Poisson Statistics Experiment

Burak Kara
Boğaziçi University • Physics Department

***Abstract*—The experiment aims to investigate behaviors of Poisson Distribution which can express almost all counting processes. In the experiment, we have used Cs-137 and Ba-133 to obtain the number of gamma radiations emitted in a 10-Seconds and 1-Second time intervals. We investigated the obtained data with both Gaussian Distribution and Poisson Distribution. We observed that for sufficiently big mean values the Poisson Distribution converges to the Gaussian Distribution.**

I. Theory

The Poisson distribution is a discrete probability distribution. Therefore, almost all counting processes can be expressed by the Poisson distribution[1]. It is named after the famous mathematician Siméon Denis Poisson.

The Poisson distribution is a special case of the Binomial Distribution [2]. Moreover, the Gaussian Distribution -also known as the normal distribution- is also a special case of the binomial distribution [3]. The Poisson and Gaussian distributions are indistinguishable when the mean of the distribution becomes sufficiently large [1].

In this experiment, we have both investigated the discrete counting process of two Gamma sources (Cs-137, Ba-133) and the indistinguishability of Gaussian and Poisson distributions according to formulas:

Binomial Distribution:

$$Pr(k;n,p) = \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k}, \tag{1}$$

where, n is Bernoulli trials
p is probability of success and k is success.

Poisson Distribution:

$$Pr(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{2}$$

where, λ is mean and k is success

Gaussian Distribution:

$$G(x,\sigma,\nu) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{3}$$

where, μ is the mean and σ is standard deviation.

However, the Poisson process has a different nature when it is being used to calculate the distribution of successive events.

Probability of n events in a t interval followed by another event within a dt time is

$$P_q(n+1,t) = \frac{(\alpha t)^n e^{-\alpha t}}{n!}[1] \tag{4}$$

where, $\alpha = \mu/t$

II. Method

We have used two different Gamma sources which are Caesium-137 and Barium-133. They have 30.2y and 10.5y half-life respectively (See. Fig.2-3). We have recorded the number of gamma radiation emitted from these two sources with a Geiger tube. We have analyzed the numbers of radiations according to the theory (See. Eq.2-3).

1) Cs-137 is placed.
2) Operating voltage of the Geiger tube is determined (460V in our case).
3) Recorded the number of counts for 10-s intervals.
4) Recorded the number of counts for 1-s intervals.
5) Ba-133 is placed.
6) Recorded the number of counts for 10-s intervals.
7) Recorded the number of counts for 1-s intervals.
8) Ba-133 is placed to obtain 1 count per second rate.
9) Distribution of successive counts is recorded with a chart recorder.

III. The Experimental Setup

- Geiger Counter.
- Gamma Ray Sources.
- Chart Recorder.

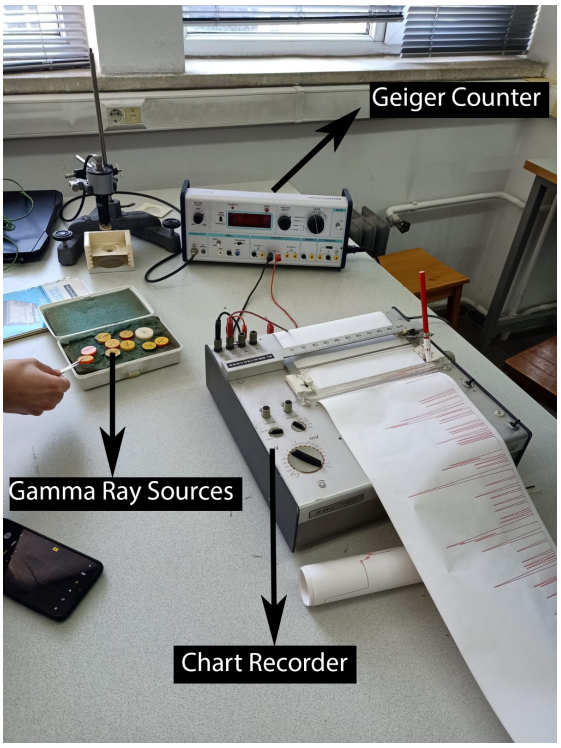

Fig. 1. Apparatus.

Fig. 2. Barium-133

Fig. 3. Caesium-137.

