xi) + \eta\mathcal{F}_5(\xi) + \alpha_1\sqrt{1-\eta^2}\sin(\Delta\Phi)\mathcal{F}_3(\xi)$. This distribution can be obtained by integrating (2) over θ_2 and ϕ_2, the angles governing the decay of the Λ-hyperon which we wish to ignore.

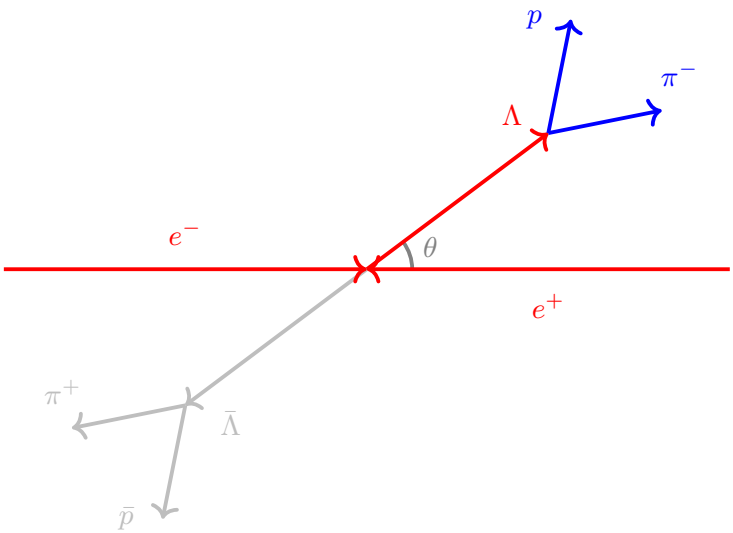

Figure 5: Graphical representation of the reaction showing which particles are considered by each of the five-, three-, and one-dimensional formalisms. Parts in red are considered in all formalisms, those in blue are considered by both the three- and five-dimensional formalisms, those in grey are only considered in the full five-dimensional formalism.

Finally, it is worth mentioning that it is convenient in the three-dimensional case to reduce the number of measured angles to two by defining a new angle θ_p that wholly describes the decay event. We define θ_p in terms of θ_1 and ϕ_1 through the relation $\cos\theta_p := \sin\theta_1 \sin\phi_1$, thus defining θ_p as the angle between the momentum of the proton, and the normal to the production plane $\vec{n}_{\Pi_1}$. If the three-dimensional case is considered in terms of θ and θ_p it is instead referred to as the two-dimensional case.

3.3 The 1D formalism

Finally, we consider the so-called *'one-dimensional'* case depicted in red Figure 5. This is the simplest case that we can construct as it depends on only a single parameter η and only considers the production angle θ. Despite not giving us any information regarding the decay of the Λ-hyperon or antihyperon, this case can be instrumentally useful by making it easier to measure the production parameter η that can then be assumed in the higher-dimensional cases. The resulting simplified distribution is $\mathcal{W}(\xi) = \mathcal{F}_0(\xi) + \eta\mathcal{F}_5(\xi) = 1 + \eta\cos^2(\theta)$. This distribution can be attained from the three-dimensional distribution given above by integrating over the θ_1 and ϕ_1 relating to the decay of the Λ-hyperon.

3.4 Computational tools

To conclude this section we now briefly discuss the use of computational tools and methods to study the reaction. While it is possible to run physical experiments of this reaction in particle colliders, such as the Beijing Spectrometer III (BESIII) collaboration of which the nuclear physics research group at Uppsala is a part, this is often incredibly incredibly expensive, time consuming, and logistically challenging. For this reason, as well as the ever-increasing ubiquity and availability of powerful computing technologies, it is currently both desirable and convenient to supplement these expensive physical experiments with computational methods. Furthermore, simulation tools often play a crucial role when designing, planning, and preparing for physical experiments, as well as when interpreting their results.

However, previous approaches to modelling this reaction have been complex and unwieldy, requiring significant background knowledge of both the underlying physics and experience with

low-level programming languages such as FORTRAN. Furthermore, these approaches model the entire physical process depicted in Figure 3, taking into account each particle and their properties in order to simulate the reaction. This involves using generating angles to simulate particles which would then produce signals in a simulated particle detector. These simulated detection signals would then exactly mimic the data produced by physical experiments. When using simulated data for the purpose of testing, validating, and refining methods of inferring physical parameters from the data (see Section 6), it may not be necessary to have simulated the reaction at this level of detail. Instead, using data generated directly from the mathematical distribution (2) is often sufficient for this purpose and can greatly reduce the time spent on data generation.

The remainder of this report focuses on this alternative, simpler approach that works directly with the mathematical distribution (2), rather than modelling the physical process itself. Additionally, in terms of practical considerations, we have exclusively used Python, a high-level programming language which is far more approachable and understandable than those used in previous computational tools for studying this reaction.

4 Monte-Carlo Methods

One effective way to simulate systems with a large number of degrees of freedom is to apply Monte-Carlo simulations [NB99]. The Monte-Carlo methods were invented to improve decision making under uncertainty and named after the famous casino town. These methods are also helpful for numerical integration and optimization problems. For example, it can be used to estimate parameters which correspond to a global maximum of a likelihood function.

In quantum physics, measurements and observations are considered as fundamentally random. Thus, we cannot describe deterministically one particle or a small system of particles. However, we can use probabilistic approach and predict the expected behaviour for en ensemble of particles as a group. This approach is analogue to the Monte-Carlo methods. The basic principle of the Monte Carlo methods is to use randomness. These methods allows to repeat the same computation experiments and obtain the sample description of the system.

In this work, we use Monte-Carlo to generate angular sample according to given distribution (2). Using these methods, we will simulate the chain reaction of electron/positron collision and following decays of hyperon/antihyperon. We implemented two Monte-Carlo methods: Hit-and-Miss and Metropolis method.

4.1 Hit-and-Miss (Rejection sampling)

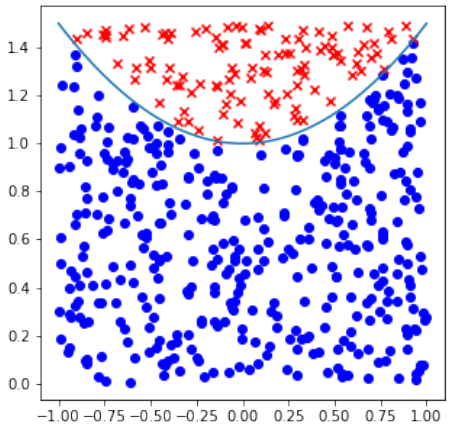

Figure 6: Example of generating 1D case (3.3) for $\eta = 0.5$. The blue dots are accepted samples, the red "x"-marks are rejected ones.

The Hit-and-Miss method, or acceptance-rejection method, is a basic approach used to generate samples. The basic idea is to generate one sample using a simple prior distribution (typically a uniform one) and after accept it or reject. To accept the new sample, the algorithm checks if the value of distribution (2) is below surface defining the function that describes the distribution we want to create. If it does, then the event is accepted. Otherwise, the event is rejected. Fig.6 illustrates the example for 1D case. The blue line is distribution function (3.3) with $\eta = 0.5$. Generated samples which are above the distribution curve are rejected. Here, the function is unique. More generated samples are necessary to have more precise approximation of the curve.

The algorithm is following:

0. Define the limiting upper value $h = \mathcal{W}_{max}(\xi)$
1. Generate new systems state ξ_{new}.
2. Generate $y \in [0, 1]$.
3. Accept the new state if $y < W(\xi_{new})/h$.

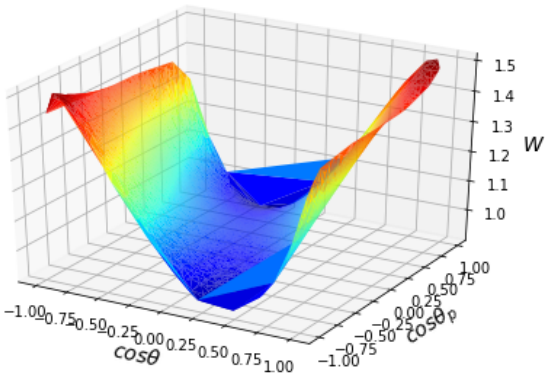

Figure 7: Example of 3D case (3.2). The axis z and the color is the value of distribution function.

