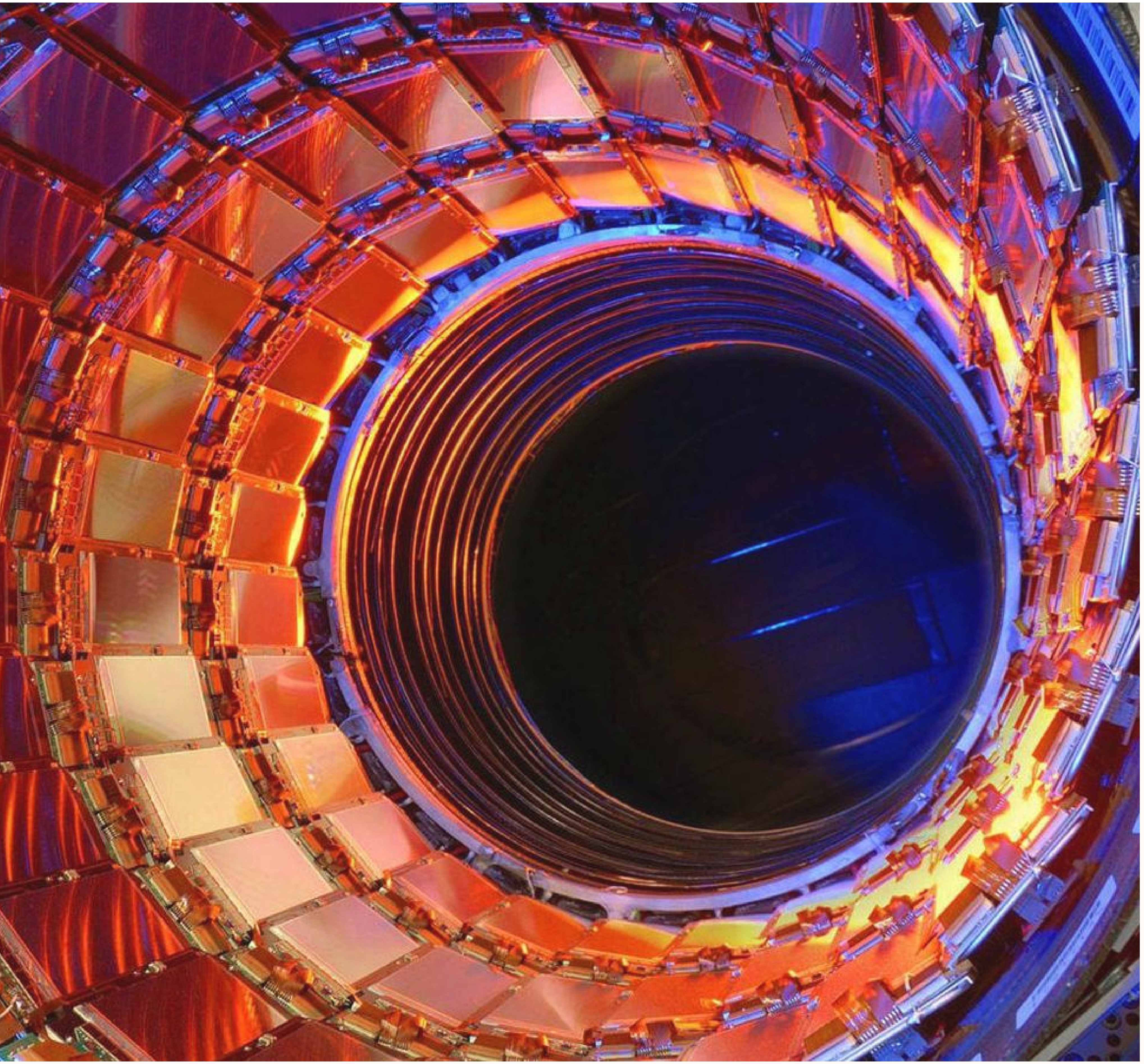


PROGRESS OF TASK 2: MPD PID PERFORMANCE BY MEANS OF IONIZATION LOSS DE/DX

ALEJANDRO SAN JUAN LOPEZ

REMEMBERING THE ACTIVITIES

1.- Simulate A+A collisions

- The runMC.C and runReco.C macros were used
- The BOX generator was used (1000 events and multiplicity equal to 100)
- The particles obtained were: Pi, e, P, K, d, He3.

2.- Optimize track selection criteria

- The MpdPtMCAnalysisTask.cxx macro was used
- Cutoffs were established for Pt, η and nhits.
- Code was written to obtain the dE/dx distributions for each particle.