IV. The Data

TABLE I
NUMBER OF COUNTS OF SOURCES

cs-137 10-Sec	Cs-137 1-Sec	Ba-133 1-Sec	Ba-133 10-Sec
97	12	3	18
105	11	5	20
106	12	4	28
96	10	6	37
101	10	0	18
116	7	2	21
103	9	3	39
98	11	2	27
115	9	0	21
113	14	2	34
103	11	4	27
128	10	1	31
109	6	6	27
98	9	5	22
80	11	1	25
120	12	4	31
122	16	0	20
107	12	5	25
107	10	2	30
99	6	0	25
97	12	0	41
103	7	1	25
100	9	3	26
98	9	4	25
104	9	0	28
90	8	0	24
119	15	3	22
107	10	3	27
98	3	1	22
108	5	2	30
90	13	3	20
91	7	2	23
116	14	4	24

cs-137 10-Sec	Cs-137 1-Sec	Ba-133 1-Sec	Ba-133 10-Sec
102	11	4	25
91	9	1	23
109	7	2	28
95	8	0	21
115	9	3	23
101	8	2	17
103	14	3	28
86	8	2	26
110	15	4	31
96	13	1	27
93	10	5	17
101	12	3	22
117	12	2	32
114	5	1	21
96	13	3	17
106	6	1	14
112	11	1	29
112	4	2	32
119	8	2	25
94	11	3	25
104	10	6	22
105	9	2	26
122	10	2	29
110	11	3	21
107	13	0	25
86	9	3	34
115	7	1	29
110	11	0	21
106	12	2	30
109	12	1	20
129	6	2	25
106	8	2	24
119	12	2	35
81	15	1	27
96	10	2	19
102	10	0	24
103	8	3	24
128	6	8	24
110	12	1	29
109	12	3	19
131	11	1	30
82	11	4	24
107	15	3	30
103	15	1	20
103	10	1	23
99	11	3	24
114	9	2	27
95	10	3	33
102	12	6	26
86	12	3	29
107	14	5	23
109	11	4	27
94	13	2	20
88	13	1	31
108	8	1	22
114	13	2	26
92	16	3	24
98	17	1	23
101	14	3	21
104	8	4	28
109	9	2	15
112	14	2	14
98	10	4	22
96	11	2	18
106	13	3	20
106	7	2	17
86	13	1	22

We also obtained the data from the chart recorder by recording lengths between two spikes.

The data obtained from chart recorder in millimeter:

12-3-14-11-12-10-11-15-4-5-12-2-4-7-7-23-14-9-10-10-17-5-13-8-10-14-8-2-17-7-2-4-29-8-4-2-3-6-8-2-8-19-28-3-2-5-7-2-13-18-18-19-6-28-14-3-4-7-18-8-8-4-5-7-5-5-10-7-7-4-19-18-7-9-4-7-12-4-4-8-4-7-3-23-9-15-2-7-21-3-4-9-25-10-15-17-3-18-18-3-15-6-3-5-13-14-14-2-1-6-6-10

V. The Analysis

We have obtained the counts as a function of voltage applied during 100-Seconds interval and we have concluded to use 460V.

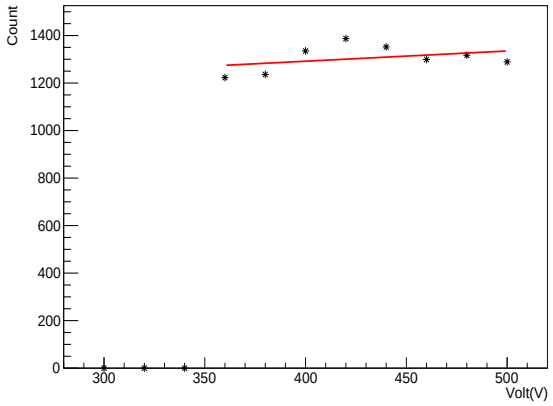

Fig. 4

We have calculated the Mean and Standard Deviation of the data sets. and we have plotted the $\frac{\sqrt{\mu}}{\sigma}$ as a function of μ. We expect the slope of the best fit line is 0 and y-intercept is 1.

