Muon_Physics

May 28, 2019

1 Muon Physics

__Measurement of muon lifetime and flux

University of California, Santa Barbara, Physics, 93117 Goleta, CA Yuning Zhang

In [103]:
```
import numpy as np
import matplotlib.pylab as plt
from datetime import datetime
from scipy.optimize import curve_fit
```

In [104]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [105]:
```
def read_data(file_name,delimiter=" "):
    with open(data_path+file_name) as f:
        data=list(map(lambda x:int(x.strip().split(delimiter)[0]),f.readlines()))
    return np.array(data)
```

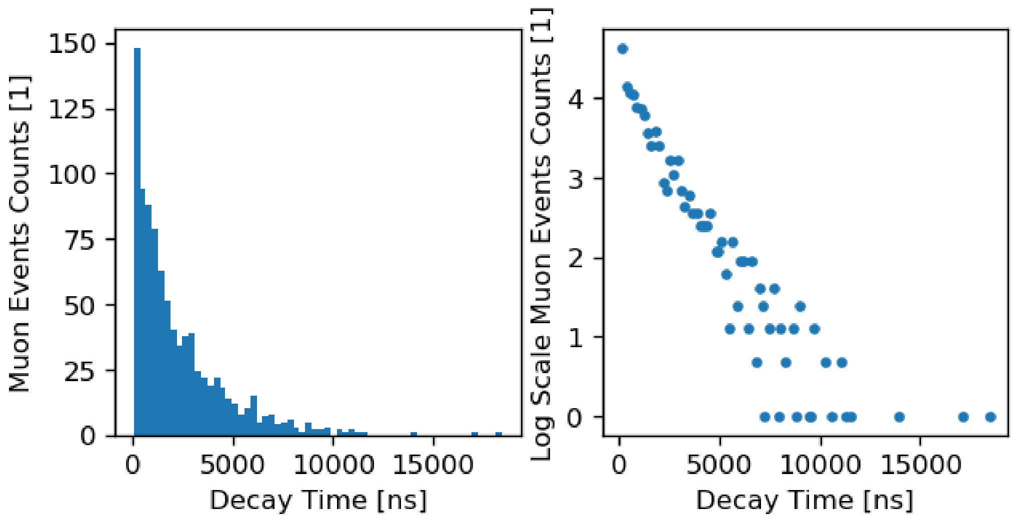
In [106]:
```
def filter_data(data,low_bound,up_bound):
    '''
    return filtered data array between lower bound and upper bound.
    the unit of the boundary value is nanosecond
    eg: 6 uSec = 6000 nSec
    '''
    return data[np.vectorize(lambda x: low_bound<x<up_bound)(data)]
```

In [107]:
```
def get_hist_data(data,N_bins,precision_error=20):
    '''
    filter data with the upper and lower bound
    given the bins number of histogram, return the counts and average in each bin,
    with the error of counts and standard deviation of average value
    '''
    N_data=len(data)
    bin_counts,bin_partitions=np.histogram(data,bins=N_bins)
    labels=np.digitize(data,bins=bin_partitions[1:-1])
    bin_sums=np.zeros(N_bins)
    for i in range(N_data):
```

```
            bin_sums[labels[i]]+=data[i]
        bin_means=bin_sums/bin_counts

        bin_square_errors=np.zeros(N_bins)
        for i in range(N_data):
            mean=bin_means[labels[i]]
            bin_square_errors[labels[i]]+=(data[i]-mean)**2
        bin_stds=np.sqrt(bin_square_errors/bin_counts+precision_error**2)

        xdata=bin_means[~np.isnan(bin_means)]
        ydata=bin_counts[bin_counts!=0]
        xerror=bin_stds[~np.isnan(bin_stds)]
        yerror=np.sqrt(ydata)
        return (xdata/1000,ydata,xerror/1000,yerror)
```

```
In [108]: precision_error=20 #ns
          data_path="./data/"#"19-05-20-14-08.data"
          file_name="19-05-22-18-03.data"#"19-05-20-14-08.data"#"19-05-02-17-41.data"#"05_13_Muo
          test_data=read_data(file_name,delimiter=" ")[5000:]
```

```
In [109]: filtered=filter_data(test_data,low_bound=40,up_bound=20000)
          ratio=1
          f=plt.figure()
          f.dpi=120
          ax1=f.add_subplot(121)
          plt.ylabel("Muon Events Counts [1]")
          plt.xlabel("Decay Time [ns]")
          ax1.hist(filtered,bins=60)
          xdata,ydata,xerror,yerror=get_hist_data(filtered,N_bins=100)
          ax2=f.add_subplot(122,sharex=ax1)
          plt.xlabel("Decay Time [ns]")
          plt.ylabel("Log Scale Muon Events Counts [1]")
          ax2.plot(xdata*1000,np.log(ydata),".")
          for ax in [ax1, ax2]:
              xmin, xmax = ax.get_xlim()
              ymin, ymax = ax.get_ylim()
              ax.set_aspect(abs((xmax-xmin)/(ymax-ymin))*ratio, adjustable='box-forced')
          plt.show()
          print("Total Events Number: ",len(test_data))
          print("Muon Events Number: ",len(filtered))
```



