

Muon Lifetime Measurement

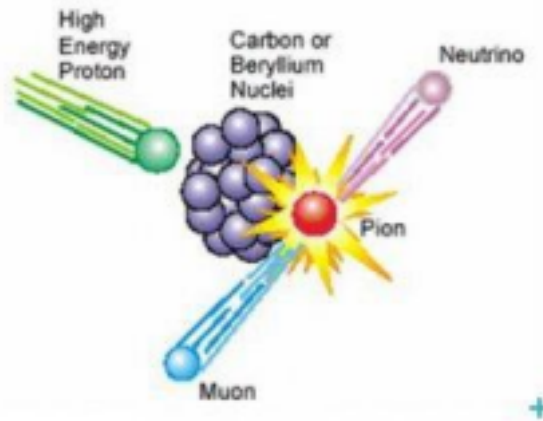
Burkan Bereketoğlu

Introduction

Muons are one of the elementary particles and also one of the core components of matter. They are found by Anderson and Neddermeyer in 1937 by exposing a cloud chamber to cosmic rays coming to the earth. In the upper layers of the atmosphere, there are primary cosmic rays that are mainly consisting of protons, that collide and interact with nitrogen and oxygen in the atmosphere and then turn into secondary particles, protons as their daughter particles, create showers of particles via strong and EM interactions. Later on these, secondary particles, some of them, create muons that decay with weak interactions. Many of the particles created other than muons and also some of the muons vanish before even get close to the sea level surface. However, some muons can touch the earth's surface. Muons are different from the protons and the secondary particles that are created; they can only decay via EM and are weak, but they can interact with the matter only with EM, hence they lose energy or momentum. This loss is described by the Bethe-Bloch equation. Furthermore, to add more detail to the muon case one can say that, muons are different from electrons because they are approximately 200 times heavier than the electron but other than that, similar to e^- . The cosmic ray muons that are used in the experiment are the muons that are created in the upper layers of the atmosphere and some of these muons have the chance to pass the atmosphere layers before vanishing and reach sea-level surfaces. The ones that reach the sea level surfaces can be measured further of their attributes in a laboratory such as CERN. To add further into muons, the fact that muons have 200 times more mass than electrons actually makes muons unstable (Muheim, n.d.).

This leads them to decay by weak force mostly exclusively to an electron or positron with two neutrinos in both, the difference of an electron and positron occur due to the muon's spin. Muon has $\frac{1}{2}$ spin, but it can be negative or positive. If positive, then there will be a positron, else there will be an electron after the decay. Here are the decay formulas:

$$\begin{aligned}\mu^- &\rightarrow e^- + \nu_e^- + \nu_\mu^- \\ \mu^+ &\rightarrow e^+ + \nu_e + \nu_\mu\end{aligned}$$



[1] (Muon beam creation, 2019)

Moreover, the decay time probability for the muon is a Poisson distribution decay time probability. Hence it follows an exponential decay path. The distribution of decay of muon is the activity function, which is;

