PHYC40600 Physics with Astronomy and Space Science Lab 2; Muon Detection and Half-life Estimation

Daragh Hollman
daragh.hollman@ucdconnect.ie
(Dated: October 4, 2023)

The aims of this investigation were to determine the half-life and mean lifetime of the muon and determine the cosmic muon flux as a function of zenith angle. This was achieved by measuring the lifetime of 36128 muon decays and comparing two binned averaging methods. The muon half-life was determined to be $\tau_{\frac{1}{2}} = (1.51 \pm 0.07)\,\mu$s, with a corresponding mean lifetime of $\tau = (2.2 \pm 0.1)\,\mu$s. These values are within uncertainty of accepted averages. No distinct relationship was found between the muon flux and the zenith angle. This is in contrast to measurements observed by others, and is likely due to the existence of large statistical errors from low total counts, or potentially an external unknown source of muons.

I. INTRODUCTION

The muon is a fundamental sub-atomic lepton part of the standard model of particle physics. The study of muons is important not only because it is fundamental to our understanding of particle physics, but it also has applications in other fields such as its use for radiography in archaeology and civil engineering [1]. For example, muon radiography has been used since the 1960s to measure thickness of material, such as searching for hidden chambers inside the Chephren pyramid [2] or in the characterisation of other large volume bodies such as volcanoes [3] and mineral deposits [4]. The understanding of muons and their properties opens up these fields to processes such as these to further knowledge and science across multiple disciplines.

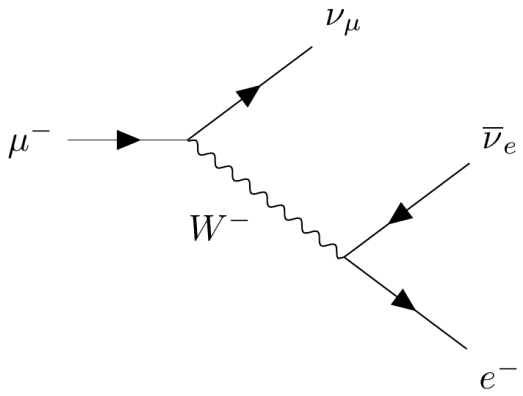

FIG. 1: A Feynman diagram for the muon decay as described in equation 1 [8]

II. THEORY

A. Muon Decay

Muons are elementary charged particles which are unstable [5]. The muon predominantly decays in the following way [6]:

$$\mu^- \rightarrow e^- \bar{\nu}_e \nu_\mu \tag{1}$$

This is a weak interaction with a Feynman diagram as shown in figure 1. It is also possible to have oppositely charged anti-muons which decays into the anti-particles of those in equation 1, these decay as follows:

$$\mu^+ \rightarrow e^+ \nu_e \bar{\nu_\mu} \tag{2}$$

Muon decay is a weak force and radioactive process and hence the decay of individual muons is random, but the decay for many muons exhibits an exponential decay curve - characterised by a decay constant λ - as described in equation 3.

$$N(t) = N_0 e^{-\lambda t} \tag{3}$$

where $N(t)$ is the number of radioactive particles at time t, and N_0 is the initial number of radioactive particles. The lifetime, τ, and half-life, $\tau_{\frac{1}{2}}$, can be calculated from the decay constant as follows [7]:

$$\tau = \frac{1}{\lambda}\,; \qquad \tau_{\frac{1}{2}} = \frac{\ln(2)}{\lambda} \tag{4}$$

B. Scintillation

As a muon, or any particle, passes through a scintillator its energy is absorbed by the scintillator, exciting atoms inside the material. When these atoms de-excite, they emit a photon which can be detected [9]. A scintillator can hence be used to detect muons which pass through it. If a muon loses all of its kinetic energy inside of the scintillator it will stop and subsequently decay which will create a second pulse of light as the electron moves through the scintillator. In this report, this process is referred to as a "double-detection".

C. Muon Flux

As cosmic rays impact the Earth's atmosphere, they undergo many decays resulting in a "shower" of different particles. Muons are formed from the decay of charged pions in this process [10]. These muons are created in the upper atmosphere and travel close to the speed of light towards the surface. The muon flux at sea level is approximately 1×10^{-2} muons cm^{-2} per minute [11]. A zenith angle of 0° (pointing straight upwards) is expected to provide the largest muon flux as it presents the least amount of atmosphere for the muon to travel through. We expect the flux to decrease as zenith angle increases.

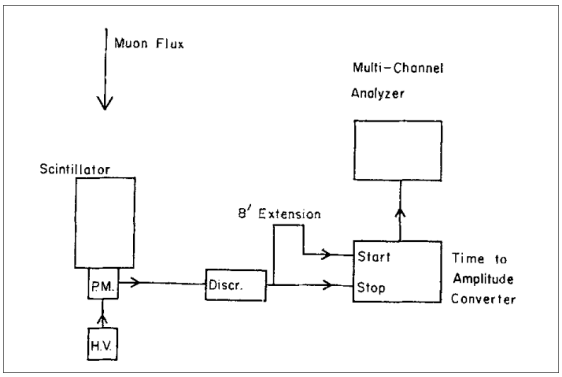

FIG. 2: The setup used in the experiment to detect muon decays. The acronyms used are as follows: P.M. - Photomultiplier tube, H.V. - High voltage input to the P.M., and Discr. - Discriminator.

D. Poisson Statistics

In this experiment due to the measuring of discrete random events, Poisson statistics are used. Poisson distributions are generally appropriate for counting events observed per unit interval, and hence they are important for dealing with random processes such as radioactive decay [12]. As the mean of a Poisson distribution increases beyond ≈ 10 events, the Poisson distribution closely approaches a Gaussian, or normal distribution, with variance equating to the square root of the mean [13].

In this investigation the "double detection" will therefore be measured, and the time difference between the start of the first pulse and the start of the second will yield the lifetime of the muon. If this is measured enough times, the random lifetimes will display an exponential distribution as described by equation 3. Neighbouring lifetime measurements will be binned and averaged using Poisson statistics to improve the accuracy of the measurements to yield a final exponential curve with which the half-life and mean lifetime will be determined.

III. METHODOLOGY

A. Apparatus

The apparatus was set up to replicate the diagram in figure 2. This is a simple setup, devised by R.E. Hall et al. to minimise the complexity of detection logic [11]. The muon flux causes scintillations which are measured and transformed into an electrical pulse by the photomultiplier tube (PMT). That signal is processed by a NIM model 623B discriminator [14] with a variable threshold. Signals arriving with amplitude above the threshold voltage are output as a fixed (but variable) width pulse. The output of the discriminator is split into the start and stop inputs of an ORTEC model 566 Time to Amplitude Converter (TAC) [15]. To ensure correct TAC pulses, the two cables used to connect the output of the discriminator to the TAC are of different lengths (approx. 60 cm difference). This was to create a short delay in the signalling to ensure the TAC is always in the correct measuring state. In its default state, the TAC only accepts start pulses and hence ignores any stop pulses. The built-in delay from the differing length cables ensures that any on-going measurements are stopped such that the TAC can accept the start pulse.