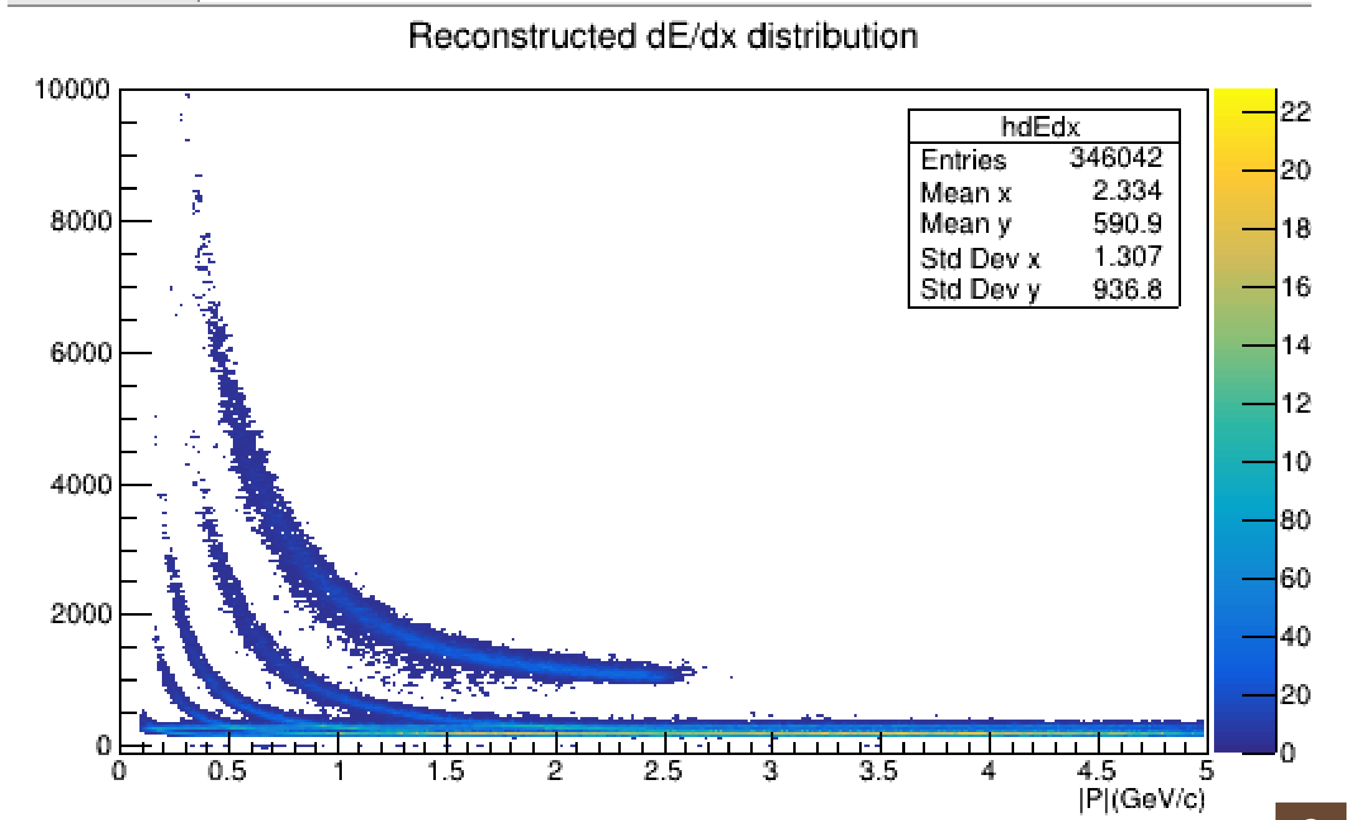
1

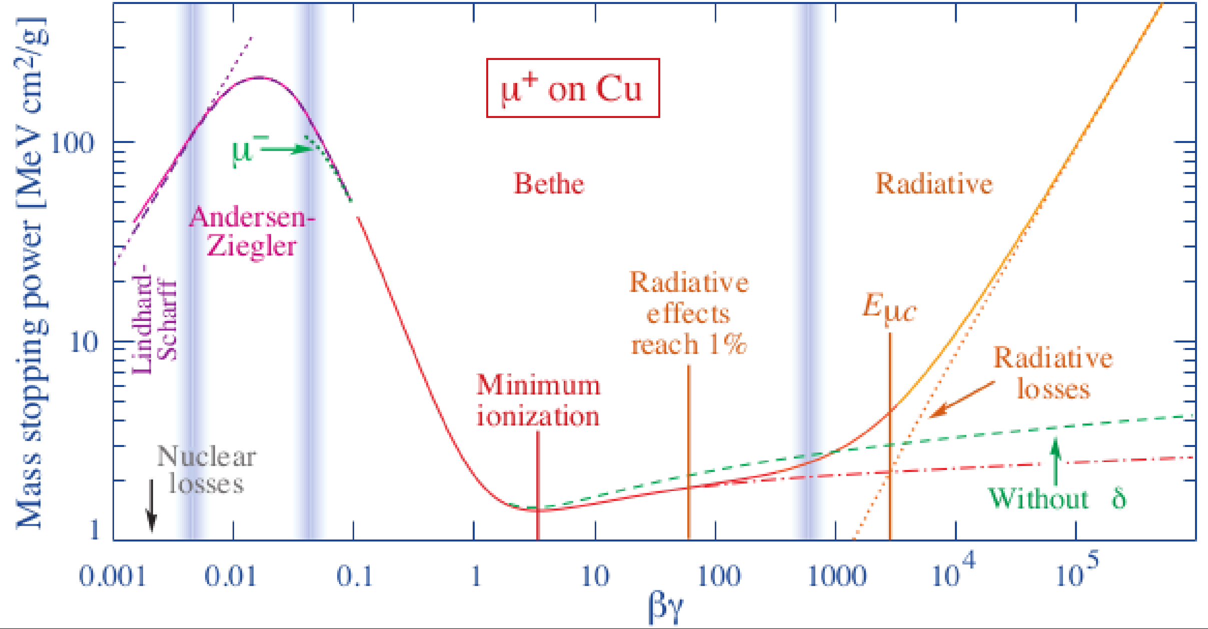
```
for (Int_t i = 0; i < 6; i++)
   {
   switch (generator)
   {
   case EGenerators::BOX: // Box generator
   {
   //for (Int_t i = 0; i < 6; i++)
   //{
      gRandom->SetSeed(0);
      FairBoxGenerator *boxGen = new
FairBoxGenerator(partPdgC[i], 100);
      //FairBoxGenerator *boxGen = new
FairBoxGenerator(partPdgC[4], 100);
      //FairBoxGenerator *boxGen = new FairBoxGenerator(13,
100); // 13 = muon; 1 = multipl.
      boxGen->SetPRange(0.0,
5.0);                          // GeV/c, setPRange vs
setPtRange
      boxGen->SetPhiRange(0,
360);                          // Azimuth angle range
[degree]
      boxGen->SetThetaRange(0,
180);                         // Polar angle in lab system
range [degree]
      boxGen->SetXYZ(0., 0.,
0.);                           // mm o cm ??
      primGen->AddGenerator(boxGen);
      break;
     //}
   }
```