The main advantage of this method is simplicity. However, this method often has lower efficiency and this method can be difficult to use for higher dimensions as small fraction of

new states survive. Figure 7 illustrates the distribution function for 3D case (3.2). The color on the plot is the value of the distribution function. The red regions correspond to the most concentrated points. The accepted value would be located under the surface of function. As we can see, there is a lot of space where points would be rejected.

This method can be improved to adaptive rejection sampling. In this case, defining upper curve $h(x) = \mathcal{W}_{max}(x)$ instead of the one limiting upper value $h = \mathcal{W}_{max}(x)$.

4.2 The Metropolis algorithm

4.2.1 Markov chain

A Markov chain is a special type of stochastic process which satisfies Markov condition, i.e. that the probability of each new generated sample depends only on the previously generated sample. The main property of Markov chain can be expressed as following:

$$P(\xi_i = x_i|\xi_1 = x_1, ..., \xi_{i-1} = x_{i-1}) = P(\xi_i = x_i|\xi_{i-1} = x_{i-1}) \tag{5}$$

In our case of angular distribution, we have continuous state space S. We denote the transition probability to generate the angular sample ξ_{i+1} from state ξ_i by $p(\xi_i, \xi_{i+1})$. We define the probability that system has the state ξ_i as $\pi(\xi_{i+1})$. We can define stationary state:

$$\pi(\xi_{i+1}) = \int_S \pi(\xi_{i+1})p(\xi_i, \xi_{i+1})d\xi_i \tag{6}$$

Simply speaking, a stationary distribution of a Markov chain is a probability distribution that remains the same in the Markov process as time goes.

4.2.2 Markov chain Monte-Carlo (MCMC, Metropolis)

Here, we discuss the Metropolis Monte Carlo method based on Markov chain. The main idea is to construct Markov chain which has the given probability distribution as equilibrium distribution. In Markov chain Monte Carlo methods (often referred to as MCMC methods), the next sample is dependent on the last accepted one.

For generating an appropriate random set of states according to the given probability distribution, the conditions of ergodicity and detailed balance should be placed on the Markov chain as it was proposed in [MRR$^+$53]. If conditions are satisfied, Markov chain achieves equilibrium and its stationary distribution (6) gets the given distribution. Ergodicity means that algorithm can generate any state ξ_{new} from any other state ξ_0 in a finite number of steps. In physics the detailed balance means that each elementary process is in equilibrium with its reverse process. Detailed balance condition can be written as follows:

$$p_{\xi_0}P(\xi_0 \to \xi_{new}) = p_{\xi_{new}}P(\xi_{new} \to \xi_0), \tag{7}$$

where $P(\xi_0 \to \xi_{new})$ is the transition probability from state ξ_0 to ξ_{new}. Here, $p_{\xi_0}, p_{\xi_{new}}$ are stationary distribution probabilities: $p_{\xi_0} = \mathcal{W}(\xi_0)$ (2). Therefore, the transition probabilities should satisfy the following condition:

$$\frac{P(\xi_0 \to \xi_{new})}{P(\xi_{new} \to \xi_0)} = \frac{p_{\xi_{new}}}{p_{\xi_0}} \tag{8}$$

In order to constraint transition probabilities for generating the distribution, we introduce the acceptance ratio $A(\xi_0 \to v)$ and write:

$$P(\xi_0 \to \xi_{new}) = g(\xi_0 \to \xi_{new})A(\xi_0 \to \xi_{new}),$$

where $g(\xi_0 \to \xi_{new})$ is the selection probability that algorithm will generate a state ξ_{new} from an initial state u and $A(\xi_0 \to \xi_{new})$ is the acceptance probability which defines if the algorithm

should accept a new state and change the system into a new configuration. The condition (7) can then be expressed as a fixed ratio:

$$\frac{P(\xi_0 \to \xi_{new})}{P(\xi_{new} \to \xi_0)} = \frac{g(\xi_0 \to \xi_{new})A(\xi_0 \to \xi_{new})}{g(\xi_{new} \to \xi_0)A(\xi_{new} \to \xi_0)} \tag{9}$$

We have a freedom to choose any selection probabilities $g(\xi_0 \to \xi_{new}), g(\xi_{new} \to \xi_0)$ and acceptance probabilities $A(\xi_0 \to \xi_{new}), A(\xi_{new} \to \xi_0)$ which satisfy (9). In our work, we use the simplest case when prior generating probabilities equal and come from uniform distributions: $g(\xi_0 \to \xi_{new}) = g(\xi_{new} \to \xi_0)$. Obviously, the ergodicity condition is satisfied as from any state we can generate any new one.

We denote one angular vector by ξ. One generated sample is one vector $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$. To implement this method, we follow a few steps:
1. Start from any initial angular vector ξ_0.
2. Generate new systems state ξ_{new}.
3. Calculate the acceptance ratio: $A = \mathcal{W}(\xi_{new})/\mathcal{W}(\xi_0)$.
4. Generate random number $y \sim U[0, 1]$. Accept the new state if $y < A$.

The main different with the Hit-and-Miss method is step 3. Here, the acceptation depends on the ratio between probability of new state and old one. So, the probability of the new state depends only on the current state.

In this method, the system should achieve equilibrium to have stationary distribution (6). Thus, we skip first iterations before collecting the samples as algorithm should to achieve equilibrium.

4.3 Project Goals

In this project, our final goal is to implement generator according to distribution (2). We focus on implementing 5D case (**??**) and 3D case (**??**) to study properties of generators. We want to explore if the methods described above are efficient when we move from 3D case to 5D. We can expect, that both methods are less efficient in 5D case in comparison to 3D case. Also, we can assume that hit-and-miss method would be less efficient in 5D case as we discussed above the example for moving to 3D case from 1D case.

The correctness of the implemented methods should be verified using Method of Moments or Maximum likelihood. We expect to have validated generator function. The functions take as arguments four parameters $\beta = (\eta, \Delta\Phi, \alpha_1, \alpha_2)$ (2). Implemented methods could be used to study charge-parity violation by varying α-parameters.

4.4 Strategy

We have organised the work as follows:

1. Implement Hit-and-Miss algorithm in 1D (3.3) as a prototype for the MC generator and the parameter estimation methods.

2. To model 3D case according to (3.2).

   - Implement Hit-and-Miss method and Metropolis.
   - Verify generators using Method of Moments and Likelihood method.
   - Compare performance of two methods measuring time of running and calculating the acceptance rate.

3. Focus on 5D case according to (2) which is our main goal.

   - Write code for both Monte-Carlo methods.

- Verify generators using Method of Moments.
- Check the convergence of Moments to theoretical values
- Compare performance of two methods measuring time of running and calculating the acceptance rate.
- Apply new parameters $\alpha 1$ and $\alpha 2$ to generators to study CP violation.

5 Implementation details

5.1 Hit-and-Miss Monte-Carlo

We now detail the specific implementation details for the Hit-and-Miss Monte-Carlo generator for the full five-dimensional distribution given in (2). The generator was written in Python and closely follows the theoretical Hit-and-Miss algorithm given in Section 4.1.