$$N(t) = N_0 \cdot e^{-\left(\frac{t}{\tau}\right)} \quad (3)$$

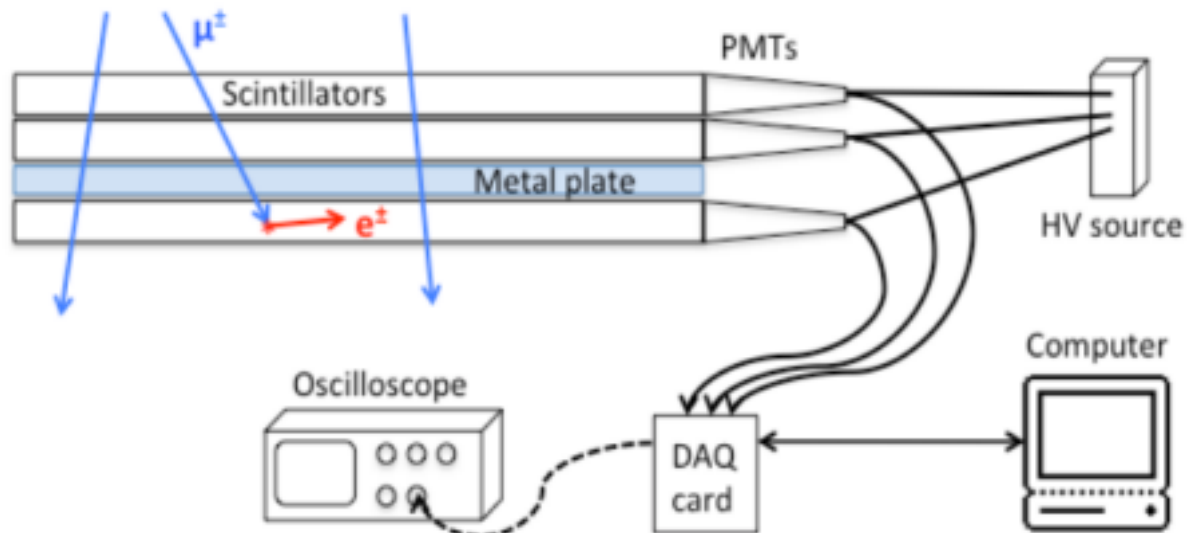
Equation 3, given above τ is the mean lifetime or real lifetime in the second step used in this experiment, and t is the vanishing time after 0 for each specific muon. For all t is different but is the same. N_0 is the initial number of the muons for the experiment, and the $N(t)$ function is the number of the muons not vanished in the system at the time t .

Experimental Setup

The experiment concluded a single basic plastic scintillator used for simplification and lowering the costs and to make the experiment accessible to undergraduate students and scientists from different areas of physics. Furthermore, python and python data acquisition and data analysis packages are used in the experiment to further process and understand the data and deduce a meaningful outcome. The abovementioned experiment is a sample experiment conducted by Damir Bosnar(2018), but for its reachability and quality of the explanations given used by the author of this report.

In the results, none of the abovementioned systems were used, moreover data that is taken before the pandemic is used to make the analysis. Still, the Python programming language is used for deduction and data analysis, but not the ones mentioned above. Pandas, Numpy, Matplotlib, Math, and Random libraries of Python are used in the experiment.

Here you can see the sample experimental setup in figure 2:



[2] (Bosnar, 2018)

As mentioned above, we use a scintillator, but the usage of a scintillator is not only important but how it gathers muons and what is a scintillator is also important.

A scintillator is can be described as a material that brightens up or glows when a high-energy photon or charged particle etc. So, if you put photomultiplier tubes that are connected to a data acquisition system that is connected to a visualization system can show you the decay of the count of the muons in a muon shower.

The how part is about the kinetic energy and remaining range, if the kinetic energy of the muon is below the necessary amount to finish the remaining range then the muon will enter to scintillator and the process starts. The range is here is measured in $x = \rho s$ in which s is the thickness in cm and ρ is the density in g/cm^3 . so basically x has units in g/cm^2 , which makes it easier to make a comparison of energy loss and range between materials with different densities. The muons that are positive just stop in the scintillator and decay, however, negative ones can bind, make atomic nuclei and form muonic atoms. Due to such bonds of negative muons can interact weakly with protons before they decay. If the differentiation is made clear with positive and negative muons the lifetime of free muons can be measured successfully. However, the data is really simple and already taken so the measurement methods were different and discussed in later contents (Bosnar, 2018).

Algorithm & Result

The approach used to make a logical outcome from the data to get lifetime value, mean lifetime measurement methods are used. To get a mean lifetime, two different approaches are used. One of which is a simple approach by just using the randomized time before vanishing by a sampling function used then, these randomized arrays turned into means inside