It is worth noting that this extension in the start cable - and hence delay in the start signal - does decrease the measured time between subsequent start and stop pulses, and therefore the TAC will produce reduced amplitude outputs. However, as this delay is on the order of nanoseconds and our measurements are on the order of microseconds, the impact of this should be negligible [11].

The TAC will output a pulse with amplitude corresponding to the time difference between its start and stop inputs. This output is read by a Muli-Channel Analyser (MCA) which records a count in a channel corresponding to the amplitude.

1. Muon Source and Detection

The source of muons used in this investigation was the natural cosmic ray flux which was incident at approximately 1×10^{-2} muons cm^{-2} per minute over all directions. They pass through a scintillator which creates a pulse of light which is converted to an electrical pulse by a PMT at 1800 V. Most muons will pass through the scintillator, however, if a muon loses all of its kinetic energy in the scintillator (i.e. stopping) it will decay with its products creating a second pulse [11]. This "double detection" was used to find the time between the muon entering the scintillator, and when it decays. This time will be random, but statistically will follow the exponential decay in equation 3. In reality we must adjust this equation to account for muons arriving at close enough times that they would be counted as a double detection. This will create a background in our data which follows

a Poisson distribution, accounted for in equation 7.

B. Analytical Methods

The MCA was set up to record TAC amplitudes across 4096 channels. To perform statistical analysis on these data, it was required to bin the data into larger groups. Inside each bin, the counts could be statistically analysed to find a mean and variance for each bin. This was done using two methods: a Gaussian approximation of a Poisson distribution, and the method of maximum likelihood estimation.

1. Gaussian Approximation of Poisson Statistics

The data in each bin will follow a Poisson distribution due the nature of counting discrete random events [12]. The Poisson distribution approaches a Gaussian distribution as the mean gets larger. With a mean larger than 10, the Poisson distribution has sufficiently low skew to be approximated by a Gaussian distribution [13]. Hence the mean (μ) was calculated and the variance given as follows:

$$\sigma = \sqrt{\mu} \tag{5}$$

and therefore a standard error of:

$$\mathrm{SE} = \frac{\sigma}{\sqrt{N}} \tag{6}$$

where N is the number of samples in the bin.

2. Maximum Likelihood Estimation (MLE)

Another method can also be used to find an average value and uncertainties for the data in each bin. The probability mass function of the Poisson distribution was used to determine the probability that the data corresponds to a Poisson distribution with a given mean. By varying this mean, the sum of these values can be considered the likelihood of mean, and we can find the mean which produces the largest likelihood. A minimising function such as SCIPY's *optimize.minimize* [16] was used to do this process by instead of maximising the likelihood, minimising the negative likelihood. A Nelder-Mead algorithm was used to find mean which produced the minimum negative likelihood. The 68% confidence interval of the Poisson distribution was then automatically calculated to estimate the standard deviation of the bin.

3. Choosing a bin size

As mentioned previously the data had to be binned to improve the counting statistics of the measurements. This leaves open the question of what is an appropriate bin size. This was investigated by creating a set of fake data matching an arbitrary exponential decay function and then looking at the effect of binning on the data points. Figure 3 shows the visual effect of binning the data for such a function for bin sizes between 20% and 5%. It can be seen that with large ($\approx>$ 10%) bin sizes the binned data can significantly drift from the raw data for such a function. To attempt to choose the most ideal bin size, least squares fitting was used to fit an exponential decay to the binned data for a variety of bin sizes. This was done automatically for 1000 bin sizes between 0.01% and 0.1% and the uncertainties for each parameter of the exponential were calculated from the covariance matrix. These uncertainties were plotted as a function of bin size in figure 3.

The output of this plot is somewhat unexpected. Ideally the uncertainty would initially decrease with bin size until approaching some plateau where the uncertainty no longer decreases significantly. This point would be chosen then as any larger bin size would result in a loss of data points without increasing uncertainty. Perhaps adding noise to the fake data would've created this effect but due to time constraints this could not be investigated further. Instead 0.045% was chosen arbitrarily (as indicated by the vertical line) as the bin size to use.

4. Fitting

The data in each bin was represented by a point with value of the mean and uncertainty taken from the bin's averaging method (both the Gaussian approximation, and the MLE method were used). A curve could then be fit to these points using least squares fitting. From the theory section, it is clear that our data should follow an exponential decay as in equation 3 and hence a similar form was chosen to be the fitting function.

$$f(t) = Ae^{-Bt} + C \tag{7}$$

The background parameter, C, was included to allow the algorithm to adjust for any unexpected background, which is further discussed in the analysis.

5. Calibration

As the MCA solely records counts in terms of a channel number. These channels correspond to the time differences from the TAC - i.e. the decay time of the muon - and hence they must be calibrated to relate the channel number to decay time. This was done using a pulse generator to create a double pulse with a variable delay between the pulses. The output of the pulse generator was split in parallel. Split between the oscilloscope - so that the pulse delay could be more

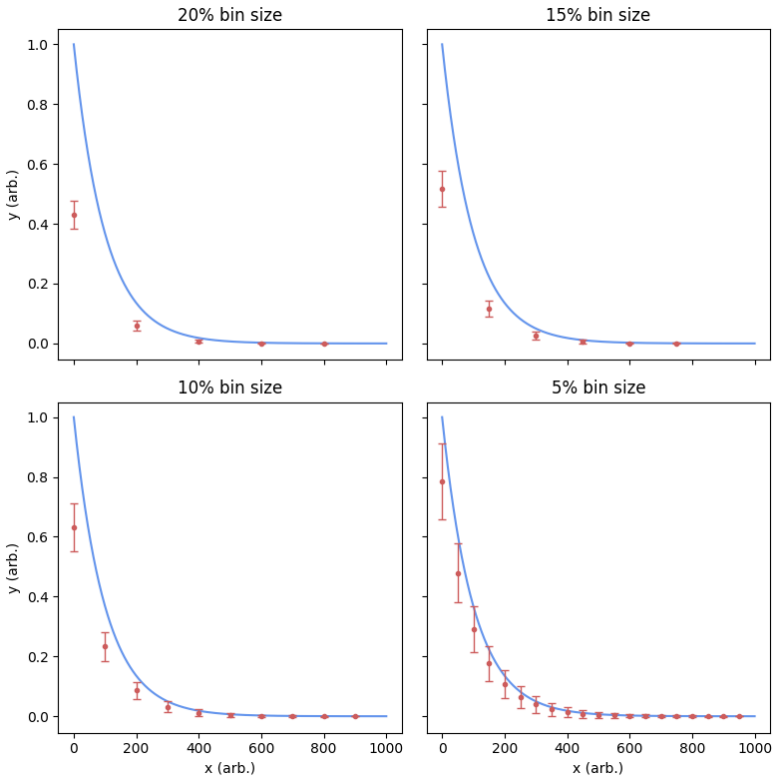

FIG. 3: A demonstration of how increasing the bin size decreases the accuracy of the least squares fit, but also decreases the standard error on the binned data points. Binned data points are displayed in red with the original data in blue. It is clear that with large bin sizes the binned data can significantly drift from the raw data.

accurately measured - and the discriminator, the output of which was split in parallel between the start and stop inputs of the TAC.