Firstly, a simple function representing the distribution (2) is given in Listing 1:

```
def W(beta, xi):
    eta, deltaphi, alpha1, alpha2 = beta    # Unpack individual parameters
    costheta, sintheta, costheta1, sintheta1, phi1, costheta2, sintheta2, phi2 =
        xi    # Unpack angles

    cosdeltaphi = np.cos(deltaphi)
    sindeltaphi = np.sin(deltaphi)

    cosphi1     = np.cos(phi1)
    sinphi1     = np.sin(phi1)

    cosphi2     = np.cos(phi2)
    sinphi2     = np.sin(phi2)

    # Calculate F_i's
    f0 = 1
    f1 = sintheta*sintheta*sintheta1*sintheta2*cosphi1*cosphi2 + costheta*
        costheta*costheta1*costheta2
    f2 = sintheta*costheta*(sintheta1*costheta2*cosphi1 + costheta1*sintheta2*
        cosphi2)
    f3 = sintheta*costheta*sintheta1*sinphi1
    f4 = sintheta*costheta*sintheta2*sinphi2
    f5 = costheta*costheta
    f6 = costheta1*costheta2 - sintheta*sintheta*sintheta1*sintheta2*sinphi1*
        sinphi2

    # Calculate value of W
    w = f0 + eta*f5 + alpha1*alpha2*(f1 + np.sqrt(1-eta*eta)*cosdeltaphi*f2 +
        eta*f6) + np.sqrt(1-eta*eta)*sindeltaphi*(alpha1*f3 + alpha2*f4)

    return w
```

Listing 1: Function representing the five-dimensional distribution $\mathcal{W}(\xi|\beta)$

The function `W(beta, xi)` simply takes as input two tuples representing the parameters $\beta = (\eta, \Delta\Phi, \alpha_1, \alpha_2)$ and angles $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$. However, since the angles are generated such that the values of $\cos\theta$, rather than the angles θ themselves, are drawn from a uniform distribution, $\cos\theta$ and $\sin\theta$ are passed as input to `W()` in place of θ. This increases the overall efficiency of the generator as it means that the value of θ is only computed using inverse trigonometric functions if the sample is accepted in the Hit-and-Miss algorithm[3]. The function then returns the value of the distribution for this set of input angles and parameters.

[3]This is also the case for both θ_1 and θ_2.

Secondly, the function `hit_miss_generator(N, params)` is the heart of the program, implementing the main functionality of the Hit-and-Miss algorithm. The Python code for this function is given in Listing 2.

```
def hit_miss_generator(N, params):
    eta, deltaphi, alpha1, alpha2 = params  # Unpack parameters
    hits = []       # Create empty return array

    sindeltaphi = np.sin(deltaphi)
    cosdeltaphi = np.cos(deltaphi)

    i = 0       # Initialise counter for number of accepted samples

    # Upper bound on distribution
    dist_max = 1 + abs(eta) + abs(alpha1)*abs(alpha2)*(1 + np.sqrt(1-eta*eta)*
        abs(cosdeltaphi)*0.5 + abs(eta)) + np.sqrt(1-eta*eta)*abs(sindeltaphi)*(
        abs(alpha1) + abs(alpha2))*0.5

    while len(hits) < N:
        i += 1

        # Draw angles from uniform distribution
        costheta = np.random.uniform(-1, 1)
        sintheta = np.sqrt(1 - costheta*costheta)   # sin(theta) >= 0 since 0 <=
            theta <= pi

        costheta1 = np.random.uniform(-1, 1)
        sintheta1 = np.sqrt(1 - costheta1*costheta1)
        phi1 = np.random.uniform(-np.pi, np.pi)

        costheta2 = np.random.uniform(-1, 1)
        sintheta2 = np.sqrt(1 - costheta2*costheta2)
        phi2 = np.random.uniform(-np.pi, np.pi)

        # Calculate W() for this set of angles
        dist = W((eta, alpha1, alpha2, deltaphi), (costheta, sintheta, costheta1
            , sintheta1, phi1, costheta2, sintheta2, phi2))

        # Draw uniform sample
        sample = np.random.uniform(0, 1)

        if sample < dist/dist_max:
            # Append to return array
            theta = np.arctan2(sintheta, costheta)
            theta1 = np.arctan2(sintheta1, costheta1)
            theta2 = np.arctan2(sintheta2, costheta2)
            hits.append((theta, theta1, phi1, theta2, phi2))

    print(f"{i} samples generated\t{N} accepted\tgiving an acceptance ratio of {
        N/i}")

    return hits
```

Listing 2: Function implementing the Hit-and-Miss Monte-Carlo algorithm

The function `hit_miss_generator()` takes as input N - the desired Monte-Carlo sample size, and β - the tuple of physical parameters for the distribution $\mathcal{W}$. A few elements of this code are worthy of further explanation. Firstly, the distribution's upper bound is computed and stored in the variable `dist_max`. This upper bound is approximated by independently computing the maximum values of each of the functions $\mathcal{F}_i$ and substituting these maxima into the distribution. This method therefore does not take into account the interdependency of the $\mathcal{F}_i$ and so is likely to be an overestimate of the true upper bound of $\mathcal{W}$. Secondly, note that the values of $\sin\theta$, $\sin\theta_1$, and $\sin\theta_2$ as computed from the corresponding uniformly-distributed cosines, are necessarily

positive. This is as desired due to the angles θ, θ_1, and θ_2 all being within the range $[0, \pi]$ as given in Table 2. Finally, if a sample is accepted, the angles θ, θ_1, and θ_2 are recovered from the values of their sines and cosines through use of the `numpy` function `arctan2`[4]. This function is based on the regular arctangent function with certain modifications to make it better suited for computing the polar coordinate θ for a point with Cartesian coordinates (x, y), which would usually be computed as $\theta = \arctan(\frac{y}{x})$.

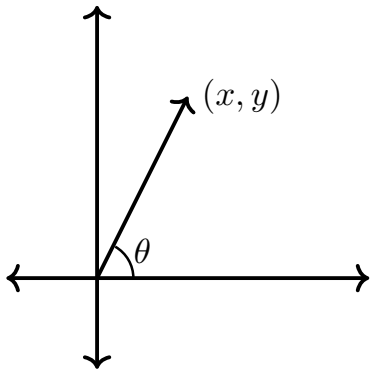

Figure 8: The value of θ can be computed as $\text{atan2}(y, x)$.

In particular, use of the arctan2 function ensures that the value returned is in the range $[-\pi, \pi]$ and that the edge cases where one or both of x or y is zero are defined. The full definition, as implemented in `numpy` is:

$$\text{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{for } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{for } x < 0,\ y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{for } x < 0,\ y < 0, \\ +\frac{\pi}{2} & \text{for } x = 0,\ y > 0, \\ -\frac{\pi}{2} & \text{for } x = 0,\ y < 0, \\ 0 & \text{for } x = y = 0. \end{cases} \tag{10}$$

Finally, the implementation has a `main()` function that sets the parameters based on user input and writes the generated sample angles to a `.txt` file. The main function is given in Listing 3:

```
def main():

    # Get inputs or use default parameter values
    if len(sys.argv) == 1:
        N = 100000
        eta = 0.461
        deltaphi = 0.74

    elif len(sys.argv) == 2:
        N = int(sys.argv[1])
        eta = 0.461
        deltaphi = 0.74

    elif len(sys.argv) == 4:
        N = int(sys.argv[1])
        eta = float(sys.argv[2])
        deltaphi = float(sys.argv[3])

    else:
```

[4] `https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html`

```
        raise ValueError("Incorrect inputs! Leave blank for default N, eta,
            deltaphi; define N only; or define all!")

    # Set default values for alpha1 & alpha2
    alpha1 = 0.754
    alpha2 = -0.754
    print(f"Parameters set to:\nN = {N}\neta = {eta}\ndeltaphi = {deltaphi}\
        nalpha1 = {alpha1}\nalpha2 = {alpha2}\n")

    np.random.seed()        # Seed random number generator

    print("Generating data...")
    data = hit_miss_generator(N, (eta, deltaphi, alpha1, alpha2))
    print("Data generated!\n")

    print("Writing to file...")
    write_file = open("gen_angles_5d.txt", "w")
    for sample in data:
        theta, theta1, phi1, theta2, phi2 = sample
        write_file.write(str(theta) + " " + str(theta1) + " " + str(phi1) + " "
            + str(theta2) + " " + str(phi2) + "\n")
    write_file.close()

    print("Done!")
```

Listing 3: Main function for Hit-and-Miss generator

5.2 Metropolis

The Metropolis method was implemented in Python as well. We use library Numpy [H+20] to calculate mathematical functions and generate random numbers for prior distribution from uniform one. The listings of the code correspond to 5D case (2).