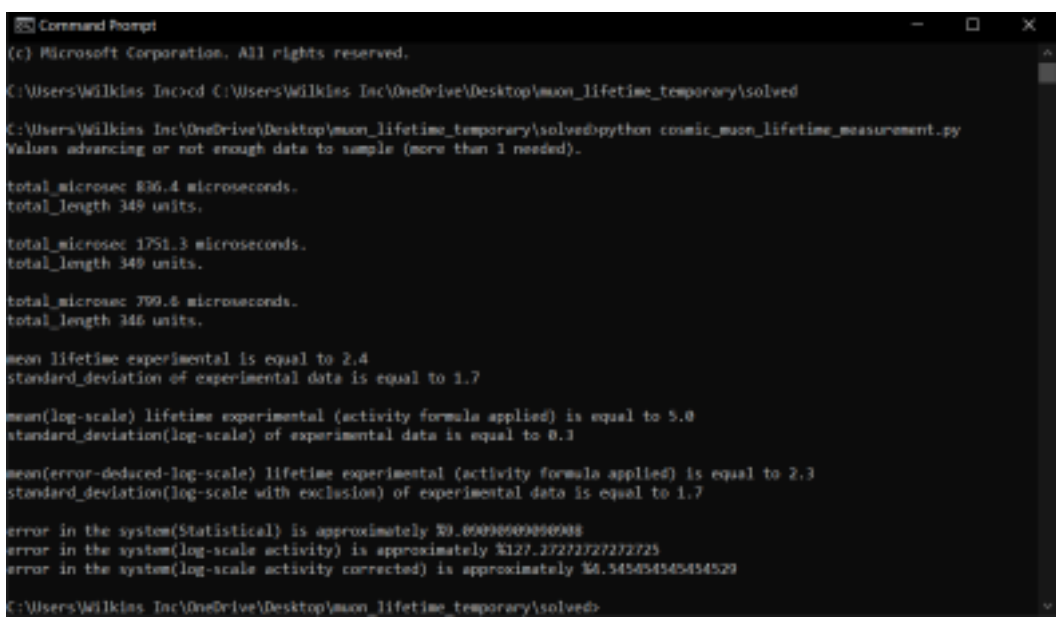
their second step lists before making one and only mean the randomized values in specific intervals made their own means. Later on, with their own number of values, these values are added together and divided into the total data to get the real mean. Basically, the real experimental mean was measured in two functions and two steps after making a sampling for uniformity and randomness.

The method used for sampling is random. The sample method is from a random library, which only accepts integer values. Still, our values are decimal, so to use it for our values and for further smaller values, I increased the values by 10^6 then divided them by 10^6 after sampling was done for each value.

For means, as mentioned above, only a regular mathematical operation is done, and the values are added to a new list in each function. In the end, there is also a standard deviation function that works by first finding the variance and taking the square root of the variance, so one can easily take the variance values in the middle of the standard deviation function.

For the second method, the Activity function mentioned in the introduction is used to understand the mean from the actual mean. Due to the small number of data and colossal noise compared to the data, the Activity function worked poorly before noise reduction. Still, after the noise reduction, the system worked fine and even better than the regular mean and standard deviation findings from the data. The activity function and mean function are the same functions, just appended to different lists with different formulas. This is done for a clearer vision of the code.

Here is the result of the algorithm that is used, it is an excerpt of the command prompt of Windows 10 with all the necessary result values explicitly shown.



```
Command Prompt
(c) Microsoft Corporation. All rights reserved.
C:\Users\Wilklis Inc>cd C:\Users\Wilklis Inc\OneDrive\Desktop\muon_lifetime_temporary\solved
C:\Users\Wilklis Inc\OneDrive\Desktop\muon_lifetime_temporary\solved>python cosmic_muon_lifetime_measurement.py
Values advancing or not enough data to sample (more than 1 needed).
total_microsec 836.4 microseconds.
total_length 349 units.
total_microsec 1751.3 microseconds.
total_length 349 units.
total_microsec 799.6 microseconds.
total_length 346 units.
mean lifetime experimental is equal to 2.4
standard_deviation of experimental data is equal to 1.7
mean(log-scale) lifetime experimental (activity formula applied) is equal to 5.0
standard_deviation(log-scale) of experimental data is equal to 0.1
mean(error-deduced-log-scale) lifetime experimental (activity formula applied) is equal to 2.3
standard_deviation(log-scale with exclusion) of experimental data is equal to 1.7
error in the system(Statistical) is approximately %0.0909090909090909
error in the system(log-scale activity) is approximately %127.37272727272727
error in the system(log-scale activity corrected) is approximately %6.545454545454529
C:\Users\Wilklis Inc\OneDrive\Desktop\muon_lifetime_temporary\solved>
```

[3]

Conclusion

To conclude as can be seen from the excerpt from the command prompt, one can clearly see the effectiveness of the statistical approach in a noisy environment. But one should also look at the difference between noise redacted versions of the activity formula applied system. Before the redaction, the error is really high and even more than %100, which is not acceptable. For particle physics between %5- %10 is acceptable levels, below %5 is really great but not achieved in most new experiments. Muon lifetime measurement is not a new system hence below %5 can be asked. The deviation is too high because the mean is 2.4.