2

```
   Int_t nhits  = track->GetNofHits();

   if (TMath::Abs(ptmc) < 0.1) continue;
   if (TMath::Abs(etamc) > 1.3) continue;
   if (nhits < 16) continue;

   float dEdx = track->GetdEdXTPC();
   //mhdEdx->Fill(TMath::Abs(ptmc)*TMath::CosH(etamc),
Edx);
   mhdEdx->Fill(P.Mag(), dEdx);

   //if (pdg == 211 || pdg == 321 || pdg == 2212 || pdg ==
1 || pdg == 13 )
   if (pdg == 1000020030)
   {
       //He3
       mhdEdxHe3->Fill(TMath::Abs(ptmc)*TMath::CosH(etamc),
Edx);

   }else if (pdg == 1000010020)
   {
       //Deuterions
       mhdEdxd->Fill(TMath::Abs(ptmc)*TMath::CosH(etamc),
Edx);
   }else if (pdg == 321)
   {
       //Kaons
       mhdEdxK->Fill(TMath::Abs(ptmc)*TMath::CosH(etamc),
Edx);
   }else if (pdg == 2212)
   {
```

WE OBTAIN:





3.-PARAMETRIZE DE/DX BANDS FOR PARTICLE SPECIE

3.1 FIT FUNCTIONS

3.2 CUTS IN THE DE/DX DISTRIBUTIONS

3.3 GAUSSIAN FIT FOR CUTTING

3.4 DEGREES OF FREEDOM AND PARAMETERS

3.5 FIT FUNCTION FOR EACH DE/DX

FIT FUNCTION

For the region of minimum ionization, which is the one we are working on, the function that fits said region will be the "Bethe equation" and is given by:

$$\left\langle -\frac{dE}{dx}\right\rangle = K z^2 \frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}\ln\frac{2m_e c^2\beta^2\gamma^2 W_{\max}}{I^2} - \beta^2 - \frac{\delta(\beta\gamma)}{2}\right]$$

K — $4\pi N_A r_e^2 m_e c^2$ (Coefficient for dE/dx)
z — charge number of incident particle
Z — atomic number of absorber
A — atomic mass of absorber
I — mean excitation energy
$W_{\max}$ — Maximum possible energy transfer
$m_e c^2$ — electron mass × c^2
$\delta(\beta\gamma)$ — density effect correction to ionization energy loss
β — Relative velocity divided by the speed of light in vacuum
γ — Lorentz factor

But for our data we will have the option of using two fitting functions that satisfy the Bethe equation, these functions will be defined in two different macros.

In MpdPairKK.cxx

1

$$f(x) = \frac{p_0}{x^2}\times\left(p_1\log(x^2) - p_2 x^2 - p_3 x - p_4 - p_5 x^3\right) + p_6$$

```
float MpdPairKK::dEdx_sigma_P(float dEdx, float mom) const
{
   if (mom < 0.08) return -999;
   if (dEdx <= 0) return -999;

   dEdx = log(dEdx);

   if (mom < 0.09) mom = 0.09;
   if (mom > 3.5) mom = 3.5;

   float mean[7]  = {1.183672e-001, -3.394823e-001, 2.418100e+001, -1.377881e+001,
                     2.534160e+000, -1.563054e+000, 2.010767e+000};
   float width[7] = {-1.488536e+007, -2.075514e-010, 2.418077e-009, -2.996053e-009,
                      1.397806e-009,  -4.161277e-010, 7.174217e-002};

   float mean_exp, width_exp;

   mean_exp =
      mean[0] / mom / mom *
         (mean[1] * log(mom * mom) - mean[2] * mom * mom - mean[3] * mom - mean[4] - mean[5] * mom * mom * mom) +
      mean[6];
   width_exp =
      width[0] / mom / mom *
         (width[1] * log(mom * mom) - width[2] * mom * mom - width[3] * mom - width[4] - width[5] * mom * mom * mom) +
      width[6];

   return (dEdx - mean_exp) / width_exp;
}
```

In MpdPid.cxx

2

$$\frac{dE}{dx}\Big/\frac{dE}{dx}(\min) = \frac{P_1}{\beta^{P_4}}\left\{P_2 - \beta^{P_4} - \ln\left[P_3 + \left(\frac{1}{\beta\gamma}\right)^{P_5}\right]\right\}$$

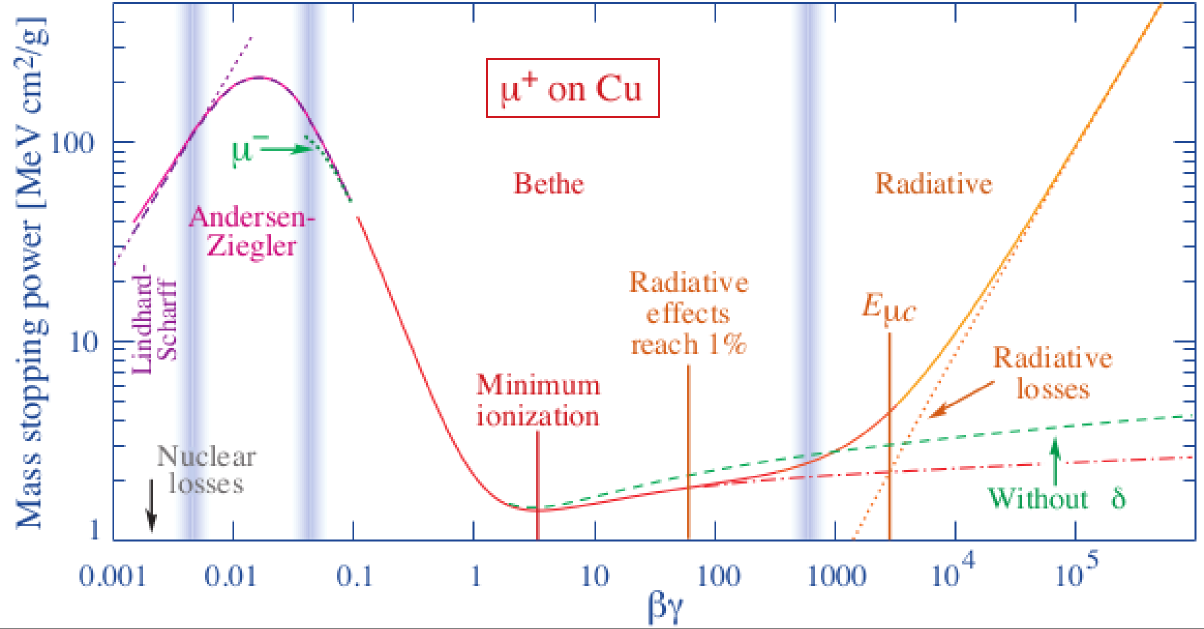
$$f = x_1(x_2 - x_3) \qquad x_3 = \ln\left[c + \left(\frac{M}{p}\right)^g\right]$$