First, we implement general functions to calculate the density distribution function, see Listing 4. The main function (2) *WDoubleTag* is decomposed due to style and practical reasons despite such implementation on Python could be more expensive in time. After, we use this function when implementing Metropolis generator, comparing visualisations and computing Moments empirically.

```
def F0(cos_th, cos_th1, cos_th2, phi1, phi2):
    return 1

def F1(cos_th, cos_th1, cos_th2, phi1, phi2):
    r = ((np.sin(np.arccos(cos_th)))**2 *np.sin(np.arccos(cos_th1))*np.sin(np.
        arccos(cos_th2)) *
            np.cos(phi1)*np.cos(phi2) + (cos_th)**2 *cos_th1*cos_th2)
    return r

def F2(cos_th, cos_th1, cos_th2, phi1, phi2):
    r = np.sin(np.arccos(cos_th))*cos_th*(np.sin(np.arccos(cos_th1))*cos_th2*
            np.cos(phi1) + cos_th1*np.sin(np.arccos(cos_th2))*np.cos(phi2))
    return r

def F3(cos_th, cos_th1, cos_th2, phi1, phi2):
    return np.sin(np.arccos(cos_th))*cos_th*np.sin(np.arccos(cos_th1))*np.sin(
        phi1)

def F4(cos_th, cos_th1, cos_th2, phi1, phi2):
    return np.sin(np.arccos(cos_th))*cos_th*np.sin(np.arccos(cos_th2)) * np.sin(
        phi2)

def F5(cos_th, cos_th1, cos_th2, phi1, phi2):
    return (cos_th)**2
```

```
def F6(cos_th, cos_th1, cos_th2, phi1, phi2):
    r = cos_th1*cos_th2 -(np.sin(np.arccos(cos_th)))**2*
            np.sin(np.arccos(cos_th1))*np.sin(np.arccos(cos_th2))*np.sin(phi1)*
                np.sin(phi2)
    return r

def WDoubleTag(cos_th,cos_th1,cos_th2,phi1,phi2,alpha,dPhi,alpha1,alpha2):
    xi = (cos_th,cos_th1,cos_th2,phi1,phi2)
    return F0(*xi)+alpha*F5(*xi) + alpha1*alpha2*(F1(*xi)+ ((1-alpha**2)**0.5)*
        np.cos(dPhi)*F2(*xi)+alpha*F6(*xi))
         + ((1-alpha**2)**0.5)*np.sin(dPhi)*(alpha1*F3(*xi)+alpha2*F4(*xi))
```

Listing 4: General functions for density distribution

Next, we write a function (Listing 5) which generates sample according to (2) via the Metropolis method. As the Markov chain needs to achieve equilibrium, we do not save samples from the first 10000 iterations of the algorithm (this refers to variable *burnin_size* in the code). We choose this number empirically by plotting the variances of mean values of angular vector components ξ versus different samples sizes. This number is not too large (in order to save time) but the sample variance of the sampled values becomes significantly lower.

```
def metropolis_hastings(target_density, params,size=5000):
    burnin_size = 10000 #skip this number of steps
    size += burnin_size
    x0 = np.array([[1,1,1,1,1]])/5 #just start point
    xt = x0
    samples = []
    alpha,dPhi,alpha1,alpha2 = params

    f = open("lAngles.txt", "w")
    xt_candidate = np.zeros((1,5))

    for i in range(size):
        #generate prior distribution
        xt_candidate[:,:3] = np.random.uniform(-1,1,size=(1, 3))
        xt_candidate[:,3:] = np.random.uniform(-np.pi, np.pi,size=(1, 2))
        th,th1,th2,ph1,ph2 = xt_candidate[0]
        #calculate density for new generated state
        q = target_density(th,th1,th2,ph1,ph2,alpha,dPhi,alpha1,alpha2)
        th,th1,th2,ph1,ph2 = xt[0]
        z = target_density(th,th1,th2,ph1,ph2,alpha,dPhi,alpha1,alpha2)
        #calculate acceptance ratio
        #the main difference with Hit-and-Miss
        accept_prob = (q)/(z)
        #decide either to accept new state or reject
        if np.random.uniform(0, 1) < accept_prob:
            xt = xt_candidate.copy()
        samples.append(xt)
        f.write(" ".join(str(item) for item in xt_candidate[0]) + "\n")

    samples = np.array(samples[burnin_size:])
    samples = np.reshape(samples, [samples.shape[0], 5])
    f.close()
    return samples
```

Listing 5: Metropolis generator

We use the implemented function *metropolis_hastings* in the *main* function to collect the samples. The function allows to use different parameters set as expected to study CP-violation.

6 Verification methods

Once a Monte-Carlo generator has been implemented, an important aspect to consider is how to verify its correctness. It is important to ensure that any generator creates samples that accurately represent the angular distribution function by ensuring that the angles generated with a given set of input parameters do indeed reflect the distribution with those parameters. This can be carried out through the use of parameter estimation methods. These methods can be thought of as functioning in the opposite direction to the Monte-Carlo generators discussed above; whereas generators take as input the physical distribution parameters $\beta = (\eta, \Delta\Phi, \alpha_1, \alpha_2)$ and generate independent samples $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$ based on these parameters, parameter estimation methods aim to infer the parameters that were used to generate a set of observed angles $\{\xi_i\}_{i=1}^N$.

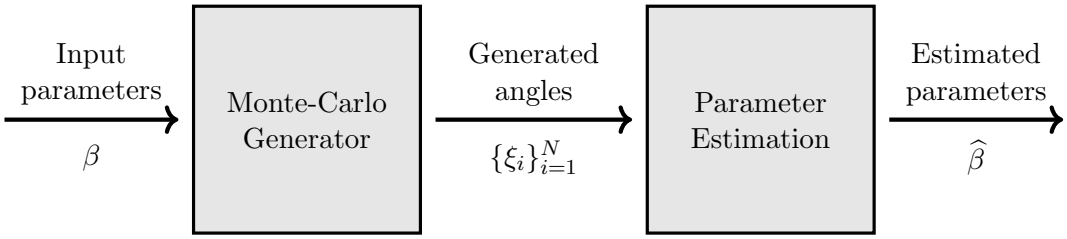

Figure 9: Verification through parameter estimation

In this section we detail two parameter estimation methods, the *method of moments*, and the *method of maximum log-likelihood*, that were used to verify the functionality of our Monte-Carlo generators.

6.1 Method of moments

First, we consider the *method of moments* [See e.g., Cow98, ch. 8], a simple parameter estimation method which can be applied to estimate the value of the ratio R, introduced in Section 3.1, from a set of observed angles $\{\theta_i\}$. This is a general method that gives the relationship between a parameter and a moment calculated from a set of data. In this case we relate the value of R and the second moment of $\cos\theta$, denoted $\langle\cos^2\theta\rangle$. In order to derive this relationship, we first take an alternative formulation of the one-dimensional distribution given in Section 3.3,

$$\frac{d\sigma(q^2)}{d\cos\theta} = \frac{2\pi\alpha^2\beta}{4q^2}[1 + \cos^2\theta + \frac{R^2}{\tau}\sin^2\theta], \tag{11}$$

where q is the *momentum transfer*, $\alpha \approx \frac{1}{137}$ is the *fine structure constant*, and $\beta = \sqrt{1 - \frac{1}{\tau}}$ [Tho22]. To show that this is indeed equivalent to the distribution in Section 3.3 we carry out the following algebraic manipulations,

$$\begin{aligned}
\frac{d\sigma}{d\cos\theta} &= \frac{2\pi\alpha^2\beta}{4q^2}[1 + \cos^2\theta + \frac{R^2}{\tau}\sin^2\theta] \\
&= \frac{2\pi\alpha^2\beta}{4q^2}[1 + \cos^2\theta + \frac{R^2}{\tau}(1 - \cos^2\theta)] \\
&= \frac{2\pi\alpha^2\beta}{4q^2}[1 + \frac{R^2}{\tau} + (1 - \frac{R^2}{\tau})\cos^2\theta] \\
&= \frac{2\pi\alpha^2\beta(\tau + R^2)}{4q^2\tau}[1 + \frac{\tau - R^2}{\tau + R^2}\cos^2\theta] \\
&= \frac{2\pi\alpha^2\beta(\tau + R^2)}{4q^2\tau}[1 + \eta\cos^2\theta],
\end{aligned}$$