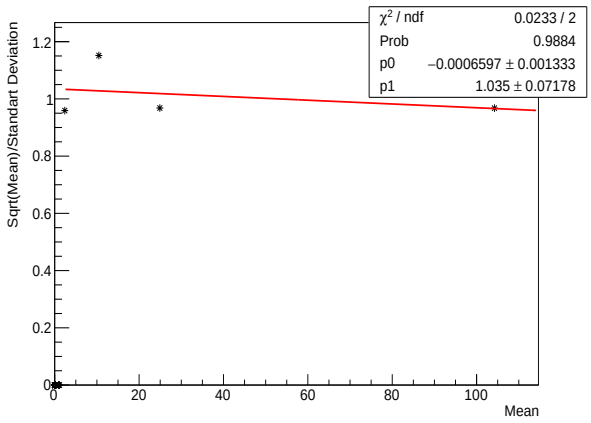

Fig. 5. The Slope of the line is -0.0007 +- 0.001 and The y-Intercept is 1.035 +- 0.07

We have fitted the Gaussian and Poisson distributions to the data sets. The mean, standard deviation, χ^2, and the fit parameters are shown on the plot legends.

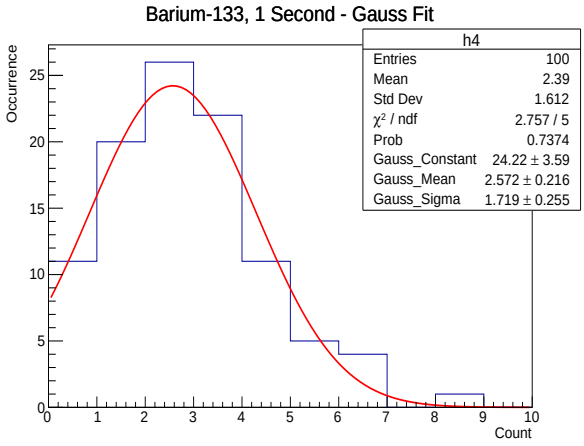

Fig. 6

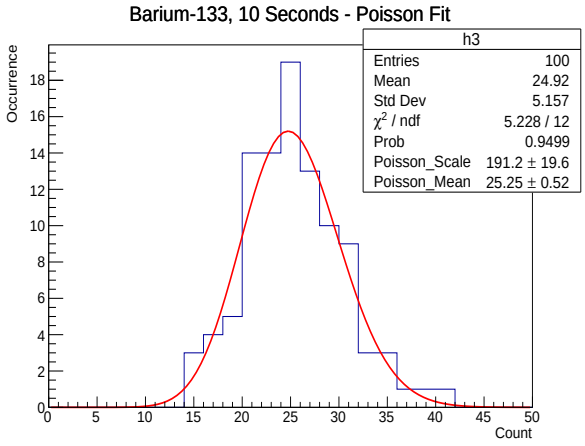

Fig. 9

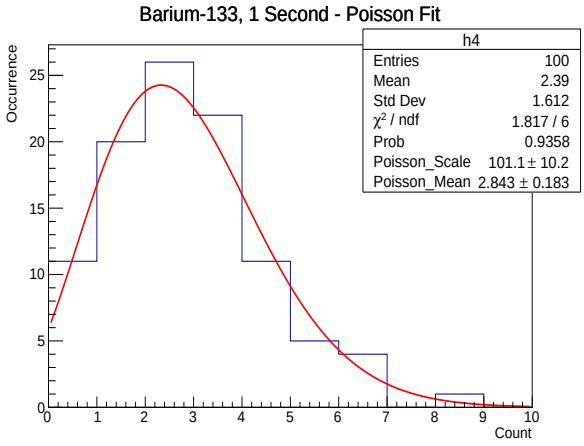

Fig. 7

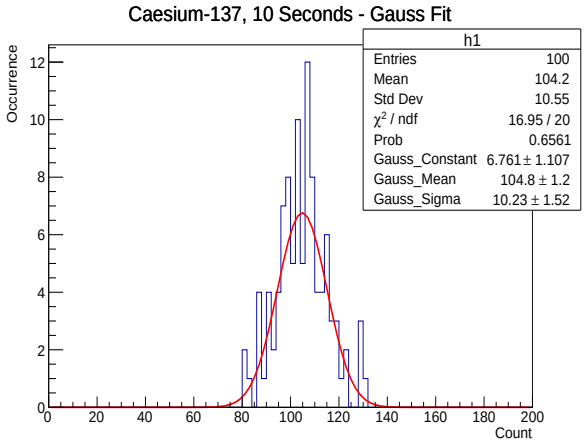

Fig. 10

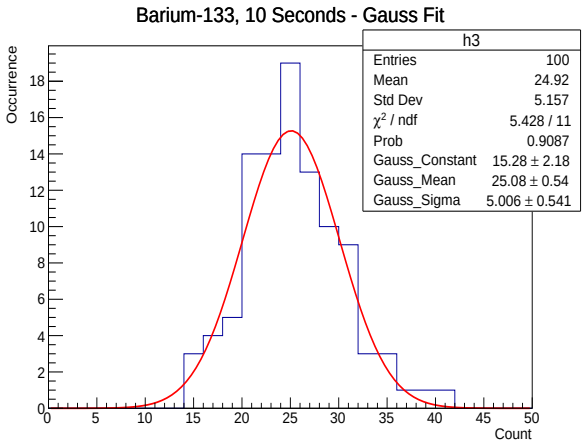