$$x_1 = \frac{a}{\beta^d} \qquad \beta = \frac{p}{\sqrt{p^2 + M^2}}$$

$$x_2 = b - \beta^d$$

```
Double_t MpdPid::parPrBB(Double_t *x, Double_t *par)
{
   Double_t x1, x2, x3, p[5], xint = fTrackingState != MpdPidUtils::kCFHM ? 0.384089 : 0.34173458, ans;
   for (Int_t k = 0; k < 5; k++) p[k] = x[0] < xint ? par[k] : par[k + 5];
   x1  = p[0] / TMath::Power(x[0] / TMath::Sqrt(x[0] * x[0] + 0.88), p[3]);
   x2  = p[1] - TMath::Power(x[0] / TMath::Sqrt(x[0] * x[0] + 0.88), p[3]);
   x3  = TMath::Log(p[2] + TMath::Power(1.0 / (x[0] / 0.9383), p[4]));
   ans = x1 * (x2 - x3);
```

Both functions depend on 6 parameters. For our case we choose function 1.

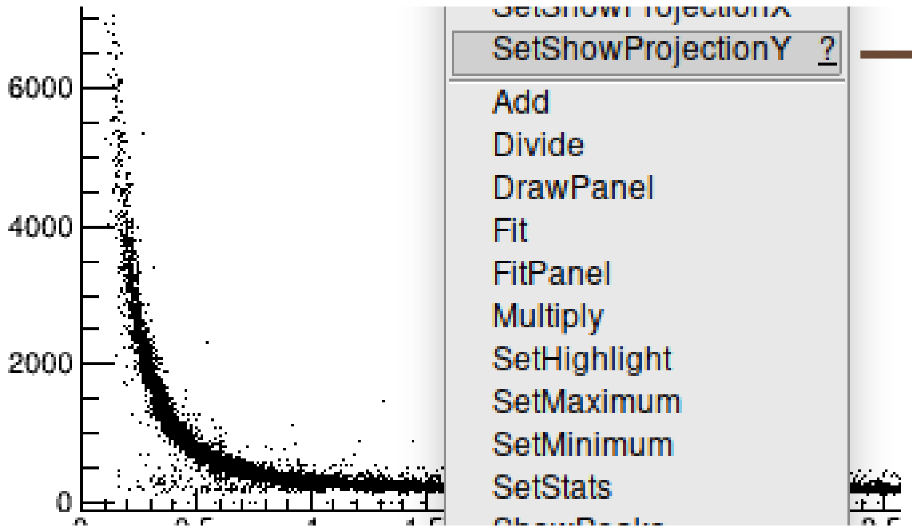


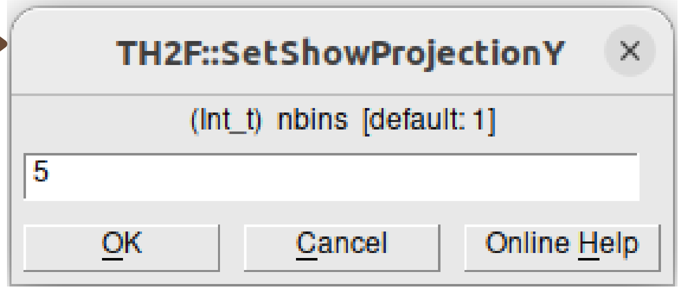
3.-PARAMETRIZE DE/DX BANDS FOR PARTICLE SPECIE

CUTS IN THE DE/DX DISTRIBUTIONS

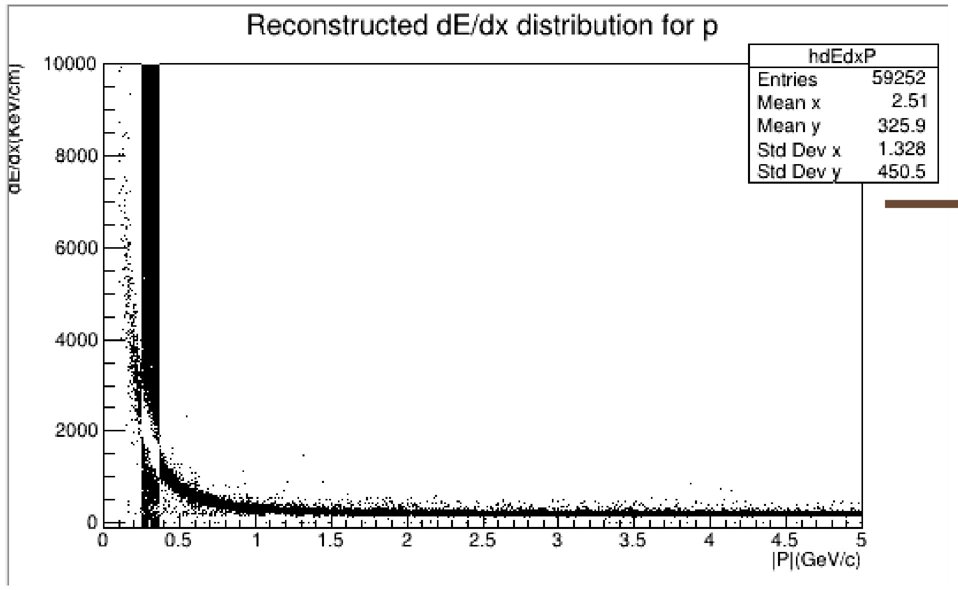
Because we have 6 parameters this implies that our degrees of freedom should be greater than 6, so to properly estimate the parameters we will take 21 degrees of freedom or cuts in the dE/dx distribution. These cuts will be made in the following way:

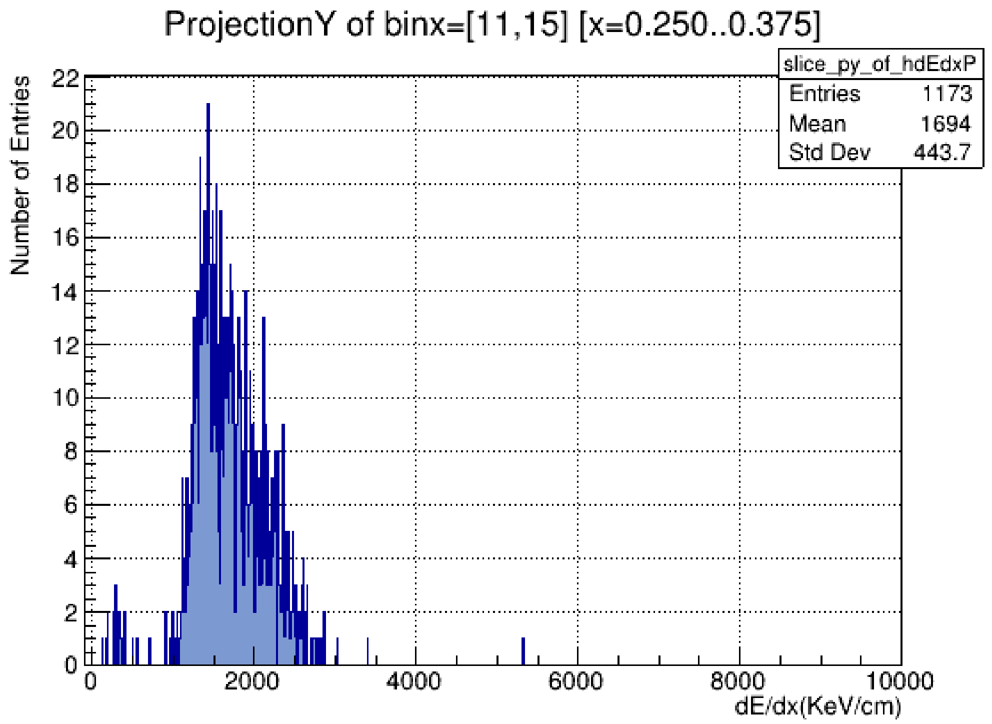
1 On the histogram to work on, select the setShowProjectionY option and specify a bin value