Measurements were taken from $(0.1 \pm 0.01)\,\mu$s to $(1 \pm 0.01)\,\mu$s corresponding to MCA channels from 73 ± 4 t0 807 ± 4. Least squares fitting was again - but with a linear function - used to map a linear calibration function to these data.

C. Muon Flux as a Function of Zenith Angle

To measure the Muon Flux as a function of the zenith angle, a different apparatus was used. This apparatus consisted of two scintillators and PMTs separated by 2 meters vertically and affixed to a frame free to rotate. The scintillator will create a pulse of light which is detected by the PMT for any angle of incidence, and hence only if both detectors receive a pulse at the same time (as the time of flight between the detectors is negligible) we can say that the muon has come from a direction in-line with both of the detectors. We will refer to this as a coincidence count. By measuring the number of coincidence counts in a given time across different angles, we can characterise the muon flux as a function of the zenith angle, which can be converted to muon flux as a function of distance travelled through the atmosphere.

The output from each PMT was again passed through an amplifier to increase the signal strength. A PMT voltage of 1800 V was used. The outputs of the amplifier were again also passed through a discriminator. The pulses were measured directly from the PMTs using an oscilloscope. Noticing that the pulses had lower amplitude than that from the single detector setup, the discriminator threshold was lowered until a reasonable count-rate was found at 250 mV. The output of each discriminator module was then passed into an ORTEC Model CO4020 Quad 4-Input Logic Unit [17] to detect these coincidences and then passed to a Thandar TF1100 Universal Counter. The output pulse width from the logic unit was too thin to be detected by the counter so a gate generator was inserted to increase the length of the pulse to 1 ms. A LeCroy Model 222 Dual Gate Generator was used [18]. The zenith angle, total counts, and counting time were recorded for 8 angles between 0° and 90°.

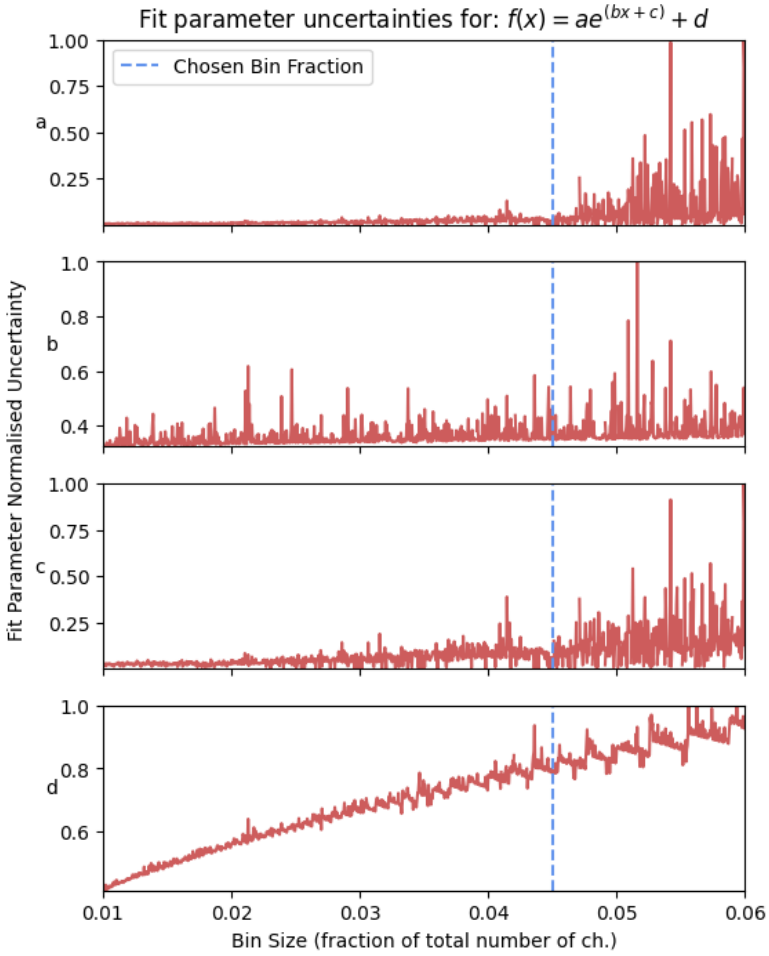

FIG. 4: The effect on the uncertainty of fitting parameters for the example data set used in figure 3 as a function of bin size. The blue vertical line represents the bin size chosen for the analysis.

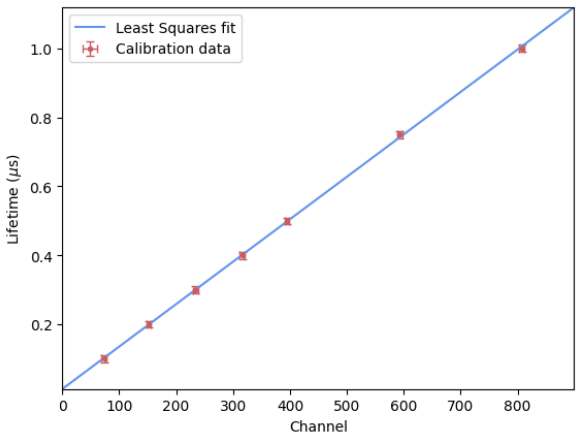

FIG. 5: The calibration curve created by passing the pulse generator output directly into the MCA. It found a linear relationship with a slope of 0.00123 ± 0.000007 and y intercept of 0.012 ± 0.003.

IV. RESULTS AND ANALYSIS

A. Muon Half-life Determination

A long data run was taken using a discriminator threshold of (800 ± 5)mV and a discriminator pulse width of (40 ± 15)ns. The MCA software was configured in a way as to take 100 measurements, each ≈ 1.2 hours long. This resulted in $>$ 5days long data collection, with 36128 double detection occurrences as shown in figure 6. These data were binned as shown in figure 7 and subsequently calibrated.

1. *Gaussian Approximation*

The binned data from the Gaussian approximation method were plotted in figure 8 and an exponential decay curve was fit. The half-life and mean lifetime of the muons were calculated from the decay constant as in equation 4 and determined to be $\tau_{\frac{1}{2}} = (1.51 \pm 0.07)\,\mu$s

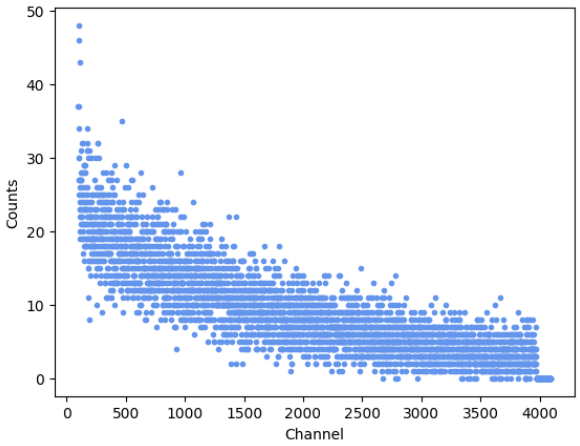

FIG. 6: Full 5 day data set with 36128 occurrences.

and $\tau = (2.2 \pm 0.1)\,\mu$s

2. *Maximum Likelihood Estimation*

This was also repeated for the MLE method as plotted in figure 9. The half-life and mean lifetime of the muons were similarly calculated from the decay constant and determined to be $\tau_{\frac{1}{2}} = (1.6 \pm 0.3)\,\mu$s and $\tau = (2.3 \pm 0.4)\,\mu$s