Fig. 8

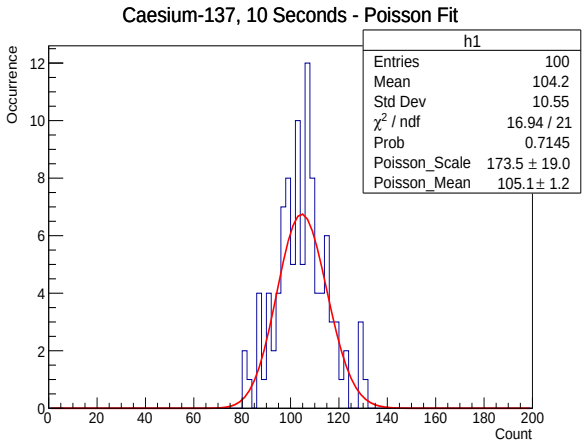

Fig. 11

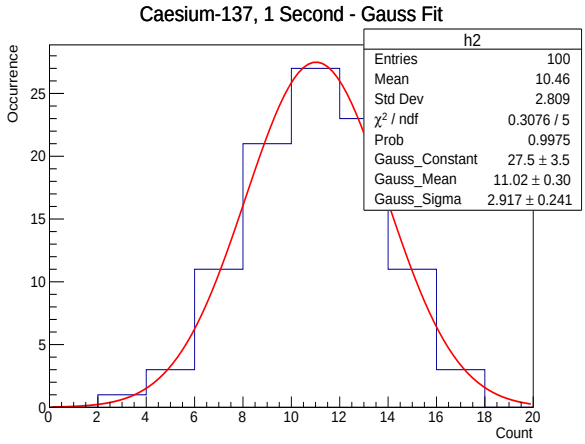

Fig. 12

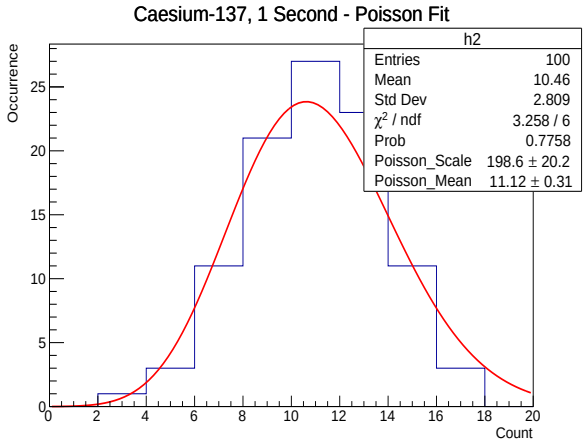

Fig. 13

We have used the modified Poisson distribution (See. Eq4) and the chart data to obtain following plots. The first plot is the n=0 case and the second one is the n=1 case.

We have calculated the true value of α value by dividing the number of spikes to the total length and we have found it to be $\alpha = 0.107$. (See. Appendix - Modified Poisson)