In the end, one could easily go with the statistical part without hesitating but if a better, neat program is written the basic statistical method is just too easy to use and not precise enough. For big data and small variance, the activity approach is better. The error margin is lower, which shows better preciseness. Other than analysis, one can also deduce that there are several different ways to find the mean lifetime for muons and should use the one that is better for their own system. If data is small or variance is big, there is something different and for the opposite, there is something different is offered in the hands of statistics.

References

Bosnar, D. (2018, January 23). A simple setup for cosmic muon lifetime measurements. IOPscience.

<https://iopscience.iop.org/article/10.1088/1361-6404/aaadec>

Muheim, F. (n.d.). *Muon lifetime experiment*. Muheim Projects. Retrieved June 2, 2021, from

<https://www2.ph.ed.ac.uk/%7Emuheim/teaching/projects/>

Muon beam creation. (2019, December 20). [Illustration]. How Can Muons Be Used for Research? <https://nmi3.eu/service/print-template-artid=150.html>

Code script

```
import pandas as pd
import random
import math
import matplotlib.pyplot as plt
import numpy as np

def tuples(x,y): # tuple making function for ease of access.
    a = []
    if len(x) == len(y):
        for i in range(len(y)):
            a.append((x[i],y[i]))
    else:
        print('give me equal length lists.')

    return a

def sampling(i,n): # sampling to create randomness.
    if ((len(i) is not 0 and len(n) is not 0) and (len(i) is not 1 and len(n) is not 1)):
        sample = random.sample(range(int(100000*i[-2]), int(100000*i[-1])), abs(n[-2]-n[-1]))
        # multiplied with 100000 to have fit enough values in sampling.
        sample = [x/100000 for x in sample]
        return sample
    else:
        print('Values advancing or not enough data to sample (more than 1 needed). \n')

def mean_find(n,y,x): # pre-processor for sampling to make means
    mean = []
    length = []
    log_R = [] # values before the first 3 measurement errors not excluded
    ex_log_R = [] # values after the first 3 measurement errors excluded
    #ex_mean = []
    for i in range(len(n)):
        if ((n[i] is not None) and (len(n[i]) != 0)):
            log_R.append(((y[i]/real_muon_lifetime)+math.log(x[i]/y[i]))/y[i])
            mean.append(sum(n[i])/len(n[i]))
            length.append(len(n[i]))

    for i in range(3,len(log_R)):
        ex_log_R.append(log_R[i])

    #for i in range(0,len(mean)-3):
        #ex_mean.append(mean[i]) Used to experiment if statistical approach changes with exclusion of data.

    return mean, length, log_R, ex_log_R

def real_mean_calculator(i,x): # pre_processed means in their interval are processed to produce one
    mean total_len = 0
    total_microsec = 0
    mean = 0
    for z in range(len(i)):
        total_microsec += i[z]*x[z]
        total_len += x[z]
    mean = total_microsec/total_len
    print('total_microsec {} microseconds.'.format(round(total_microsec,1)))
    # since real value has one(1) decimal point
    print('total_length {} units.\n'.format(total_len))