3. *Discussion*

The mean lifetime of the muon as measured by the Particle Data Group is $(2.1969811 \pm 0.0000022)$ [6], and hence a half-life of $\approx 1.523\ldots$. The value for lifetime is within the uncertainty of our measurements for both bin averaging methods. The value for half-life is within the uncertainty of that calculated using the MLE method but this measurement is less precise than that found with the Gaussian method. However, the value for half-life is slightly higher than that calculated with the Gaussian approximation method. This discrepancy is likely due to the muon flux being comprised of both μ^- and μ^+ particles. The μ^+ particles have a slightly increased half-life compared to that of the of the μ^- as the μ^- can interact with a proton via the weak force forming a neutron whereas a μ^+ cannot. This process is known as muon capture [19]. More importantly however, the muons have opportunity to undergo capture in their entire journey through the atmosphere resulting in a reduced flux of μ^-. Sirunyan et al. present the ratio of positive to negative muon fluxes to be 1.2766 ± 0.0032 meaning 56% of the flux are μ^+ and 44% μ^- [20].

Note that we see non-negligible values for the background parameter in equation 7 of ≈ 0.5 counts justifying its existence in the fitting function. These background counts are most likely due to coincidental double

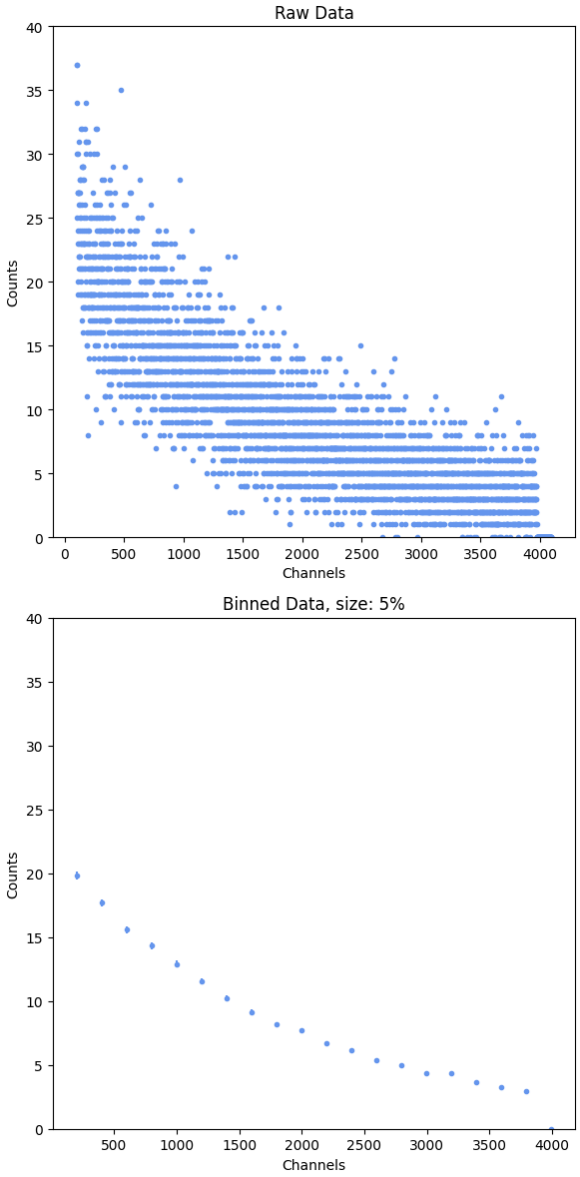

FIG. 7: Example of how the larger data set is binned. Here, the bin size is sufficiently small such that the error bars on the binned data are not visible.

pulses from the scintillator as two muons arrive at close enough times. These events would also follow a Poisson distribution and given the total counts from the discriminator this could be characterised and compared to the background parameter. However, due to time constraints this could not be included in this investigation.

B. Muon Flux

For 10 angles between 0 and 60 degrees the coincidence counts were measured. For each angle the counts were measured for approximately 20 minutes. These data are recorded in table I and are plotted in figure 10.

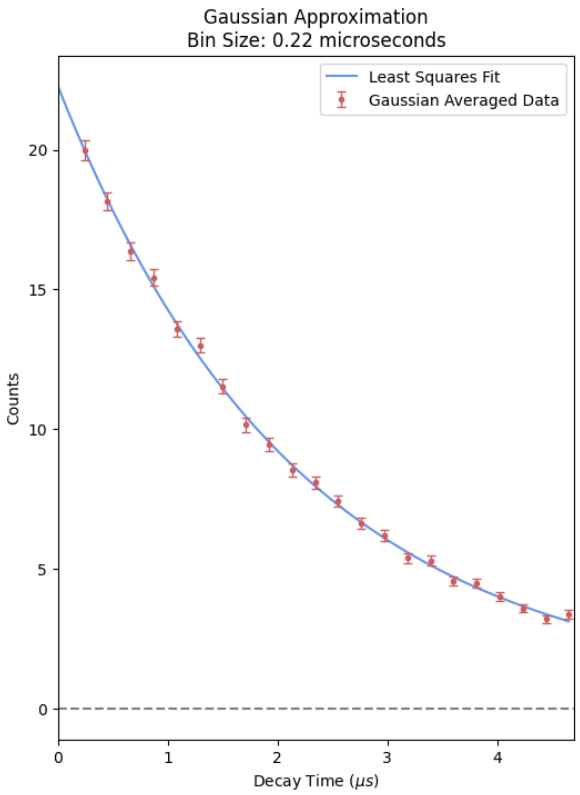

FIG. 8: Binned data plotted using the Gaussian approximation of a Poisson distribution (red). An exponential decay curve is plotted using least squares fitting (blue).

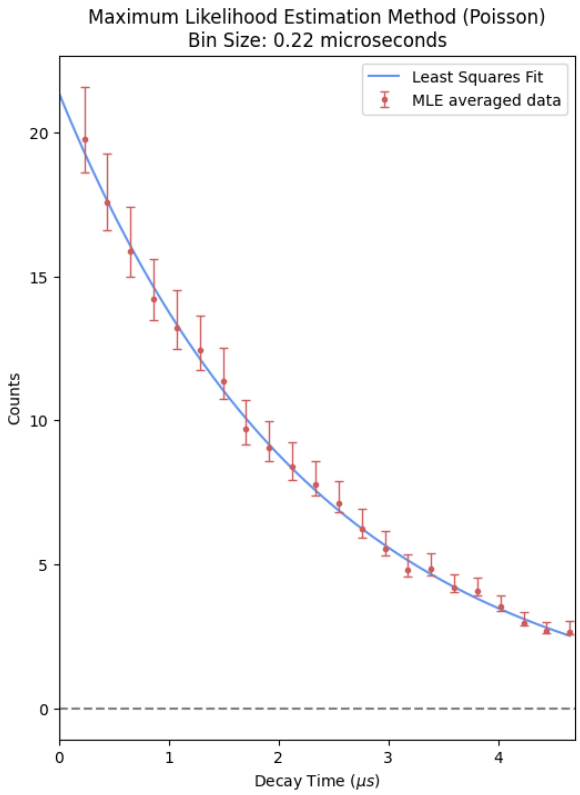

FIG. 9: Binned data plotted using the maximum likelihood estimation method (red). An exponential decay curve is plotted using least squares fitting (blue).