where in the final step we have used the relationship $\eta = \frac{\tau - R^2}{\tau + R^2}$ given in (4). Thus, ignoring the constant term, we get the one-dimensional distribution as presented previously:

$$\mathcal{W}(\theta|\eta) = 1 + \eta \cos^2\theta.$$

Now, using the interpretation of the moment $\langle \cos^2\theta \rangle$ as the expected value of $\cos^2\theta$ in the range $[-1, 1]$, where $\langle X \rangle$ denotes the statistical moment of the variable X, we get the following expression involving the distribution (11):

$$\begin{aligned}
\langle \cos^2\theta \rangle &= \frac{\int_{-1}^{1} \cos^2\theta \frac{d\sigma}{d\cos\theta} d\cos\theta}{\int_{-1}^{1} \frac{d\sigma}{d\cos\theta} d\cos\theta} \\
&= \frac{\int_{-1}^{1} \frac{2\pi\alpha^2\beta}{4q^2} \cos^2\theta [1 + \cos^2\theta + \frac{R^2}{\tau}\sin^2\theta] d\cos\theta}{\int_{-1}^{1} \frac{2\pi\alpha^2\beta}{4q^2} [1 + \cos^2\theta + \frac{R^2}{\tau}\sin^2\theta] d\cos\theta} \\
&= \frac{R^2 + 4\tau}{5(R^2 + 2\tau)},
\end{aligned}$$

where the division by $\int_{-1}^{1} \frac{d\sigma}{d\cos\theta} d\cos\theta$ is required for normalisation of the distribution. From here, rearranging for R, we get the desired relationship between $\langle \cos^2\theta \rangle$ and R:

$$R = \sqrt{\tau \frac{10\langle \cos^2\theta \rangle - 4}{1 - 5\langle \cos^2\theta \rangle}}. \tag{12}$$

For a sample of N events, the uncertainty of the moment $\langle \cos^2\theta \rangle$ is given by

$$\sigma_{\langle \cos^2\theta \rangle} = \sqrt{\frac{1}{N-1}[\langle \cos^4\theta \rangle - \langle \cos^2\theta \rangle^2]},$$

which, using error propagation with the help of the chain rule, gives the uncertainty of R to be:

$$\begin{aligned}
\sigma_R &= \left| \frac{dR}{d\langle \cos^2\theta \rangle} \right| \sigma_{\langle \cos^2\theta \rangle} \\
&= \left| \frac{dR}{dR^2} \right| \left| \frac{dR^2}{d\langle \cos^2\theta \rangle} \right| \sigma_{\langle \cos^2\theta \rangle} \\
&= \frac{1}{2R} \frac{10\tau}{[1 - 5\langle \cos^2\theta \rangle]^2} \sigma_{\langle \cos^2\theta \rangle} \\
&= \frac{5\tau}{R[1 - 5\langle \cos^2\theta \rangle]^2} \sigma_{\langle \cos^2\theta \rangle}.
\end{aligned} \tag{13}$$

By constructing the estimator of the second moment as the arithmetic mean of $\cos^2\theta$ from the observed data,

$$\widehat{\langle \cos^2\theta \rangle} = \frac{1}{N} \sum_{i=1}^{N} \cos^2\theta_i, \tag{14}$$

where $\{\theta_i\}_{i=1}^{N}$ is the set of observed angles, an estimate for R can be computed by substituting the approximation $\widehat{\langle \cos^2\theta \rangle}$ into (12). This approximation can then be used to extract an estimate for the production parameter $\widehat{\eta}$ from (4), which can then be compared to the true value of η that was used to generate the data.

This method is of limited applicability due to its reliance on the analytic relationship (12) and thus is only available for computing the value R from the one-dimensional distribution of the $\{\theta_i\}_{i=1}^{N}$. For inferring other parameters from the higher-dimensional distributions we must

use other parameter estimation methods. One of these, the maximum log-likelihood method is discussed next.

Alternatively, we can estimate the parameters using the moment of absolute value for $\cos\theta$ instead of $\cos^2\theta$. To perform integration to obtain normalization factor and theoretical expected values, we use Wolfram Mathematica [Inc]. To begin, we calculate the normalization factor:

$$\mathcal{N} = \int \mathcal{W}(\xi) d\cos\theta d\Omega_1 d\Omega_2 = \frac{32}{3}(3+\eta)\pi^2. \tag{15}$$

First, we can use the first moment of absolute value $|cos\theta|$:

$$\langle|\cos\theta|\rangle = \frac{1}{\mathcal{N}}\int \cos\theta \mathcal{W}(\xi) d\cos\theta d\Omega_1 d\Omega_2 = \frac{3(2+\eta)}{4(3+\eta)}. \tag{16}$$

From this moment we can obtain the estimation for parameter η:

$$\hat{\eta} = \frac{6 - 12\langle\widehat{|\cos\theta|}\rangle}{4\langle\widehat{|\cos\theta|}\rangle - 3}, \tag{17}$$

$$\sigma_{\hat{\eta}} = \sqrt{\left(\frac{d\eta}{d\langle|\cos\theta|\rangle}\right)^2 \sigma^2_{\langle|\cos\theta|\rangle}} = \frac{\sqrt{12}\sigma_{\langle\widehat{|\cos\theta|}\rangle}}{4\langle\widehat{|\cos\theta|}\rangle - 3}. \tag{18}$$

Next, using the estimation $\hat{\eta}$ we can obtain the estimation for product of parameters $\alpha_1\alpha_2$. To this end, we use the expected value of product $|\phi_1||\phi_2|$:

$$\langle|\phi_1||\phi_2|\rangle = \frac{1}{\mathcal{N}}\int |\phi_1||\phi_2| \mathcal{W}(\xi) d\cos\theta d\Omega_1 d\Omega_2 = \frac{2\alpha_1\alpha_2 + 3\pi^2 + \eta * \pi^2}{4(3+\eta)}. \tag{19}$$

Finally, we estimate product $\alpha_1\alpha_2$:

$$\widehat{\alpha_1\alpha_2} = \frac{1}{2}(4\langle|\phi_1||\phi_2|\rangle(3+\hat{\eta}) - \hat{\eta}\pi^2 - 3\pi^2), \tag{20}$$

$$\sigma_{\widehat{\alpha_1\alpha_2}} = \sqrt{\left(\frac{d\alpha_1\alpha_2}{d\langle|\phi_1||\phi_2|\rangle}\right)^2 \sigma^2_{\langle|\phi_1||\phi_2|\rangle} + \left(\frac{d\alpha_1\alpha_2}{d\eta}\right)^2 \sigma^2_\eta}, \tag{21}$$

$$\sigma_{\widehat{\alpha_1\alpha_2}} == \sqrt{2(\eta+3)\sigma^2_{\langle|\phi_1||\phi_2|\rangle} + \frac{1}{2}(4 + \langle|\phi_1||\phi_2|\rangle + \pi^2)\sigma^2_\eta}. \tag{22}$$

We use this theoretical values to compare with the empirical values from samples obtained using our implemented generators.

6.2 Maximum log-likelihood

The second parameter estimation method that we discuss in this report is that of the method of maximum log-likelihood [See Ver21, for a more detailed exposition]. This is a slight variation on the well-established method of maximum likelihood estimation which, given a set of observed data and an assumed statistical distribution, constructs a *'likelihood'* function from the distribution's parameter space to the real numbers, where the function value at each point represents the likelihood that the observed data were generated by a process represented by the distribution with the corresponding set of parameters. The task is then to find the set of parameters that maximises this function either through analytical or numerical optimisation techniques.