    return round(mean,1) # since real value has one(1) decimal point

def standard_dev(n,i): # to find standard deviation
    standard_dev = 0
    for k in i:
        standard_dev += k-n
    standard_dev = standard_dev/(len(i)-1)
    standard_dev = math.sqrt(standard_dev)
    return round(standard_dev,1) # since real value has one(1) decimal point
```

```

colnames = ['Time(microsecond)', 'Muon Count'] # giving names to Dataframe.
data = pd.read_csv('Burkan-cosmic-muon-lifetime-data.csv', names = colnames, header = None)
df = pd.DataFrame(data) # turning csv into dataframes for convenience in turning to lists

pre_time = df['Time(microsecond)'].tolist() # before deleting header of the time column in list
time = [] # full list version after header of time.
pre_count = df['Muon Count'].tolist() # before deleting header of the muon count column in list
count = [] # full list version after header of count.

real_muon_lifetime = 2.2 # according to nyu, wikipedia and so.
error_point = 0 # for simplicity

samples = [] # sampling values in tuples for each data.
rem = [] # take the time values from the tuple to use in sampling
mu_rem = [] # take the muon count values from the tuple to use in sampling

for i in range(1,len(pre_time)):
    time.append(float(pre_time[i])) # the process into turning to lists without header

for i in range(1,len(pre_count)):
    count.append(int(pre_count[i])) # the process into turning to lists without header

time_count = tuples(time,count) # Tuple of microsecond, muon count, more convenient than lists.
#print(time_count)

for i in time_count:
    mu_rem.append(i[1])
    rem.append(i[0])
    samples.append((sampling(rem,mu_rem)))

#print(samples)

pre_mean, pre_length, pre_Log_R, pre_ex_log_r = mean_find(samples,time,count)
#print(pre_mean) # mean lifetime for each list
#print(pre_length) # counts in that lifetime of list
#print(pre_Log_R) # measuring with  $R = R_0 * e^{-(t/T)}$ 

mean = real_mean_calculator(pre_mean,pre_length) # Statistical approach on the mean lifetime

#ex_mean = real_mean_calculator(pre_mean,pre_length) # to see if statistical approach has error. None
found. #print(ex_mean)

standard_deviation = standard_dev(mean,rem) # standard deviation
"""
Two different scenarios.
"""
log_r = real_mean_calculator(pre_Log_R, pre_length)
excluded_log_r = real_mean_calculator(pre_ex_log_r, pre_length)

standard_deviation_log_r = standard_dev(log_r,rem)
standard_deviation_excluded_log_r = standard_dev(excluded_log_r,rem)

for i in range(len(rem)):
    if (mean==rem[i]):
        error_point = mu_rem[i]

for i in range(len(rem)):
    if (log_r == rem[i]):
        error_point_2 = mu_rem[i]

for i in range(len(rem)):
    if (excluded_log_r == (rem[i]-0.1)):
        error_point_3 = (mu_rem[i]+mu_rem[i-1])/2

error = 100*((mean-real_muon_lifetime)/real_muon_lifetime) # error between real and experimental
error_log_r = 100*((log_r-real_muon_lifetime)/real_muon_lifetime) # error between real and experimental(log_r)
error_ex_log_r = 100*((excluded_log_r-real_muon_lifetime)/real_muon_lifetime) # error between real and experimental(excluded_log_r)