The uncertainty on the angle was due to the coarse graduation of the apparatus and the uncertainty on the count rate was propagated from the uncertainty on the time measurements and counts - which were determined to be the square root of the value assuming the counts follow a Poisson distribution.

The data in figure 10 follows no discernible trend. This is unexpected as J.L. Autran et al. observe a distribution following a cosine power law [21]. It is possible that our results lake sufficiently large count numbers to accurately measure this distribution or that there was some external source of muons at a particular range of angles which was unaccounted for.

V. CONCLUSION

The aims of this investigation were to determine the half-life and mean lifetime of the muon and determine the cosmic muon flux as a function of zenith angle. The muon half-life was determined to be $\tau_{\frac{1}{2}} = (1.51 \pm 0.07)\,\mu$s, with a corresponding mean lifetime of $\tau = (2.2 \pm 0.1)\,\mu$s. These values are within uncertainty of averages made by the Particle Data Group at CERN [6]. No distinct relationship was found between the muon flux and the zenith angle. This is in contrast to measurements observed by others, [21][22], and is likely due to the existence of large statistical errors from low total counts, or potentially an external unknown source of muons.

Angle [°] ($\pm 2°$)	Counts	Time [s] ($\pm 0.5s$)
0	31	1860
10	53	1613
20	66	1501
30	49	2048
35	40	825
40	59	1632
45	31	694
50	16	1709
55	20	912
60	8	1201

TABLE I: Table of values recorded while measuring the muon flux as a function of angle.

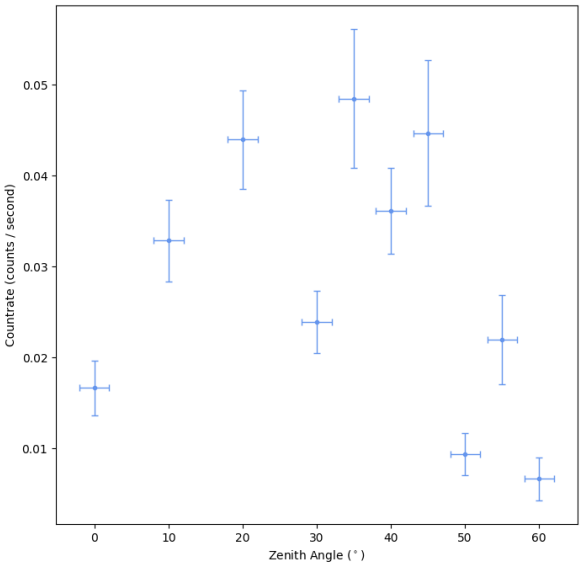

FIG. 10: Data from table I plotted to display the relationship between the cosmic muon flux and the zenith angle. No discernable relationship could be made from these data.

[1] G. Saracino, F. Ambrosino, L. Bonechi, L. Cimmino, R. D'Alessandro, M. D'Errico, P. Noli, L. Scognamiglio, and P. Strolin. Applications of muon absorption radiography to the fields of archaeology and civil engineering. *Philosophical transactions of the Royal Society of London. Series A: Mathematical, physical, and engineering sciences*, 377(2137):20180057, 2018.

[2] Luis W. Alvarez, Jared A. Anderson, F. El Bedwei, James Burkhard, Ahmed Fakhry, Adib Girgis, Amr Goneid, Fikhry Hassan, Dennis Iverson, Gerald Lynch, Zenab Miligy, Ali H. Moussa, Mohammed-Sharkawi, and Lauren Yazolino. Search for hidden chambers in the pyramids. *Science (American Association for the Advancement of Science)*, 167(3919):832–839, 1970.

[3] R Bajou, M Rosas-Carbajal, A Tonazzo, and J Marteau. High-resolution structural imaging of volcanoes using improved muon tracking. *Geophysical Journal International*, 235(2):1138–1149, 07 2023.

[4] Zong-Xian Zhang, Timo Enqvist, Marko Holma, and Pasi Kuusiniemi. Muography and its potential applications to mining and rock engineering. *Rock mechanics and rock engineering*, 53(11):4893–4907, 2020.

[5] Kanetada Nagamine. *What are muons? What is muon science?*, page 1–16. Cambridge University Press, 2003.

[6] R. L. Workman and Others. Review of Particle Physics. *PTEP*, 2022:083C01, 2022.

[7] David Jenkins. Nuclear structure and radioactive decay. In *Radiation Detection for Nuclear Physics*, 2053-2563, pages 1–1 to 1–34. IOP Publishing, 2020.

[8] A. Dobbs, Imperial College London, Introducing MICE, available at `https://blogs.imperial.ac.uk/mice/2017/09/01/introducing-mice/`, accessed on 01/10/23.

[9] Francesco Maddalena, Liliana Tjahjana, Aozhen Xie, Arramel, Shuwen Zeng, Hong Wang, Philippe Coquet, Winicjusz Drozdowski, Christophe Dujardin, Cuong Dang, and Muhammad Danang Birowosuto. Inorganic, organic, and perovskite halides with nanotechnology for high–light yield x- and -ray scintillators. *Crystals*, 9(2), 2019.

[10] G. Bonomi, P. Checchia, M. D'Errico, D. Pagano, and G. Saracino. Applications of cosmic-ray muons. *Progress in Particle and Nuclear Physics*, 112:103768, 2020.

[11] R. E. Hall, D. A. Lind, and R. A. Ristinen. A simplified muon lifetime experiment for the instructional laboratory. *American journal of physics*, 38(10):1196–1200, 1970.

[12] Philip R. Bevington and D. K. Robinson. *Data reduction and error analysis for the physical sciences*. McGraw-Hill, London;Boston, Mass;, 3rd edition, 2003.

[13] J. Quinn, University College Dublin, Measurment Probability Distributions, available at `https://physicslabs.ucd.ie/docs/data_analysis/`, accessed on 01/10/23.

[14] LeCroy. *NIM Model 623B Octal Updating Discriminator*. available at `https://www.fnal.gov/projects/ckm/jlab/623b-spec.htm`, accessed on 24/09/23.

[15] Advanced Measurment Technology, Inc. *ORTEC Model 566 Time to Amplitude Converter (TAC) Operating and Service Manual*. available at `https://www.ortec-online.com/-/media/ametekortec/manuals/5/566-mnl.pdf?la=en&revision=9b548347-6150-4d97-af90-dbe8b40c1930`, accessed on 24/09/23.

[16] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[17] Advanced Measurment Technology, Inc. *ORTEC Model CO4020 Quad 4-Input Logic Unit Operating and Service Manual*. available at `https://www.ortec-online.com/-/media/ametekortec/manuals/c/co4020-mnl.pdf?la=en&revision=9d1f5054-75e8-4948-bf5b-6aa5bb2c9429`, accessed on 03/10/23.

[18] LeCroy. *LeCroy Model 222/222N Dual Gate Generator*. available at `https://wwwusers.ts.infn.it/~rui/univ/Acquisizione_Dati/Manuals/LRS%20222.pdf`, accessed on 24/09/23.