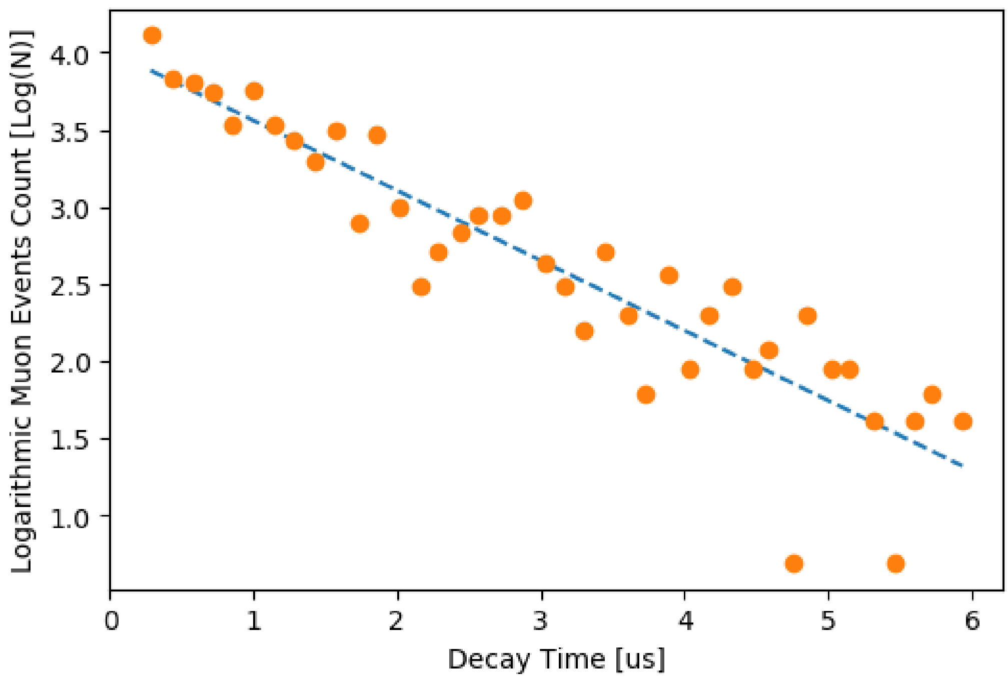

```
Total Events Number:  58775
Muon Events Number:  899
```

In [110]:
```
# Comments:

# A.
# The figure shows clearly that the impact of noise is more significant at
# larger time zone, where the effective count of muons is relatively smaller.
# The long tail data make no sense because the minimun count of events is 1 thus
# when the at the time area where muon decay probability is very smaller, the count
# we get from the histogram is actually random noise

# B.
# The peak at short time zone (100-120) also should be abandoned since
# the the precision of the apparatus is limited. The result is that all the muons
# with decay time less than the minimum resolution of the apparatus (20ns) will be
# counted in the same bin. The resolution at the short time area is relatively
# too coarse to depict a exponential boost curve.


# A successful fitting need to eliminate the noise and ineffective data
```

Linear Fitting

In [111]:
```
filtered=filter_data(test_data,low_bound=200,up_bound=6000)
xdata,ydata,xerror,yerror=get_hist_data(filtered,N_bins=40)
opt_param=np.polyfit(xdata,np.log(ydata),1)
plt.figure().dpi=100
```

```
plt.ylabel("Logarithmic Muon Events Count [Log(N)]")
plt.xlabel("Decay Time [us]")
plt.plot(xdata,np.poly1d(opt_param)(xdata),"--")
plt.plot(xdata,np.log(ydata),"o")
plt.show()
```




```
In [112]: muon_life=abs(1/opt_param[0])
          print("muon life by tentative linear fitting:",muon_life) #ns
```

```
muon life by tentative linear fitting: 2.20297928231
```