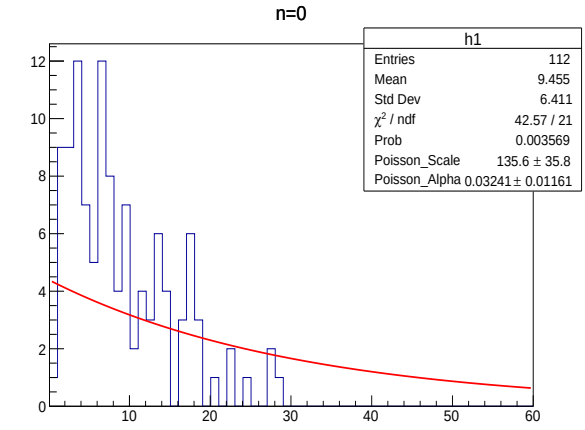

Fig. 14

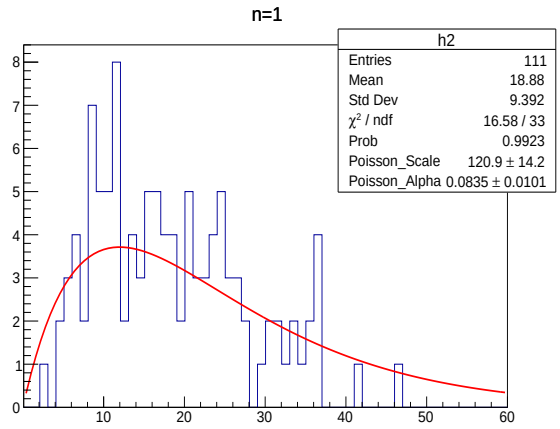

Fig. 15

VI. The Result

The fit of $\frac{\sqrt{\mu}}{\sigma}$ as a function of μ plot (See Fig.5) occurred as a horizontal line $y \simeq 1$.

We have confirmed that the Poisson Distribution and Gaussian Distributions become indistinguishable with increasing mean. As can be seen from Fig.6-13, the χ^2/ndf values become closer with higher means.

The Fig.14-15 shows the probability of n events in a t interval followed by another event. We have confirmed Eq.4 for the n=1 case; however, the n=0 case is problematic. The χ^2/ndf value is higher than acceptable.

VII. The Conclusion

We have investigated different behaviors of Poisson Distributions using various gamma sources and various time intervals. The first behavior we have confirmed is that the mean of the Poisson Distribution is equal to its standard deviation squared i.e. $\sigma^2 = \mu$. This behavior can be seen in Fig.5.

The second behavior we have observed is that the Poisson Distribution becomes indistinguishable with increasing mean. We have shown that the χ^2/ndf values become closer with an increasing mean (See. Fig-13). In other words, both Poisson Distribution and Gaussian Distributions can be considered as a good fit for a gamma decay. However, for sufficiently small mean values, Poisson Distribution becomes a better fit (See. Fig6-7).

The last behavior of Poisson Distribution we have investigated is the probability of n events in a t interval followed by another event as suggested by Eq.4. We have both investigated n=0 and n=1 cases (See. Fig13-14). The n=1 case was a success, the χ^2/ndf value is 0.5024. However, χ^2/ndf value of the n=0 case is not sufficiently close to 1. For the n = 0 case, we are 6 σ away from the true value α =0.107. For the n=1 case, we are 2.33 σ away. Therefore, we have concluded that we may miscalculate the lengths between spikes or could not see some spikes.

References

[1] E. Gülmez. *Advanced Physics Experiments*. 1st. Boğaziçi University Publications, 1999.

[2] *The Connection Between the Poisson and Binomial Distributions*. URL: https://mathcenter.oxford.emory.edu/site/math117/connectingPoissonAndBinomial/ (visited on 04/02/2023).

[3] *Derivation of Gaussian Distribution from Binomial*. URL: https://people.bath.ac.uk/pam28/Paul_Milewski,_Professor_of_Mathematics,_University_of_Bath/Past_Teaching_files/stirling.pdf (visited on 04/02/2023).

VIII. Appendix

The fit has been done with Root's built-in function (See Fig.4-14). We have used root release 6.28/00 for Ubuntu22.