More precisely, if $P(\xi|\beta)$ is a probability density function depending on parameters $\beta \in \mathbb{R}^\ell$, and $\xi = \{\xi_i\}_{i=1}^N$ is a set of data points drawn from the distribution $P(\cdot|\beta^*)$, then we construct the likelihood function as:

$$L(\beta|\xi) = \prod_{i=1}^{N} P(\xi_i|\beta) \tag{23}$$

for each point β in the parameter space of P. However, if the individual probabilities $P(\xi_i|\beta)$ are small, or N is very large, the likelihood, as a large product of small probabilities, may take impractically small values. One way of circumventing this practical difficulty is to instead construct the *log-likelihood* function by taking the logarithm of both sides of (23). Due to the properties of the logarithmic function, the product of probabilities can now be expressed as a sum of their logarithms:

$$\begin{aligned}\log L(\beta|\xi) &= \log\Big(\prod_{i=1}^{N} P(\xi_i|\beta)\Big) \\ &= \sum_{i=1}^{N} \log(P(\xi_i|\beta)).\end{aligned} \tag{24}$$

Since the logarithmic function is strictly increasing, this transformation preserves the relative likelihoods of each point in parameter space and so preserves the location of the function maximum in parameter space. Being able to reformulate the problem in terms of a sum rather than a product is also practically beneficial as it allows for easier computation of the derivatives of the likelihood function. Thus, the problem can neatly be formulated as a single maximisation problem with respect to β, namely,

$$\arg\max_{\beta} \sum_{i=1}^{N} \log(P(\xi_i|\beta)). \tag{25}$$

One further factor that must be taken into account when applying the method of maximum log-likelihood in our specific case, is that of normalisation. Recall that the likelihood function $L(\beta|\xi)$ is explicitly defined in terms of a *probability* distribution $P(\xi|\beta)$, i.e., $\int P(\xi|\beta)d\xi = 1$ for all β in the parameter space, where the integral is over the entire variable space. However, the distribution $\mathcal{W}$ of interest in the present report is not a probability distribution and so does not necessarily have this required property. Thus, in order to be able to apply the method of maximum log-likelihood we must create a normalised distribution $\widetilde{\mathcal{W}}$ with the same shape as $\mathcal{W}$ that does integrate to unity over its domain. To do this, we simply define a normalisation factor F_{norm} as the integral of $\mathcal{W}$,

$$F_{\text{norm}}(\beta) := \int \mathcal{W}(\xi|\beta)d\xi,$$

and construct $\widetilde{\mathcal{W}}$ by dividing $\mathcal{W}$ by this factor:

$$\widetilde{\mathcal{W}}(\xi|\beta) := \frac{1}{F_{\text{norm}}(\beta)}\mathcal{W}(\xi|\beta).$$

We are then able to use the normalised distribution $\widetilde{\mathcal{W}}$ in construction of the likelihood function as above. Note again that, since we take logarithms, the normalisation factor becomes an extra summation term:

$$\begin{aligned}\log(\widetilde{\mathcal{W}}(\xi|\beta)) &= \log\Big(\frac{\mathcal{W}(\xi|\beta)}{F_{\text{norm}}(\beta)}\Big) \\ &= \log(\mathcal{W}(\xi|\beta)) - \log(F_{\text{norm}}(\beta))\end{aligned}$$

It is worth mentioning that computing the integral required for normalisation in the general case is difficult as a result of it being high-dimensional and based on the complicated distribution given in (2). Furthermore, the integral must be computed for each choice of parameters β in an iterative optimisation scheme for solving (25). These factors motivate the use of numerical integration methods, the simplest of which is Monte-Carlo integration. In Monte-Carlo integration,

the integral of a function $f : \mathbb{R}^m \to \mathbb{R}$ on a subset $X \subseteq \mathbb{R}^m$ can be approximated by randomly sampling points $x_i \in X$ from a uniform distribution, taking the average function value at these points, and multiplying by the volume of X:

$$\int_X f(x)dx \approx V\Big(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\Big),$$

where the volume of X is given by the integral of the unit function $V = \int_X 1dx$.

Finally, due to the limited accuracy of Monte-Carlo integration, the data set on which it is carried out is often required to be significantly larger than that which is being analysed by the maximum log-likelihood method. This results in the majority of the computational time expended on solving the maximisation problem (25) is in fact spent computing the normalisation F_{norm} for each choice of β. Despite these practical challenges, it is maintained that the computational simplicity of Monte-Carlo integration makes it well suited to this use case.

7 Results

Here, we present our obtained results. To validate generators and to test performance we use parameters $[\eta, \Delta\Phi, \alpha_1, \alpha_2] = [0.461, \frac{\pi}{4}, 0.75, -0.75]$ as in [Col19].

7.1 Visual inspection

To visualize data we use a density heatmap. As we cannot easily visualize 5D data, we simply choose two axes and plot a density heatmap across them. Here, we perform qualitative tests by studying 2D projections of the angular distribution and check that they have the expected shapes.

First, we need to obtain a reference shape of the angular distribution. For this purpose, we can use the written functions in Listing 4. In short, we need to apply the angular vectors ξ over the entire variables space to implemented functions and observe the results. To visualise 5D, we generate uniform dataset in 5D space and for each point, we calculate the density according to (2). We prefer the randomly generated points in space rather than regular grid as generated ones are already uniform in space. This makes the procedure much faster. We refer to this computations as *a forward computation*. We can visualise forward results, using interpolation in Scipy library [V+20]. It allows to obtain the values of density function for empty space between generated points.

Figure 10 illustrates results for forward computation in the left column. In the right column, the figure presents visualisation of generated data. We see that they are qualitatively similar. The top left corner and bottom right corner have the largest concentration (red color) on both plots of $\cos\theta_1/\cos\theta_2$. On both plots of ϕ_1/ϕ_2, there are four concentrated centers. We perform this qualitative test for both Hit-and-Miss and Metropolis methods. Though this test is very coarse, it provides a first idea of the correctness of the generator which means we are now ready for quantitative tests.

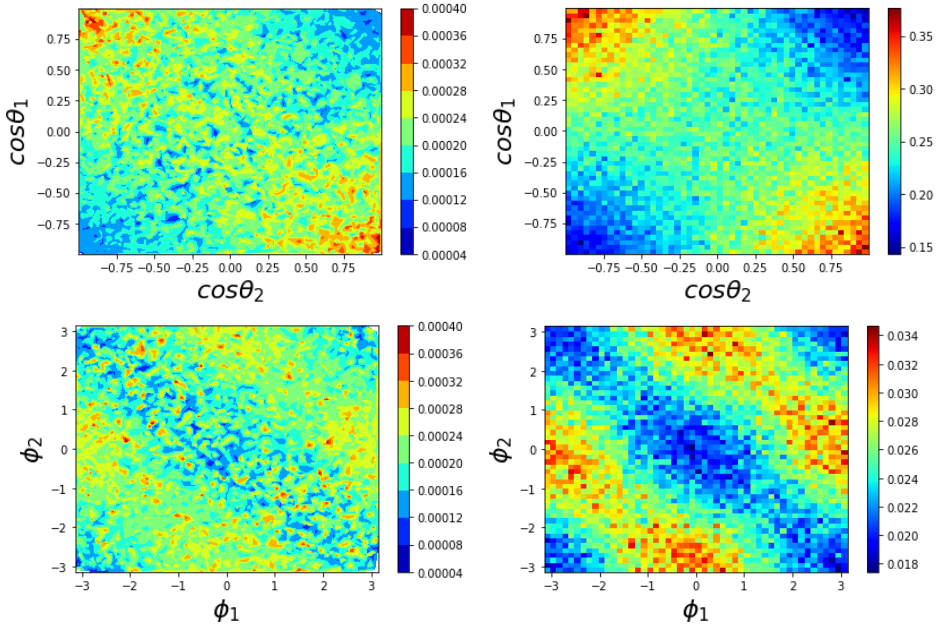

Figure 10: The left column is calculated forwardly density (2). The right column is generated data using Metropolis with $N_{steps} = 10^5$.