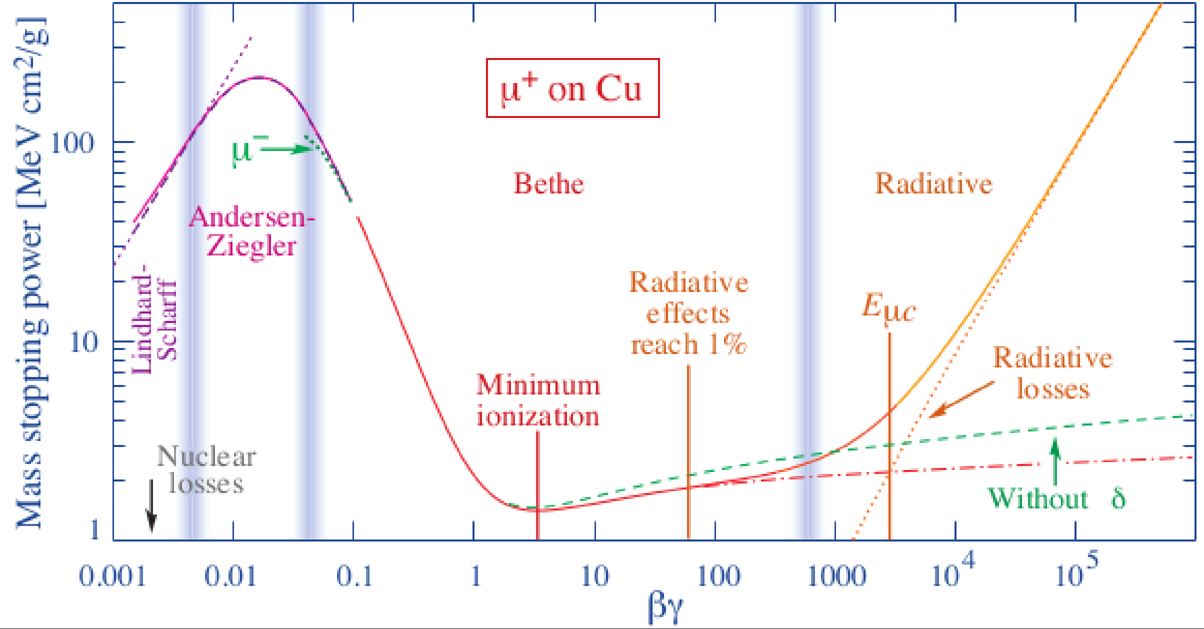




2 A second window will be displayed showing the distribution of the number of entries that belong to the selected cut.







3.-PARAMETRIZE DE/DX BANDS FOR PARTICLE SPECIE

3.1 FIT FUNCTIONS

3.2 CUTS IN THE DE/DX DISTRIBUTIONS

3.3 GAUSSIAN FIT FOR CUTTING

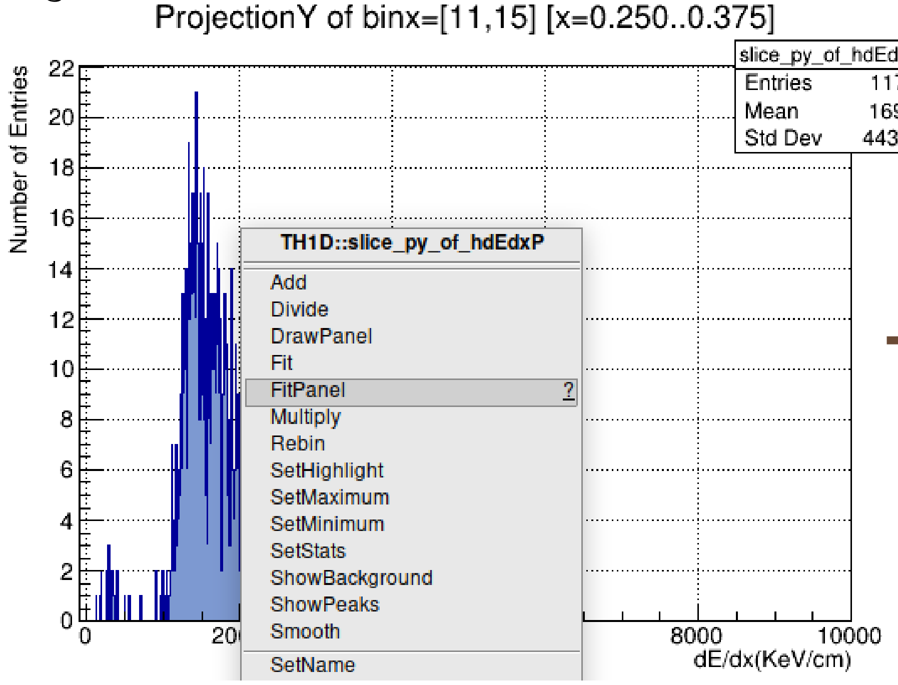
3.4 DEGREES OF FREEDOM AND PARAMETERS

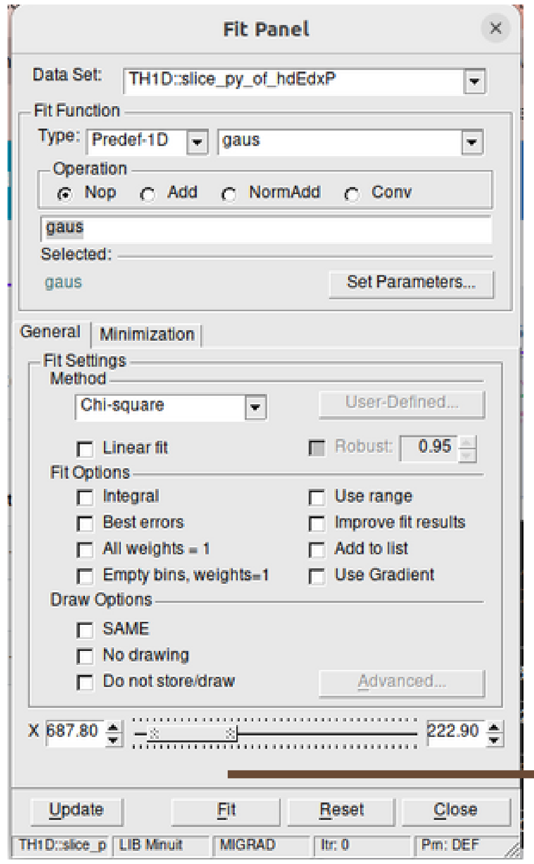
3.5 FIT FUNCTION FOR EACH DE/DX

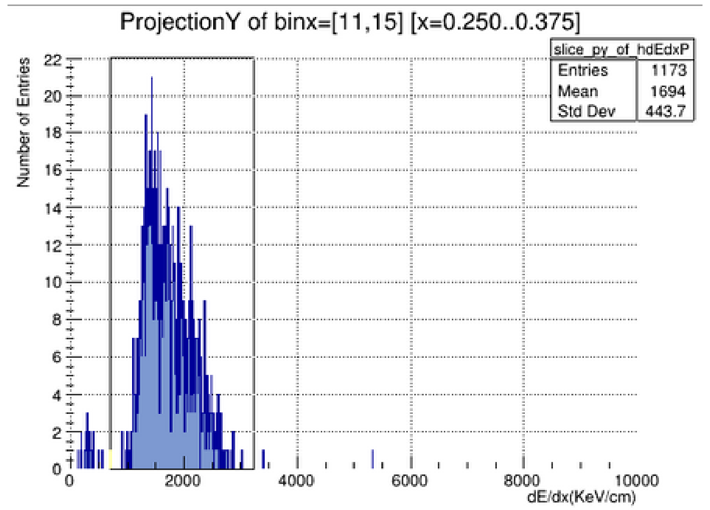
GAUSSIAN FIT FOR CUTTING

Now a Gaussian function must be fitted to the input distribution, it will be done as follows:

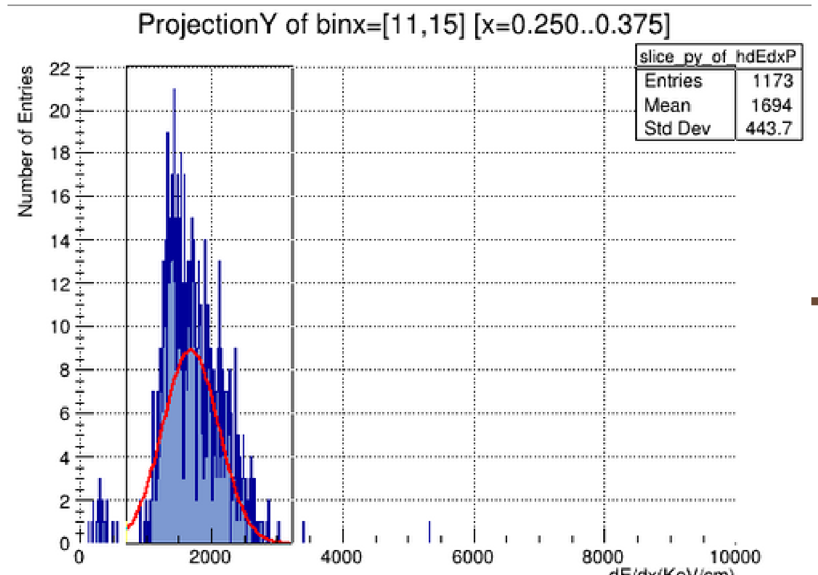
1 In the distribution select the FitPanel option and in the pop-up window select the "Predef-1D" function type, as well as the fit region.