Tentative Exponential Fitting

```
In [113]: def exp_model(x,A,lambd,B):
              return A*np.exp(-lambd*x)+B
          # here we didn't consider the constant B because the long tail is cut off
```

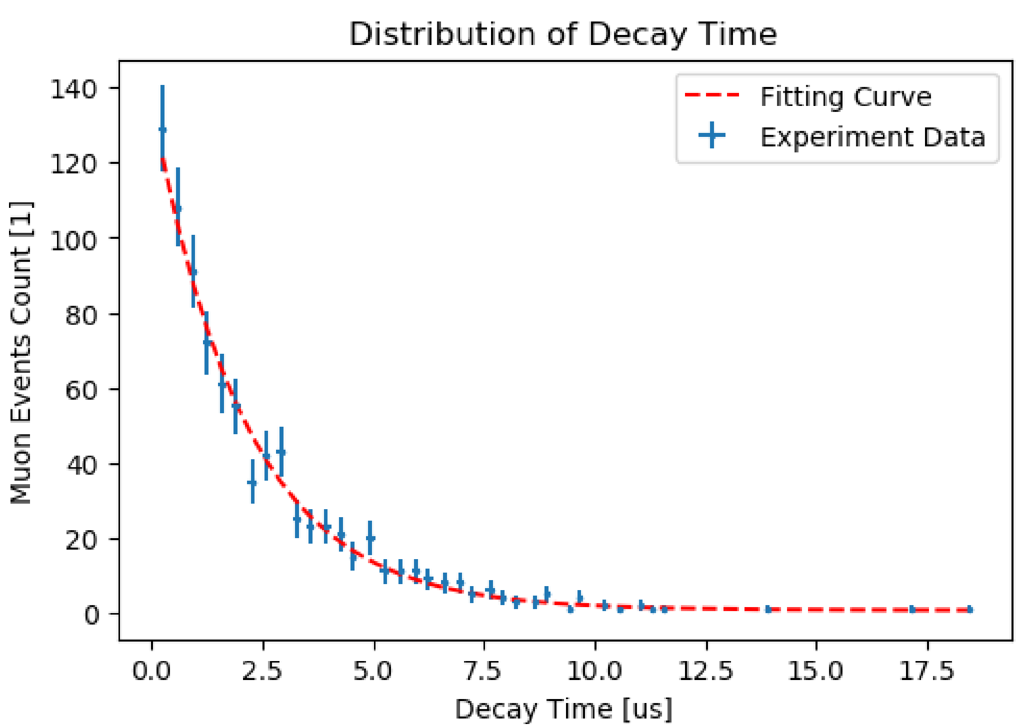
```
In [114]: filtered=filter_data(test_data,low_bound=80,up_bound=20000)
          xdata,ydata,xerror,yerror=get_hist_data(filtered,N_bins=55)
          opt_param,opt_pcov=curve_fit(exp_model,xdata,ydata,sigma=yerror,p0=(100,0.1,0))
          t_obs=1/opt_param[1]
          sigma_t_obs=muon_life*np.abs(np.sqrt(opt_pcov[1,1])/opt_param[1])
          print("opt_params:",opt_param)
          print("param_errors",np.array([np.sqrt(opt_pcov[i,i]) for i in range(len(opt_param))]))
          print("muon_life:",t_obs)
          print("uncertainty:",sigma_t_obs)#ns
```

```
opt_params: [ 135.53119481    0.47203618    0.72856033]
param_errors [ 5.37392331  0.01730273  0.28425711]
muon_life: 2.11848166686
uncertainty: 0.0807513407365
```

```
In [115]: t_neg=2.043
          sigma_neg=0.003
          t_pos=2*t_obs-t_neg
          sigma_t_pos=np.sqrt(2*sigma_t_obs**2+sigma_neg**2)
          print("corrected_muon_life:",t_pos)
          print("corrected_uncertainty:",sigma_t_pos)#ns
```