7.2 Standard error of mean values

The simplest test to check the implemented generators is to calculate the standard error of mean values of generated samples $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$. The error should decrease as the sample size N_{steps} grows.

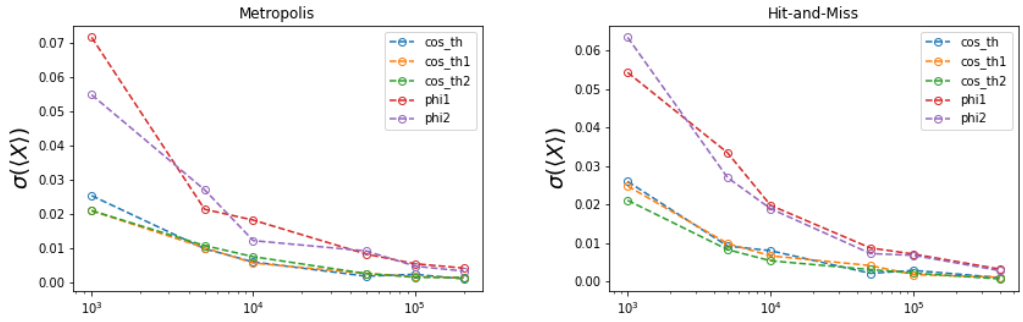

Figure 11: Standard error as a function of sample size N_{steps} for five variables $\xi = (\theta, \theta_1, \phi_1, \theta_2, \phi_2)$.

To performs standard error check, we run both programs varying N_{steps}. In each case, we calculate the sample standard error for components of the vector ξ. Figure 11 illustrates the obtained results. As we can see, the standard errors decline as expected. This plot is similar for both methods. Also, we can choose suitable value for number of the steps which should be skipped in the Metropolis algorithm. From the plot, we see that 10^4 is enough to achieve equilibrium in the system.

7.3 Parameter estimation

As discussed in Section 6, it is important to verify the functionality and accuracy of a Monte-Carlo generator to ensure that it produces data that reflects the underlying distribution. Here we

discuss the results of using parameter estimation methods on our Hit-and-Miss and Metropolis Monte-Carlo generators.

7.3.1 Maximum log-likelihood

In order to verify the correctness of our Monte-Carlo generators, we apply the method of maximum log-likelihood to our data in order to infer all four distribution parameters $(\eta, \Delta\Phi, \alpha_1, \alpha_2)$. However, results, not included here, show that this method is overly unstable and often fails to converge to suitable estimates. Due to this difficulty, attention is focused on parameter estimation using the simpler method of moments, the results of which are presented below.

7.3.2 Method of moments

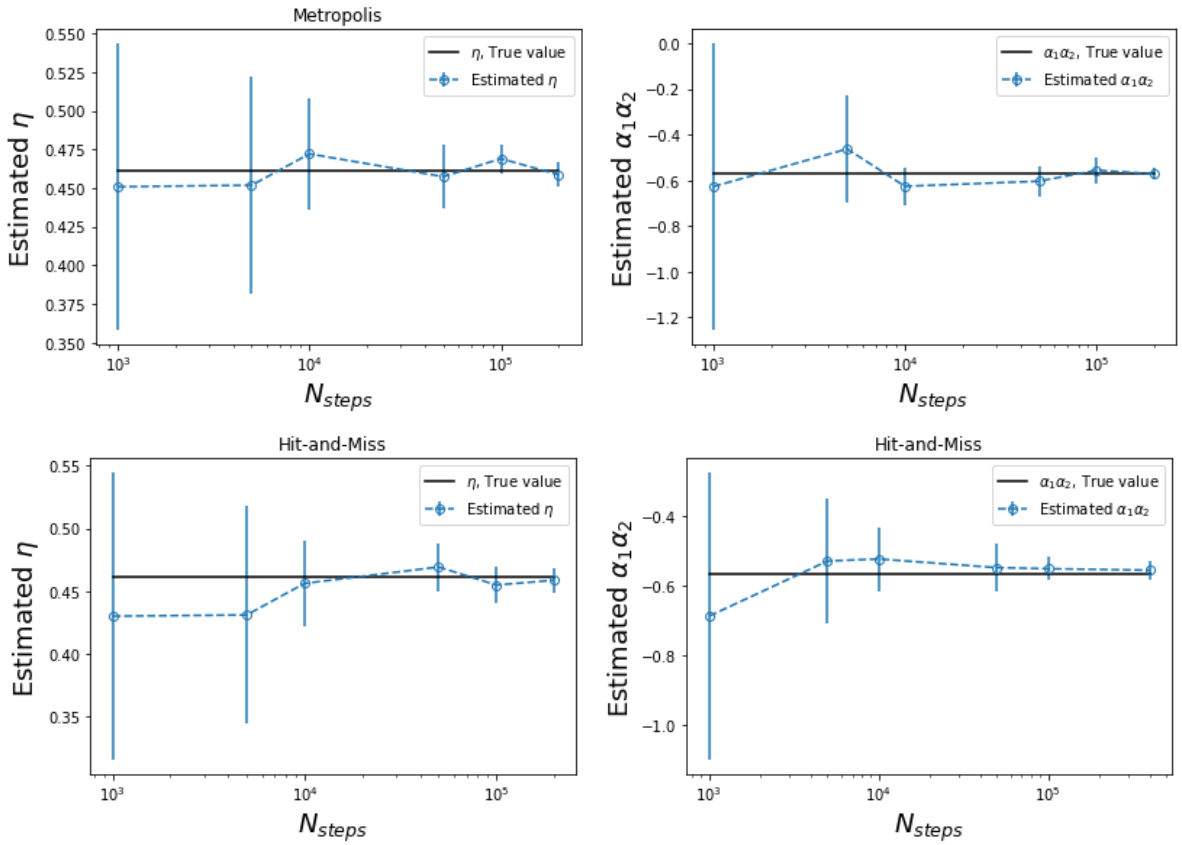

Figure 12: The estimation of parameters η (17) and $\alpha_1\alpha_2$ (20) for different sample sizes.

In the experiments, we follow the verification methods using parameter estimation as described in Figure 9. For each sample size N_{steps}, we make estimations for the parameter η (17) and the product $\alpha_1\alpha_2$ in (20) and measure uncertainty according to (18) and (22). Figure 12 presents the obtained results. As N_{steps} increases, the estimation of parameters get more precise. For both methods, it is enough to have 10^5 generated samples to obtain small errorbar.

7.4 Runtimes

One important aspect in this study is the runtime usage. To measure time, we use built-in magic command *time* [IPy21] in IPython. We choose to run 10 loop with 5 repeats for each sample size. Figure 13 shows the time results. We use Google Colab to run both methods on the same machine. The tests are performed using Intel(R) Xeon(R) CPU @ 2.20GHz. As we can see, time

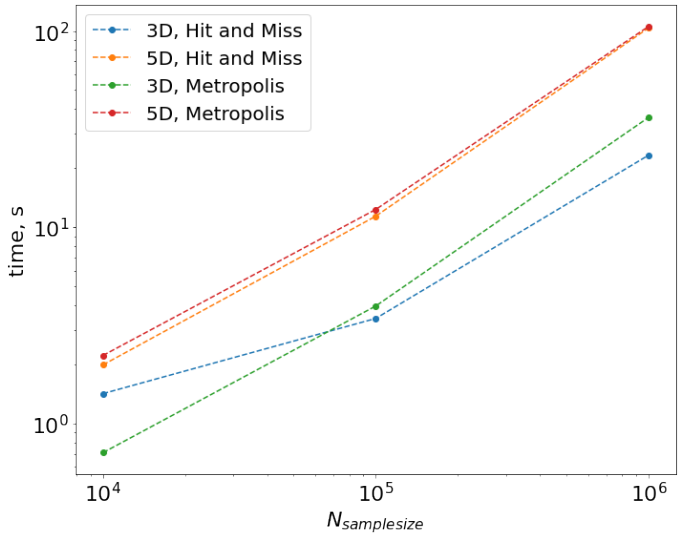

Figure 13: The best measured time using Intel(R) Xeon(R) CPU @ 2.20GHz.

consumption is similar for both methods. For low number of samples in 3D case, the Metropolis algorithm is quite slow. Probably, it is due to the time spent on achieving equilibrium. Also, time consumption could be difficult to compare as implementations are different. As we state in Implementation section, the implementation in Python language is quite slow in pack/unpack operations and calling functions.