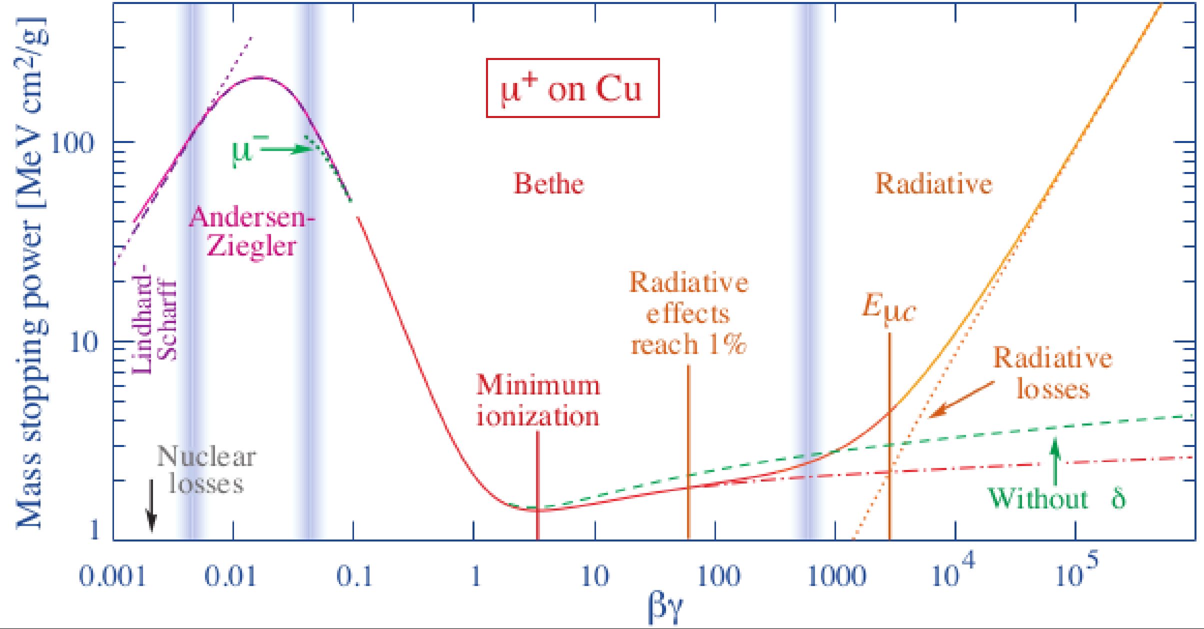


2 Once the options are specified, the "Fit" option is selected and we obtain:



The degrees of freedom (mean) and errors will appear in the terminal

```
EXT  PARAMETER
NO.    NAME        VALUE             ERROR
 1   Constant     8.93855e+00     4.12379e-
 2   Mean         1.67643e+03     1.71176e+
 3   Sigma        4.27806e+02     1.75406e+
```



3.-PARAMETRIZE DE/DX BANDS FOR PARTICLE SPECIE

3.1 FIT FUNCTIONS

3.2 CUTS IN THE DE/DX DISTRIBUTIONS

3.3 GAUSSIAN FIT FOR CUTTING

3.4 DEGREES OF FREEDOM AND PARAMETERS

3.5 FIT FUNCTION FOR EACH DE/DX

DEGREES OF FREEDOM AND PARAMETERS

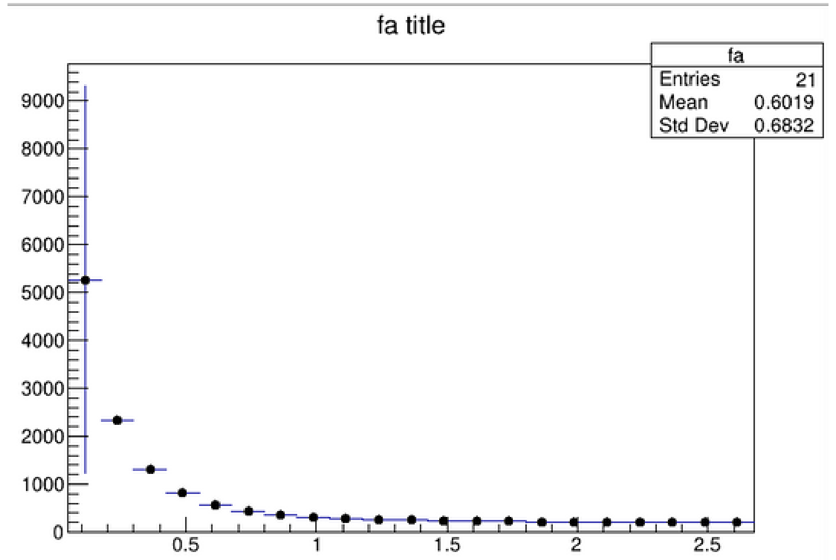
Once the 21 cuts have been made and, therefore, the 21 degrees of freedom with their respective errors have been obtained, a histogram will be defined that graphs said values.