Gauss, Poisson and Linear Fit(See Fig.5-13)

```
{
 TTree *tree = new TTree("tree", "tree");
 tree->ReadFile("count.csv");
 float cs10,cs1,br1,br10;
 tree->SetBranchAddress("cs10", &cs10);
 tree->SetBranchAddress("cs1", &cs1);
 tree->SetBranchAddress("br1", &br1);
 tree->SetBranchAddress("br10", &br10);

 //Initializing Arrays
 float *c1_10, *c1_1, *c2_1, *c2_10;
 int n = tree->GetEntries();
 c1_10 = new float[n];
 c1_1 = new float[n];
 c2_1 = new float[n];
 c2_10 = new float[n];

  //Filling Arrays
 for (int i = 0; tree->LoadTree(i) >= 0; i++){
  tree->GetEntry(i);
  c1_10[i] = cs10;
  c1_1[i] = cs1;
  c2_10[i] = br10;
  c2_1[i] = br1;
 }

  //Filling Histograms
 TH1F *h1 = new TH1F("h1","Caesium-137, 10
     Seconds - Gauss Fit",100, 0.0, 200);
   for (int i=0; i<n; i++) h1->Fill(c1_10[i]);
 h1->GetXaxis()->SetTitle("Count");
 h1->GetYaxis()->SetTitle("Occurrence");
 TH1F *h2 = new TH1F("h2","Caesium-137, 1
     Second - Gauss Fit",10, 0.0, 20);
   for (int i=0; i<n; i++) h2->Fill(c1_1[i]);
 h2->GetXaxis()->SetTitle("Count");
 h2->GetYaxis()->SetTitle("Occurrence");
 TH1F *h3 = new TH1F("h3","Barium-133, 10
     Seconds - Gauss Fit",25, 0.0, 50);
   for (int i=0; i<n; i++) h3->Fill(c2_10[i]);
 h3->GetXaxis()->SetTitle("Count");
 h3->GetYaxis()->SetTitle("Occurrence");
 TH1F *h4 = new TH1F("h4","Barium-133, 1
     Second - Gauss Fit",10, 0.0, 10);
   for (int i=0; i<n; i++) h4->Fill(c2_1[i]);
 h4->GetXaxis()->SetTitle("Count");
 h4->GetYaxis()->SetTitle("Occurrence");

 //Creating Clones For Poisson Fit
 TH1F* h1c = (TH1F*)h1->Clone();
 h1c->SetTitle("Caesium-137, 10 Seconds -
     Poisson Fit");
 TH1F* h2c = (TH1F*)h2->Clone();
 h2c->SetTitle("Caesium-137, 1 Second -
     Poisson Fit");
 TH1F* h3c = (TH1F*)h3->Clone();
 h3c->SetTitle("Barium-133, 10 Seconds -
     Poisson Fit");
 TH1F* h4c = (TH1F*)h4->Clone();
 h4c->SetTitle("Barium-133, 1 Second -
     Poisson Fit");

 //Initializing Gaus and Poisson Functions
 TF1 *f1 = new TF1("f1","gaus",0,200);
 TF1 *f2 = new TF1("f2",
     "[0]*TMath::Poisson(x,[1])",0,200);
 f1->SetParNames("Gauss_Constant","Gauss_Mean","Gauss_S
 f2->SetParNames("Poisson_Scale","Poisson_Mean");

 //Calculations for Linear Fit
 double x[4] = {h1->GetMean(), h2->GetMean(),
     h3->GetMean(), h4->GetMean()};
 double y[4] =
     {(sqrt(h1->GetMean())/h1->GetRMS()),
 (sqrt(h2->GetMean())/h2->GetRMS()),
 (sqrt(h3->GetMean())/h3->GetRMS()),
 (sqrt(h4->GetMean())/h4->GetRMS())};

 gStyle->SetOptFit(1111);

  //Fitting Functions
 TCanvas *c1 = new TCanvas(); // Linear Fit
 TGraph *mygraph = new TGraphErrors(n,x,y);
  mygraph->SetTitle("
      ;Mean;Sqrt(Mean)/Standart Deviation");
  mygraph->Draw("A*");
  TF1 *fnew = new
      TF1("fnew","[0]*x+[1]",2,120);
  fnew->SetParameters(0,1);
  mygraph->Fit(fnew, "R");

  TCanvas *c2 = new TCanvas(); //Gauss Fit
 f1->SetParameters(h1->GetMaximum(),
     h1->GetMean(), h1->GetRMS() );
 h1->Fit("f1");
 h1->Draw();

  TCanvas *c3 = new TCanvas(); //Poisson Fit
 f2->SetParameters(h1c->GetMaximum(),
     h1c->GetMean());
 h1c->Fit("f2", "R");
 h1c->Draw();

 TCanvas *c4 = new TCanvas(); //Gauss Fit
 f1->SetParameters(h2->GetMaximum(),
     h2->GetMean(), h2->GetRMS() );
 f1->SetRange(0.0,20);
 h2->Fit("f1");
 h2->Draw();

  TCanvas *c5 = new TCanvas(); //Poisson Fit
 f2->SetParameters(h2c->GetMaximum(),
     h2c->GetMean());
 f2->SetRange(0.0,20);
 h2c->Fit("f2", "R");
 h2c->Draw();

 TCanvas *c6 = new TCanvas(); //Gauss Fit
 f1->SetParameters(h3->GetMaximum(),
```

```
    h3->GetMean(), h3->GetRMS() );
 f1->SetRange(0.0,50);
 h3->Fit("f1");
 h3->Draw();

  TCanvas *c7 = new TCanvas(); //Poisson Fit
 f2->SetParameters(h3c->GetMaximum(),
    h3c->GetMean());
 f2->SetRange(0.0,50);
 h3c->Fit("f2", "R");
 h3c->Draw();

 TCanvas *c8 = new TCanvas(); //Gauss Fit
 f1->SetParameters(h4->GetMaximum(),
    h4->GetMean(), h4->GetRMS() );
 f1->SetRange(0.0,10);
 h4->Fit("f1");
 h4->Draw();

  TCanvas *c9 = new TCanvas(); //Poisson Fit
 f2->SetParameters(h4c->GetMaximum(),
    h4c->GetMean());
 f2->SetRange(0.0,10);
 h4c->Fit("f2", "R");
 h4c->Draw();

 //Creating Files
  c1->Print("linear.pdf");
  c2->Print("cs137-10-gauss.pdf");
  c3->Print("cs137-10-poiss.pdf");
  c4->Print("cs137-1-gauss.pdf");
  c5->Print("cs137-1-poiss.pdf");
  c6->Print("ba133-10-gauss.pdf");
  c7->Print("ba133-10-poiss.pdf");
  c8->Print("ba133-1-gauss.pdf");
  c9->Print("ba133-1-poiss.pdf");
}
```