[19] D. F. Measday. The nuclear physics of muon capture. *Physics reports*, 354(4):243–409, 2001.

[20] A. M. Sirunyan, W. Waltenberger, E. Widl, V. Mossolov, L. Benucci, E. A. De Wolf, V. Adler, B. Clerbaux, A. P. R. Gay, L. Vanelderen, G. Bruno, V. Oguri, A. Santoro, F. Torres Da Silva De Araujo, S. Piperov, C. H. Jiang, R. Plestina, Z. Zinonos, J. Härkönen, J. Rander, D. Sprenger, T. Klimkovich, W. Haj Ahmad, M. Aldaya Martin, M. Bergholz, A. Parenti, D. Volyanskyy, G. Daskalakis, A. Laszlo, J. M. Kohli, M. Z. Mehta, S. Paktinat Mehdiabadi, N. Manna, B. Marangelli, S. Paoletti, D. Pedrini, S. Sala, T. Dorigo, A. T. Meneguzzo, M. Tosi, S. Rahatlou, S. Argiro, C. Biino, T. J. Kim, T. Sabonis, A. Sánchez Hernández, A. Morelos Pineda, M. Cwiok, A. David, L. Uvarov, A. Gribushin, L. Sarycheva, E. Calvo, J. M. Vizan Garcia, J. Gonzalez Sanchez, L. Scodellaro, E. Auffray, C. Bernet, D. Giordano, E. Meschi, M. U. Mozer, M. Rovere, M. Simon, K. Gabathuler, J. Eugster, K. Freudenreich, M. . Sawley, C. Regenfus, I. Dumanoglu, M. Vergili, D. Cussans, C. K. Mackay, C. Brew, Z. Hatherell, T. Bose, A. Heister, P. Lawson, R. Cousins, S. Erhan, Y. Tu, S. Sharma, P. Avery, P. Sellers, S. Linn, P. Markowitz, M. Kirn, S. C. Tonwar, D. D'Enterria, M. Zanetti, D. M. Morse, J. Gu, M. Jones, K. Potamianos, S. Schnetzer, A. Gurrola, V. Khotilovich, A. Sill, D. Carlsmith, J. Klukas, D. Reeder, CMS Collaboration, and IL (United States) Fermi National Accelerator Lab. (FNAL), Batavia. Measurement of the charge ratio of atmospheric muons with the cms detector. *Physics letters. B*, 692(2):83–104, 2010.

[21] J. L. Autran, D. Munteanu, T. Saad Saoud, and S. Moindjie. Characterization of atmospheric muons at sea level using a cosmic ray telescope. *Nuclear instruments methods in physics research. Section A, Accelerators, spectrometers, detectors and associated equipment*,

903:77–84, 2018.

[22] J. E. Allen and A. J. Apostolakis. Sea-level cosmic ray spectra at large zenith angles. *Proceedings of the Royal Society of London. Series A, Mathematical and physical sciences*, 265(1320):117–132, 1961.

Appendix A: Python Notebook

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

from math import *
from scipy.optimize import curve_fit, minimize
from scipy.stats import poisson
from tqdm import tqdm
from glob import glob

from matplotlib.ticker import FormatStrFormatter
```

Loading Data

This section comprises the loading of data runs. The data is saved in sections of 1.2 hours and so a second function was created to load multiple files together.

Calibration

In [ ]:
```
channels = [73, 152, 234, 316, 395, 593, 807]
lifetimes_ns = [100, 200, 300, 400, 500, 750, 1000]

lifetimes_us = [el / 1000 for el in lifetimes_ns]

def Linear(x, m, c):
    return m*x + c

calib_pars, calib_cov = curve_fit(Linear, channels, lifetimes_us)

print(calib_pars)
print(np.sqrt(calib_cov[0][0]), np.sqrt(calib_cov[1][1]))

# Error on channel corresponds to peak curve on MCA, error on lifetime corresponds to the uncertainty on the pulse generator / oscilloscope.
plt.errorbar(channels, lifetimes_us, xerr=4, yerr=10/1000, fmt=".", lw=1, capsize=3, color="indianred", label="Calibration data")

plt.plot(np.arange(0, 900), Linear(np.arange(0, 900), calib_pars[0], calib_pars[1]), color="cornflowerblue", label="Least Squares fit")

plt.xlabel("Channel")
plt.ylabel("Lifetime ($\mu$s)")
plt.margins(0)

plt.legend()
```

In [ ]:
```
def ApplyCalibration(channelNumbers):
    lifetimes = []
    for channel in channelNumbers:
        lifetimes.append(calib_pars[0] * channel + calib_pars[1])

    return lifetimes
```

In [ ]:
```
def LoadExperimentRun(path, numberOfChannelsToRemove, calibrate=False):

    channelCounts = np.loadtxt(path)[numberOfChannelsToRemove[0]:-numberOfChannelsToRemove[1]]

    channelNumbers = np.arange(numberOfChannelsToRemove[0], len(channelCounts)+numberOfChannelsToRemove[0])

    if calibrate is True:
        channelNumbers = ApplyCalibration(channelNumbers)

    return (channelNumbers, channelCounts)

def LoadMultipleRuns(basePath, filenamePrefix, numberOfChannelsToRemove, calibrate=False):
    #print(basePath + filenamePrefix + "*.txt")
    filePaths = glob(basePath + filenamePrefix + "*.txt")

    filePaths.sort()
    print(f"Loading:\n{filePaths}")

    channelsSample = LoadExperimentRun(filePaths[0], numberOfChannelsToRemove, calibrate)[0]
    countsSample = LoadExperimentRun(filePaths[0], numberOfChannelsToRemove)[1]

    channelCounts = np.zeros(np.shape(countsSample))
    for filePath in filePaths:
        channelCounts = channelCounts + LoadExperimentRun(filePath, numberOfChannelsToRemove, calibrate)[1]

    occurances = 0
    for el in channelCounts:
        occurances += el


    print(f"{occurances} occurances")

    return (channelsSample, channelCounts)
```

In [ ]:
```
# Example loading
data = LoadMultipleRuns("../data/", "autoDetection_T-800mV_40ns_DMH_", (100, 1), calibrate=False)

plt.scatter(data[0], data[1], marker=".", color="cornflowerblue")
plt.xlabel("Channel")
plt.ylabel("Counts")
```

Data Analysis Pipeline

Re-Binning Data

In [ ]:
```
def RebinData(data, binFraction, method="mean", verbose=False):
    # Inputs are the data output from LoadExperimentRun, and a binFraction which is a number between 0 and 1 which determines how large the bins are

    bins = list(data[0])
    data = list(data[1])

    binWidth = (bins[-1] - bins[0]) * binFraction

    lowerBound = bins[0]
    numberOfBins = floor((bins[-1] - bins[0]) / binWidth)

    newBins = np.arange(lowerBound, binWidth * (numberOfBins + 1), binWidth)
    newBins = [el + binWidth / 2 for el in newBins]
    #newBins = [floor(el) for el in newBins]