7.5 Acceptance Rates & Efficiency

Next, to compare the performance of the methods, we measure r the acception ratio of the new generated states. By acception ratio ratio we mean the fraction of accepted samples to the total number of generated ones. We run 10 tests for each case and calculate the mean value of acception ratio.

When we move from the 3D to the 5D case, the acceptions ratio gets lower. However, the ratio for the Hit and Miss method gets much lower in comparison to that for Metropolis algorithm. This is expected when moving to higher dimensions. We show the example for moving from the 1D to the 3D case in Section 4.1. For higher dimensions, there are more empty area in the variables space.

Method	Case	Acceptance rate
Hit and Miss	3D	0.73 ± 0.01
Hit and Miss	5D	0.39 ± 0.02
Metropolis	3D	0.91 ± 0.03
Metropolis	5D	0.85 ± 0.01

Table 4: Acceptance ratio of new states. 10 runnings for each case.

7.6 Asymmetry parameters

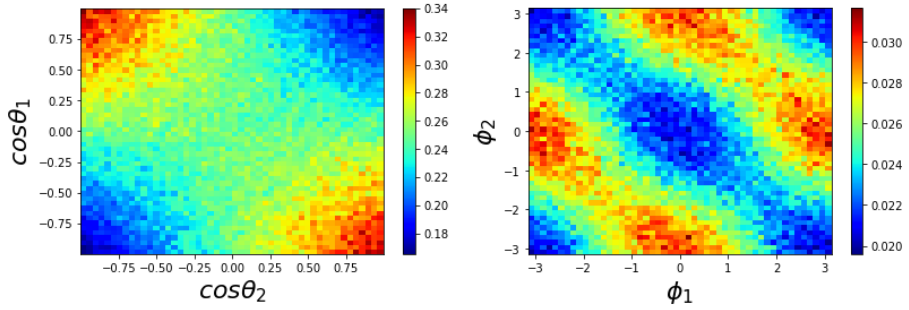

Figure 14: Generated data using parameters $[\eta, \Delta\Phi, \alpha_1, \alpha_2] = [0.461, \frac{\pi}{4}, 0.754, -0.654]$

We expect our programs to be useful when studying of the Charge-Parity violation in the chain collision-decay reaction. Now, we want to apply new parameters to generators and check the results.

We vary α parameters and apply $[\eta, \Delta\Phi, \alpha_1, \alpha_2] = [0.461, \frac{\pi}{4}, 0.754, -0.654]$. Figure 14 illustrates the new generated data. The visual difference in comparison to the previous set in Figure 10 is not obvious.

Next, we apply the procedure from Figure 9 to obtain estimations for $\hat{\eta}$ from (17) and the product $\widehat{\alpha_1\alpha_2}$. We obtain following estimations: $\hat{\eta} \approx 0.460$ and $\widehat{\alpha_1\alpha_2} \approx -0.493116$ (true value of $\alpha_1\alpha_2$ is approximately -0.502778). The estimated values are close to the applied ones. As we can see, our implementations allow to obtain samples for different parameters. We can continue study varying parameters.

8 Discussion

The main goal of our project is to implement generators for angular distribution (2) and verify their quality using different methods. We make validation of both methods Hit-and-Miss and Metropolis and test their performance for both 3D and 5D case. We apply the maximum log-likelihood method to estimate parameters, however, the method is unstable and extremely dependent on initial parameters.

As we propose before, the Hit-and-miss has lower acceptance ratio for 5D case in comparison to 3D. However, the Metropolis implementation does not outperform in time consumption. The most probable reason is the current version of the implementation. The metropolis program has extra callings of functions (due to decomposed way of writing) and extra packing/unpacking operations which is expensive in Python. This could be improved via changing the way the implementation of the density function in Listing 3 and removing extra numpy-array packing operations.

Now both our validated methods work and allow to generate samples up to 10^5 in two minutes.

9 Conclusion

This report explores the use of computational tools in the study of the $e^+e^- \to \Lambda\bar{\Lambda}$ reaction and subsequent $\Lambda \to p\pi^-$ and $\bar{\Lambda} \to \bar{p}\pi^+$ decays with the aim of finding instances of CP-violation, one of the three Sakharov conditions necessary for baryogenesis. If baryogenesis were shown to be a physical possibility, this could provide an explanation for how the universe could have come to be matter-dominated from an initial state with equal quantities of matter and antimatter. The report assesses the performance of two Monte-Carlo generators that produce simulated data

from the angular distribution that governs this reaction. One of these generators is based on Hit-and-Miss Monte-Carlo, while the other utilises the Metropolis algorithm. The report also details two parameter estimation methods, the method of moments and the method of maximum log-likelihood, that can be used to infer physical parameters from both simulated and experimental data of the reaction. The results of the assessment show that, while the Metropolis algorithm is considerably more efficient than Hit-and-Miss when it comes to accepting samples (see Table 4), it only performs marginally better in terms of run-times. Furthermore, there does not seem to be any significant difference between the two methods when it comes to their accuracy as judged by the method of moments estimates for the parameter η and parameter product $\alpha_1\alpha_2$ (see Figure 12).

There are multiple ways in which the work presented here can be built upon and extended. From a computational perspective, there are alternative Monte-Carlo algorithms, such as the *inverse transformation method*, that could potentially be applied to this reaction and subsequently compared to the two methods presented here. Secondly, further work could focus on scaling up the generators to more efficiently generate large data sets, potentially through parallel and distributed computing. Finally, from the physical side, more work could be carried out adapting the generators to other subatomic reactions that could also provide evidence for CP-violation.

References

[Ber02] Werner Bernreuther. CP violation and baryogenesis. *Lect.Notes Phys.*, 591:237–293, 2002. arXiv:hep-ph/0205279v1.

[Col19] The BESIII Collaboration. Polarization and entanglement in baryon–antibaryon pair production in electron–positron annihilation. *Nature Physics*, 15:630–634, 2019.

[Cow98] Glen Cowan. *Statistical Data Analysis*. Oxford University Press, 1998.

[FK17] Göran Fäldt and Andrzej Kupsc. Hadronic structure functions in the $e^+e^- \to \Lambda\bar{\Lambda}$ reaction. *Physics Letters B*, 772:16–20, 2017.

[H+20] Charles R. Harris et al. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[Inc] Wolfram Research, Inc. Mathematica, Version 13.0.0. Champaign, IL, 2021.

[IPy21] IPython. saw. `https://github.com/ipython/ipython/`, 2021.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[MU49] Nicholas Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[NB99] Mark Newman and Gerard Barkema. *Monte Carlo methods in statistical physics chapter 1-4*. Oxford University Press: New York, USA, 1999.

[RT99] Antonio Riotto and Mark Trodden. Recent progress in baryogenesis. *Annual Review of Nuclear and Particle Science*, 49(1):35–75, Dec 1999.

[Sak91] Andrei D. Sakharov. Violation of CP invariance, C asymmetry, and baryon asymmetry of the universe. *Phys. Usp.*, 34(5):392–393, 1991.

[Tho22] Viktor Thorén. *Hadron Physics in a Polarized World: Exploring Electromagnetic Interactions with Spin Observables*. PhD thesis, Uppsala University, 2022.

[V+20] SciPy 1.0 Contributors Virtanen, Pauli et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[Ver21] Benjamin Verbeek. Maximum likelihood estimation of hyperon parameters in Python: Facilitating novel studies of fundamental symmetries with modern software tools, 2021. Bachelor Thesis, Uppsala University.

[Z+20] P.A. Zyla et al. Review of Particle Physics. *PTEP*, 2020(8):083C01, 2020.