Modified Poisson (See Fig.14-15)

```
{
 TTree *tree = new TTree("tree", "tree");
 tree->ReadFile("length.csv");
 float c;
 tree->SetBranchAddress("c", &c);

 //Initializing Arrays
 float *L;
 int n = tree->GetEntries();
 L = new float[n];

  //Filling Arrays
 for (int i = 0; tree->LoadTree(i) >= 0; i++){
  tree->GetEntry(i);
  L[i] = c;
 }

  //Calculating alpha value
  float time = 0;
  for (int i=0; i<n; i++){
  time += L[i];
  }
  float alpha = (n+1) / time;
  //Filling Historam n=0 case
  TH1F *h1 = new TH1F("h1","n=0",60, 0.1, 60);
   for (int i=0; i<n; i++) h1->Fill(L[i]);
  //Filling Histogram n=1 case
  TH1F *h2 = new TH1F("h2","n=1",60, 0.1, 60);
   for (int i=0; i<n-1; i++)
      h2->Fill(L[i]+L[i+1]);

 //Initializing Modified Poisson Functions
 TF1 *f1 = new TF1("f1",
    "[0]*[1]*exp(-[1]*x)", 0.1, 60); //n=0
 TF1 *f2 = new TF1("f2",
    "[0]*[1]*[1]*x*exp(-[1]*x)", 0.1,60);
 f1->SetParNames("Poisson_Scale","Poisson_Alpha");
 f2->SetParNames("Poisson_Scale","Poisson_Alpha");

 gStyle->SetOptFit(1111);
  cout << alpha << endl;

  //Fitting Functions
 TCanvas *c1 = new TCanvas();
 f1->SetParameters(1, alpha);
 h1->Fit("f1","R");
 h1->Draw();

 TCanvas *c2 = new TCanvas();
 f2->SetParameters(1, alpha);
 h2->Fit("f2","R");
 h2->Draw();

 //Creating Files
  c1->Print("length_0.pdf");
  c2->Print("length_1.pdf");
}
```

Linefit for Count vs Voltage (See Fig.4)

```
{
 const int ndata = 11;

 float x[ndata] = {300,320,340,360, 380,
    400,420,440,460,480,500};
 float y[ndata] =
    {0,0,0,1223,1236,1335,1387,1352,1299,1317,1289};

 TGraphErrors *mygraph = new
    TGraphErrors(ndata,x,y,0,0);
 mygraph->Draw("A*");
 TF1 *fnew = new
    TF1("fnew","[0]*x+[1]",360,500);
 fnew->SetParameters(0.01,1300);
 mygraph->Fit(fnew,"R");
}
```