    #topBound = floor(newBins[-1] + binWidth)
    #newBins.append(topBound)

    binnedData = []

    for i, binBound in enumerate(newBins):
        #print(f"{i}/{len(newBins)}")
        if i < len(newBins)-1:
            #binnedData.append(list(data[newBins[i]:newBins[i+1]]))

            binnedData.append(np.array(data)[np.where((bins > newBins[i]) & (bins < newBins[i+1]))])

    # binnedData is a an array which each row containing a bin
    binnedData = np.array(binnedData, dtype="object")

    averagedBinnedData = []
    binnedDataLowerUncertainty = []
    binnedDataUpperUncertainty = []
    for row in tqdm(binnedData, disable= not verbose):

        avg = AverageDataInBin(row, method, verbose=verbose)
        averagedBinnedData.append(avg[0])
        binnedDataLowerUncertainty.append(avg[1])
        binnedDataUpperUncertainty.append(avg[2])

    return (newBins[:-1], averagedBinnedData, binnedDataLowerUncertainty, binnedDataUpperUncertainty)
```

Methods for Averaging Data

In [ ]:
```
def AverageDataInBin(dataInBin, method="mean", verbose=False):

    if method == "mean":
        averagedRow = sum(dataInBin) / len(dataInBin)
        uncertainty = np.std(dataInBin)

        upperUncertainty = lowerUncertainty = uncertainty / np.sqrt(len(dataInBin))


    elif method == "gaussian":
        averagedRow = np.mean(dataInBin)
        variance = np.sqrt(averagedRow)
        standardError = variance / np.sqrt(len(dataInBin))

        upperUncertainty = lowerUncertainty = standardError

    elif method == "MLE":

        # Finds the maximum likelihood for mu in a poisson distribution
        # To do this we can find the minimum of the negative log likelihood
        model = minimize(MLE_negativeLogLikelihood, np.max(dataInBin) / 2, args=dataInBin, method="Nelder-Mead")

        averagedRow = model.x
        lowerUncertainty, upperUncertainty = poisson.interval(0.68, averagedRow) # 1 sigma uncertainty
        lowerUncertainty, upperUncertainty = (lowerUncertainty / np.sqrt(len(dataInBin)), upperUncertainty / np.sqrt(len(dataInBin)))

        if verbose:
            print(averagedRow, uncertainty)

    return averagedRow, lowerUncertainty, upperUncertainty


# Method of Maximum Likelihood Estimation - Calculate the likelihood of the pmf function of a given mu for each datapoint in the bin
def MLE_negativeLogLikelihood(lamda, data):

    logLikelihoods = []
    for el in data:
        logLikelihoods.append(poisson.pmf(k=el, mu = lamda))

    return -1 * np.sum(logLikelihoods)


def FindFitParameters(data, FittingFunction, sigma, initialPars=None):

    if sigma == None:
        pars, cov = curve_fit(FittingFunction, data[0], data[1], initialPars)

    else:
        pars, cov = curve_fit(FittingFunction, data[0], data[1], initialPars, sigma=sigma, absolute_sigma=True)

    return pars, cov

def ExponentialCurve(x, a, b, c, d):
    y = []

    for point in x:
        y.append(a * exp(- b * point + c) + d)

    return y

#, [10, -0.01, 0.1, 15]
```

In [ ]:
```
# Example rebinning

data = LoadMultipleRuns("../data/", "autoDetection_T-800mV_40ns_DMH_", (100, 1), calibrate=False)

rebinnedData = RebinData(data, binFraction=0.05, method="gaussian")

fig, axes = plt.subplots(2, 1, figsize=(6, 12))
ax1, ax2 = axes

ax1.scatter(data[0], data[1], marker=".", color="cornflowerblue")
ax1.set_title("Raw Data")

ax2.errorbar(rebinnedData[0], rebinnedData[1], yerr=rebinnedData[2], fmt=".", color="cornflowerblue")
ax2.set_title("Binned Data, size: 5%")

for ax in axes:
    ax.set_ylim(0, 40)
    ax.set_xlabel("Channels")
    ax.set_ylabel("Counts")

plt.tight_layout()
```

Analysis

Finding optimum bin sizes

In [ ]:
```
fakeDataX = np.linspace(0, 1000, 1000)
fakeDataY = ExponentialCurve(fakeDataX, 1, 0.01, 0, 0)


rebinnedFakeData_20 = RebinData((fakeDataX, fakeDataY), binFraction=0.20, method="gaussian")
rebinnedFakeData_15 = RebinData((fakeDataX, fakeDataY), binFraction=0.15, method="gaussian")
rebinnedFakeData_10 = RebinData((fakeDataX, fakeDataY), binFraction=0.10, method="gaussian")
rebinnedFakeData_5 = RebinData((fakeDataX, fakeDataY), binFraction=0.05, method="gaussian")

fig, axes = plt.subplots(2, 2, figsize=(8,8), sharex=True, sharey=True)
axes = axes.reshape(4)

titles = ["20% bin size", "15% bin size", "10% bin size", "5% bin size"]
fakeData = [rebinnedFakeData_20, rebinnedFakeData_15, rebinnedFakeData_10, rebinnedFakeData_5]

for i, ax in enumerate(axes):
    ax.plot(fakeDataX, fakeDataY, zorder=0, color="cornflowerblue")

    ax.errorbar(fakeData[i][0], fakeData[i][1], yerr=fakeData[i][2], color="indianred", fmt=".", capsize=3, linewidth=1)

    ax.set_title(titles[i])

axes[2].set_xlabel("x (arb.)")
axes[3].set_xlabel("x (arb.)")
axes[0].set_ylabel("y (arb.)")
axes[2].set_ylabel("y (arb.)")

plt.tight_layout()
```

Find Best Bin Size

This section handles the plotting of the uncertainties on the parameters of a least squares fit for **any input function**. This plot can be used to estimate the most optimal bin-size.

In [ ]:
```
def FindOptimumBinFraction(data, binFractions, FitFunction=ExponentialCurve, method="gaussian", initialPars=None):

    parameterUncertainties = []

    print("Assessing bin fractions")
    for binFrac in binFractions:
        #print(f"Testing binFrac: {binFrac}", end="\r")
        rebinnedData = RebinData(data, binFrac, method=method)

        x = rebinnedData[0]
        y = rebinnedData[1]
        y = np.squeeze(y)

        tol = 0.01
        newSigma = []
        for el in rebinnedData[2]:
            if el < tol:
                newSigma.append(tol)
            else:
                newSigma.append(el)

        pars, cov = FindFitParameters((x,y), FitFunction, sigma=newSigma, initialPars=initialPars)

        for i in range(len(cov)):
            parameterUncertainties.append(np.sqrt(cov[i][i]))

    # Normalise and make array of uncertainties

    parameterUncertainties = np.array(parameterUncertainties).reshape(len(binFractions), len(cov))

    #for i, uncertainties in enumerate(parameterUncertainties.T):

        #parameterUncertainties[:,i] = [el / max(uncertainties) for el in uncertainties]

    return (binFractions, parameterUncertainties)
```

In [ ]:
```
data = (fakeDataX, fakeDataY)

binFractions = np.linspace(0.06, 0.01, 1000)

_, parameterUncertainties = FindOptimumBinFraction(data, binFractions, initialPars=[1, 0.01, 0, 0])