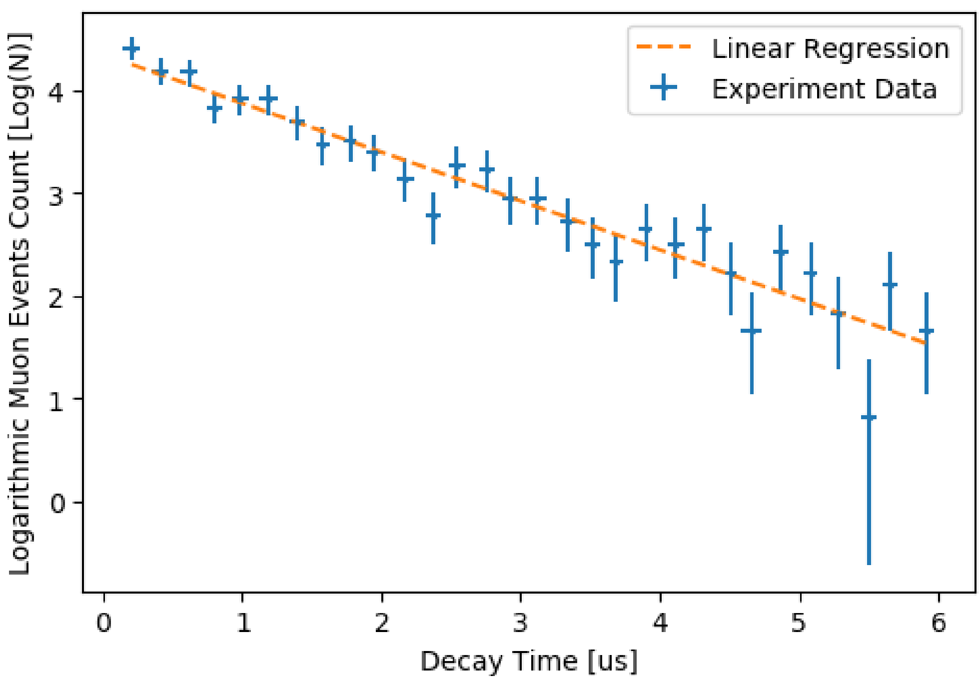
```
corrected_muon_life: 2.19396333371
corrected_uncertainty: 0.114239039131
```

```
In [116]: plt.figure().dpi=100
          plt.ylabel("Muon Events Count [1]")
          plt.xlabel("Decay Time [us]")
          plt.title("Distribution of Decay Time")
          plt.errorbar(xdata,ydata,xerr=xerror,yerr=yerror,fmt='.',markersize=3,label="Experimen
          fitting_ydata=(lambda x: exp_model(x,*opt_param))(xdata)
          plt.plot(xdata,fitting_ydata,"r--",label="Fitting Curve")
          plt.legend()
          plt.show()
```



Correct linear fitting using noise data

```
In [117]: filtered=filter_data(test_data,low_bound=100,up_bound=6000)
          xdata,ydata,xerror,yerror=get_hist_data(filtered,N_bins=30)
          ydata=np.abs(ydata-opt_param[2])
          param=np.polyfit(xdata,np.log(ydata),1)
          plt.figure().dpi=100
          plt.ylabel("Logarithmic Muon Events Count [Log(N)]")
          plt.xlabel("Decay Time [us]")
          plt.errorbar(xdata,np.log(ydata),xerr=xerror,yerr=[np.log(ydata)-np.log(ydata-yerror),
          plt.plot(xdata,np.poly1d(param)(xdata),"--",label="Linear Regression")
          plt.legend()
          plt.show()
```




```
In [118]: muon_life=abs(1/param[0])
          print("muon life by corrected linear fitting:", muon_life) #us

muon life by corrected linear fitting: 2.10209113983
```

```
In [119]: import pandas as pd
```

```
In [290]: shape_data=pd.read_excel(data_path+"data.xlsx",sheetname="Sheet3")
          precision=0.1 #cm
```

```
D,h,s,a,b,c,d,L,m=shape_data.mean() # mean value
sigma_D,sigma_h,sigma_s,sigma_a,sigma_b,sigma_c,sigma_d,sigma_L,sigma_m=np.sqrt(shape_
R,sigma_R=D/2,sigma_D/2
pd.concat([shape_data.mean(),np.sqrt(shape_data.std()**2+precision**2)],keys=["Mean","
```

Out[290]:

	Mean	StdDev
D[cm]	16.475000	0.111803
h[cm]	6.450000	0.115470
s[cm]	2.825000	0.160728
a[cm]	10.033333	0.115470
b[cm]	10.066667	0.152753
c[cm]	7.333333	0.182574
d[cm]	5.266667	0.270801
L[cm]	36.000000	0.173205
m[cm]	4.833333	0.182574

In [291]:
```
time_data=pd.read_excel(data_path+"data.xlsx",sheetname="Sheet1")
time_data
```

Out[291]:

	Angle[Deg]	Start Time	End Time	Delta_T	Delta_T.1	Count	Flux
1	90	17:05:21	17:19:18	00:13:57	837	170	0.203106
2	90	17:19:18	17:35:12	00:15:54	954	219	0.229560
3	70	16:36:30	17:01:23	00:24:53	1493	281	0.188212
4	60	17:37:20	17:46:55	00:09:35	575	104	0.180870
5	60	17:46:55	17:56:45	00:09:50	590	115	0.194915
6	60	11:40:45	11:57:20	00:16:35	995	177	0.177889
7	50	11:59:55	12:14:15	00:14:20	860	104	0.120930
8	50	12:14:15	12:29:55	00:15:40	940	100	0.106383
9	40	12:32:19	13:01:39	00:29:20	1760	140	0.079545
10	30	13:03:11	13:31:43	00:28:32	1712	92	0.053738

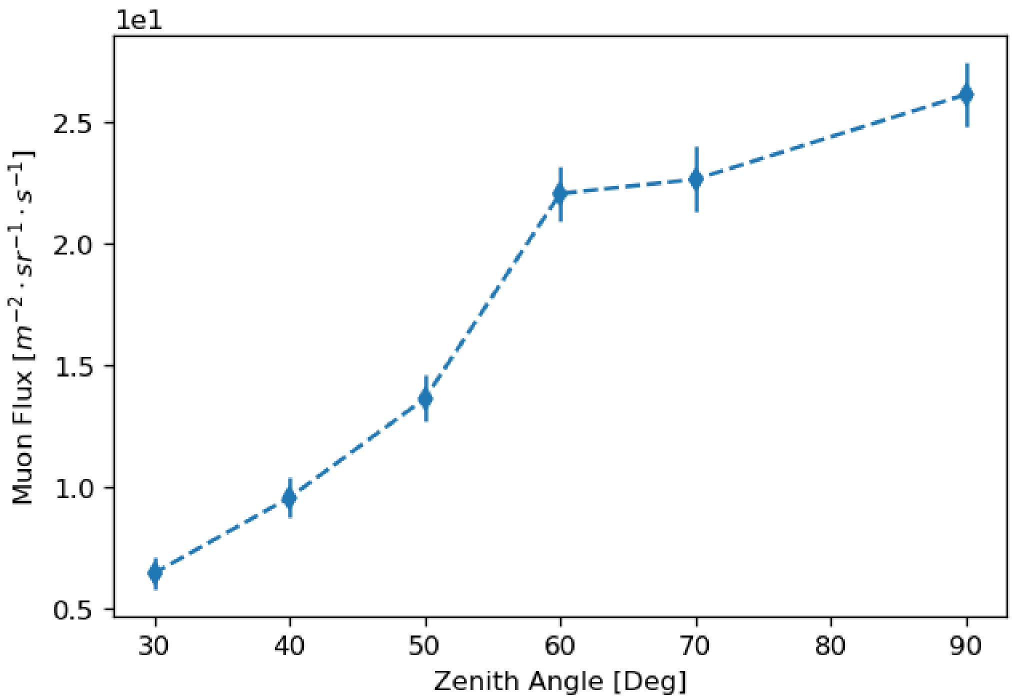
In [298]:
```
AOmega=83.0584
```

In [299]:
```
angle=[time_data.iloc[0,0]]
time=[time_data.iloc[0,4]]
count=[time_data.iloc[0,5]]
for i in range(1,len(time_data)):
    if time_data.iloc[i-1,0]==time_data.iloc[i,0]:
        time[-1]+=time_data.iloc[i,4]
        count[-1]+=time_data.iloc[i,5]
    else:
        angle.append(time_data.iloc[i,0])
        time.append(time_data.iloc[i,4])
        count.append(time_data.iloc[i,5])
angle=np.array(angle)
time=np.array(time)
count=np.array(count)
```

In [302]:
```
muon_flux=count/time/AOmega*100**2
flux_error=np.sqrt(count)/time/AOmega*100**2 # to m^2
```

```
plt.figure().dpi=120
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.xlabel("Zenith Angle [Deg]")
plt.ylabel(r"Muon Flux [$m^{-2}\cdot sr^{-1}\cdot s^{-1}$]")
plt.errorbar(angle,muon_flux,marker="d",linestyle="--",yerr=flux_error)
```

Out[302]: <Container object of 3 artists>



In [301]:
```
count*60/time/AOmega*2*np.pi # cm^-2 min^-1 # Standard value: 1 cm^-2 min^-1
```

Out[301]:
```
array([ 0.98582896,  0.85426787,  0.83212581,  0.51440505,  0.36104632,
        0.24391113])
```