```

```

# Decay-Fit

decay_func_fit = [] # decay function fit
n_0 = count[0] # to show to decay function on the fit
time_fit = np.linspace(0,10,1000) # Not accurate but representative
for i in time_fit:
    decay_func_fit.append(n_0*math.e**(-i/real_muon_lifetime))

#Print values

print('mean lifetime experimental is equal to {}'.format(mean)) # mean lifetime
print('standard_deviation of experimental data is equal to {}'.format(standard_deviation)) # standard deviation lifetime

#For different experiments

print('mean(log-scale) lifetime experimental (activity formula applied) is equal to {}'.format(log_r)) # mean lifetime
print('standard_deviation(log-scale) of experimental data is equal to {}'.format(standard_deviation_log_r)) # standard deviation in log_r lifetime

#For different experiments

print('mean(error-deduced-log-scale) lifetime experimental (activity formula applied) is equal to {}'.format(excluded_log_r)) # mean lifetime (w/ error
exclusion)
print('standard_deviation(log-scale with exclusion) of experimental data is equal to {}'.format(standard_deviation_excluded_log_r)) # standard deviation in
log_r lifetime (w/ error exclusion)

# Plotting

plt.figure(1) # first figure
plt.plot(rem,mu_rem, label = u"Muon Count vs Microseconds (\u03bcs)", color= 'navy', alpha = 0.75)
plt.plot(time_fit,decay_func_fit, label = "Decay Fit", color= 'orange', alpha= 0.75, ls= '-')
plt.axvline(x= mean, label= u"Mean lifetime experimental {} \u03bcs".format(mean),color= 'red', ls = '-')
plt.axvline(x= real_muon_lifetime, label= u"Mean lifetime actual {} \u03bcs".format(real_muon_lifetime),color= 'purple', ls= '-')
plt.title("Muon Lifetime Measurement")
plt.xlabel(u"Microseconds (\u03bcs)")
plt.ylabel("Muon Count")
plt.errorbar(mean, error_point, xerr= standard_deviation, color='green', marker='s', mfc='red',
             mec='green', ms=6, mew=4, capsize= 5)
plt.grid()
plt.legend()
plt.savefig("Muon Lifetime Measurement")
print('error in the system(Statistical) is approximately {}'.format(error))

"""
Activity formula and differe
"""

plt.figure(2) # second figure
plt.plot(rem,mu_rem, label = u"Muon Count vs Microseconds (\u03bcs)", color= 'navy', alpha = 0.75)
plt.plot(time_fit,decay_func_fit, label = "Decay Fit", color= 'orange', alpha= 0.75, ls= '-')
plt.axvline(x= log_r, label= u"Mean lifetime experimental(log-scale) {} \u03bcs".format(log_r),color= 'red', ls = '-')
plt.axvline(x= real_muon_lifetime, label= u"Mean lifetime actual {} \u03bcs".format(real_muon_lifetime),color= 'purple', ls= '-')
plt.title("Muon Lifetime Measurement_log_r")
plt.xlabel(u"Microseconds (\u03bcs)")
plt.ylabel("Muon Count")
plt.errorbar(log_r, error_point_2, xerr= standard_deviation_log_r, color='green', marker='s', mfc='red',
             mec='green', ms=6, mew=4, capsize= 5)
plt.grid()
plt.legend()
plt.savefig("Muon Lifetime Measurement_log_r")
print('error in the system(log-scale activity) is approximately {}'.format(error_log_r))

"""
Third is similar to second but with exclusion effect.
"""

plt.figure(3) # third figure
plt.plot(rem,mu_rem, label = u"Muon Count vs Microseconds (\u03bcs)", color= 'navy', alpha = 0.75)
plt.plot(time_fit,decay_func_fit, label = "Decay Fit", color= 'orange', alpha= 0.75, ls= '-')
plt.axvline(x= excluded_log_r, label= u"Mean lifetime experimental(log-scale) {} \u03bcs".format(excluded_log_r),color= 'red', ls = '-')
plt.axvline(x= real_muon_lifetime, label= u"Mean lifetime actual {} \u03bcs".format(real_muon_lifetime),color= 'purple', ls= '-')
plt.title("Muon Lifetime Measurement_log_r_after_correction")
plt.xlabel(u"Microseconds (\u03bcs)")

```



```

plt.ylabel("Muon Count")
plt.errorbar(excluded_log_r, error_point_3, xerr= standard_deviation_excluded_log_r, color='green', marker='s', mfc='red',
            mec='green', ms=6, mew=4, capsize= 5)
plt.grid()
plt.legend()
plt.savefig("Muon Lifetime Measurement_log_r_after_correction")
print('error in the system(log-scale activity corrected) is approximately {}'.format(error_ex_log_r))

"""
if len(pre_mean) == len(pre_length): #To check if mean and length len is equal.
    print('dance')

#from uncertainties import ufloat can be used for uncertainty

"""
#print(time_count[len(time_count)-1]) # to understand the len concept.
#print(time)

#filter error in 0.0 - 0.2 and 0.2-0.4
#print(count)

"""
There is an error due to first third count in the system it rise from 132-80 to 80-135
which shouldn't happen normally and will be discussed in the report and presentation.

One can make the measurement better by using sample of random generated
means and find an average value of those random systems, but I didn't want to
change the randomness and wanted to show the effect of the randomness in the
data. """

```