#plt.plot(binFractions, parameterUncertainties.T[0])

fig, axes = plt.subplots(4, 1, figsize=(6,8), sharex=True)

fig.text(0.01, 0.5, 'Fit Parameter Normalised Uncertainty', va='center', rotation='vertical')

parameterNames = ["a", "b", "c", "d"]
yLimits = [0.25, 0.25, 0.25, 0.25]

for ax, uncertainties, parameterName, yLim in zip(axes, parameterUncertainties.T, parameterNames, yLimits):

    uncertainties = np.array(uncertainties)

    uncertainties[uncertainties == inf] = np.nan

    uncertainties = uncertainties / np.nanmax(uncertainties)

    ax.plot(binFractions, uncertainties, color="indianred")

    ax.margins(0)
    ax.set_ylabel(parameterName, rotation=0)


    #ax.set_ylim(0, yLim)
    ax.axvline(x=0.045,ymin=0,ymax=1,c="cornflowerblue", linestyle="dashed", clip_on=False, label="Chosen Bin Fraction")

    if ax == axes[-1]:
        ax.set_xlabel("Bin Size (fraction of total number of ch.)")

    if ax == axes[0]:
        ax.set_title("Fit parameter uncertainties for: $f(x) = a e^{(b x + c)} + d$")
        ax.legend()
```

Gaussian Approximation of the Poisson Distribution

In [ ]:
```
data = LoadMultipleRuns("../data/", "autoDetection_T-800mV_40ns_DMH_", (100, 200), calibrate=True)

binFraction = 0.045
binSize = binFraction * data[0][-1]
print(f"Bin Size: {binSize:0.2f} us")

rebinnedData = RebinData(data, binFraction, method="gaussian", verbose=False)

x = rebinnedData[0]
y = rebinnedData[1]
y = np.squeeze(y)

yErr = np.array(list(zip(rebinnedData[2], rebinnedData[3]))).T
yErr = np.squeeze(yErr)
```

In [ ]:
```
gPars, gCov = FindFitParameters((x,y), ExponentialCurve, sigma=rebinnedData[2], initialPars=[15, 0.05, 0.1, 3])

fig, ax = plt.subplots(1, 1, figsize=(6,8))

print(gPars)

fitCurveX = np.linspace(0, max(x), 1000)
ax.plot(fitCurveX, ExponentialCurve(fitCurveX, gPars[0], gPars[1], gPars[2], gPars[3]), color="cornflowerblue", label="Least Squares Fit")

#ax.plot(fitCurveX, ExponentialCurve(fitCurveX, 15, -0.05, 0.1, 3), color="orange", label="Test Function")

ax.errorbar(x, y, yerr=yErr, fmt=".", capsize=3, linewidth=1, color="indianred", label="Gaussian Averaged Data")

ax.set_xlabel("Decay Time ($\mu s$)")
ax.set_ylabel("Counts")
ax.legend()
ax.set_title(f"Gaussian Approximation\nBin Size: {binSize:0.2f} microseconds")
ax.margins(x=0)
ax.hlines([0], 0, max(x) + 0.01*max(x), color="grey", ls="dashed")
#ax.set_yscale("log")
```

Half-life estimation

In [ ]:
```
decayConst = gPars[1]

print(f"Half-life: {log(2) / decayConst:0.5f} us")

decayConstantUncertainty = np.sqrt(gCov[1][1])

print(f"Uncertainty (propagated from fit): {np.sqrt(decayConstantUncertainty**2 * (log(2) / decayConst**2)**2)}")

print()

print(f"Lifetime: {1/ decayConst:0.5f} us")
print(f"Uncertainty (propagated from fit): {np.sqrt(decayConstantUncertainty**2 * (1 / decayConst**2)**2)}")
```

Maximum Likelihood Estimation

In [ ]:
```
data = LoadMultipleRuns("../data/", "autoDetection_T-800mV_40ns_DMH_", (90, 200), calibrate=True)

binFraction = 0.045
binSize = binFraction * data[0][-1]
print(f"Bin Size: {binSize:0.2f} us")

rebinnedData = RebinData(data, binFraction, method="MLE", verbose=False)

x = rebinnedData[0]
y = rebinnedData[1]
y = np.squeeze(y)

yErr = np.array(list(zip(rebinnedData[2], rebinnedData[3]))).T
yErr = np.squeeze(yErr)
```

In [ ]:
```
sig = []
for a, b in zip(np.squeeze(rebinnedData[2]), np.squeeze(rebinnedData[3])):
    sig.append(np.sqrt(a**2 + b**2))


pars, cov = FindFitParameters((x, y), ExponentialCurve, sigma=sig, initialPars=[19, 0.05, 0.1, 1])

print(pars)

fig, ax = plt.subplots(1, 1, figsize=(6,8))

fitCurveX = np.linspace(0, max(x), 1000)

#ax.plot(fitCurveX, ExponentialCurve(fitCurveX, 18, -0.05, 0.1, 1), color="orange", label="Test Function")

ax.plot(fitCurveX, ExponentialCurve(fitCurveX, pars[0], pars[1], pars[2], pars[3]), color="cornflowerblue", label="Least Squares Fit")

ax.errorbar(x, y, yerr=yErr, fmt=".", capsize=3, linewidth=1, color="indianred", label="MLE averaged data")

ax.set_xlabel("Decay Time ($\mu s$)")
ax.set_ylabel("Counts")
ax.legend()
ax.set_title(f"Maximum Likelihood Estimation Method (Poisson)\nBin Size: {binSize:0.2f} microseconds")
ax.margins(x=0)
ax.hlines([0], 0, max(x) + 0.01*max(x), color="grey", ls="dashed")
```

Half-life estimation

In [ ]:
```
decayConst = pars[1]

print(f"Half-life: {log(2) / decayConst:0.5f} us")

decayConstantUncertainty = np.sqrt(cov[1][1])

print(f"Uncertainty (propagated from fit): {np.sqrt(decayConstantUncertainty**2 * (log(2) / decayConst**2)**2)}")

print()

print(f"Lifetime: {1/ decayConst:0.5f} us")
print(f"Uncertainty (propagated from fit): {np.sqrt(decayConstantUncertainty**2 * (1 / decayConst**2)**2)}")
```

Muon flux as a function of Zenith Angle

In [ ]:
```
zenithData = np.loadtxt("../data/zenithAngles.txt", skiprows=1)

angles = zenithData[:,0] # degrees
anglesUncertainty = 2 # degrees

counts = zenithData[:,1]
countsUncertainties = [np.sqrt(mean) for mean in counts]

time = zenithData[:,2] # seconds
timeUncertainty = 1 # seconds, based on my ability to stop the timer and the couter at the same time.

countRates = [c / t for c, t in zip(counts, time)]
countRateUncertainties = [np.sqrt(timeUncertainty**2 * c**2 / t**4 + uc**2 / t**2) for c, uc, t in zip(counts, countsUncertainties, time)]

fig, ax = plt.subplots(1, 1, figsize=(8,8))

ax.errorbar(angles, countRates, xerr=anglesUncertainty, yerr=countRateUncertainties, fmt=".", linewidth=1, capsize=3, color="cornflowerblue")
ax.set_xlabel("Zenith Angle ($^\circ$)")
ax.set_ylabel("Countrate (counts / second)")
```