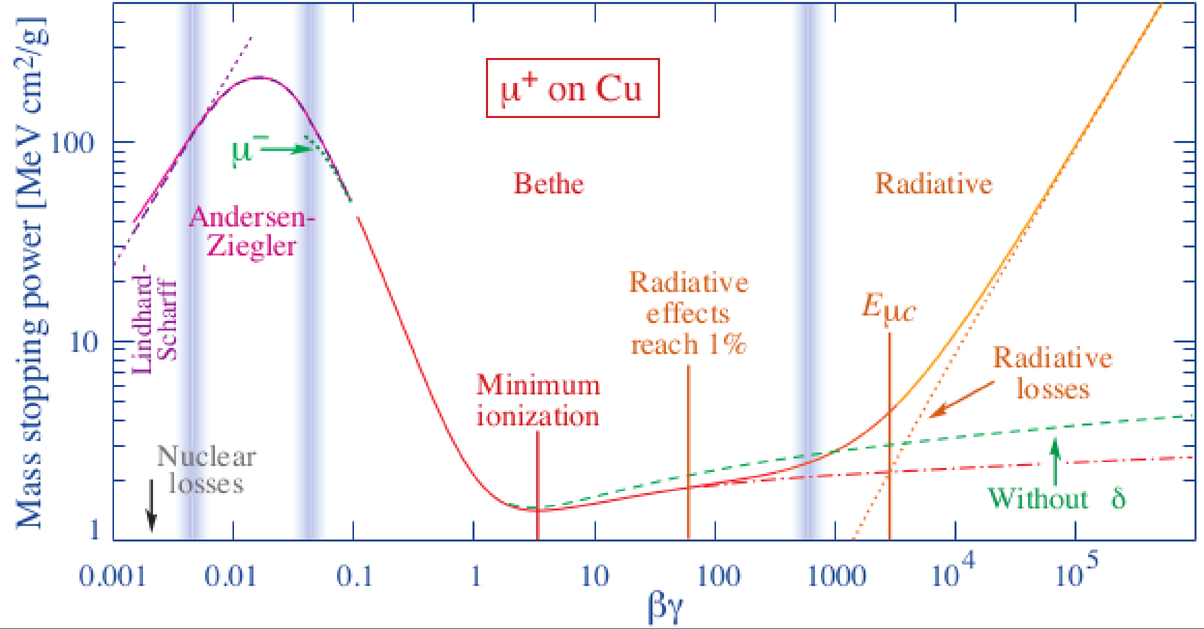
1

```
void plotProton(){

   TH1F *fa = new TH1F("fa", "fa title", 21, 0.050, 2.675);
   fa->SetBinContent(1,5.26459e+03);
   fa->SetBinContent(2,2.33194e+03);
   fa->SetBinContent(3,1.31347e+03);
   fa->SetBinContent(4,8.07440e+02);
   fa->SetBinContent(5,5.69834e+02);
   fa->SetBinContent(6,4.40979e+02);
   fa->SetBinContent(7,3.67455e+02);
   fa->SetBinContent(8,3.18473e+02);
   fa->SetBinContent(9,2.87055e+02);
   fa->SetBinContent(10,2.64130e+02);
   fa->SetBinContent(11,2.49205e+02);
   fa->SetBinContent(12,2.35646e+02);
   fa->SetBinContent(13,2.26963e+02);
   fa->SetBinContent(14,2.20440e+02);
   fa->SetBinContent(15,2.15692e+02);
   fa->SetBinContent(16,2.10908e+02);
   fa->SetBinContent(17,2.07808e+02);
   fa->SetBinContent(18,2.06383e+02);
   fa->SetBinContent(19,2.03644e+02);
   fa->SetBinContent(20,2.01427e+02);
   fa->SetBinContent(21,2.00220e+02);
```

```
   fa->SetBinError(1,4.05030e+03);
   fa->SetBinError(2,9.19109e+01);
   fa->SetBinError(3,9.94266e+00);
   fa->SetBinError(4,3.42809e+00);
   fa->SetBinError(5,1.83025e+00);
   fa->SetBinError(6,1.33770e+00);
   fa->SetBinError(7,9.42899e-01);
   fa->SetBinError(8,7.54512e-01);
   fa->SetBinError(9,6.50320e-01);
   fa->SetBinError(10,5.78024e-01);
   fa->SetBinError(11,5.70373e-01);
   fa->SetBinError(12,5.19311e-01);
   fa->SetBinError(13,5.18305e-01);
   fa->SetBinError(14,5.23691e-01);
   fa->SetBinError(15,4.87780e-01);
   fa->SetBinError(16,4.68820e-01);
   fa->SetBinError(17,4.54431e-01);
   fa->SetBinError(18,4.57337e-01);
   fa->SetBinError(19,4.71617e-01);
   fa->SetBinError(20,4.31029e-01);
   fa->SetBinError(21,4.59270e-01);

   fa->Draw();
   }
```

2 With the command root plotProton.C (name of the macro that contains the histogram), we obtain:

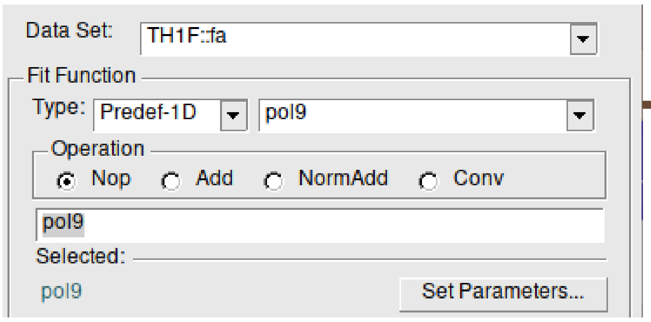


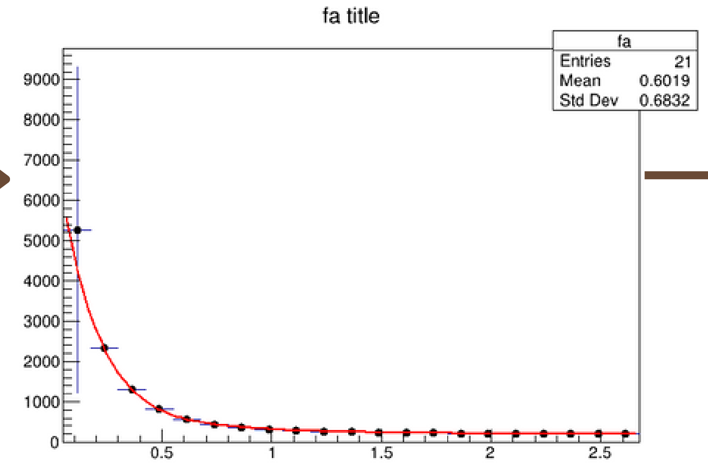


DEGREES OF FREEDOM AND PARAMETERS

3

- A function is manually adjusted (with the FItPanel option) according to the allowed options.
- In our case (within the FitPanel), the "pol9" option was selected, which refers to a 9th-degree polynomial.





We obtain the values of the parameters.

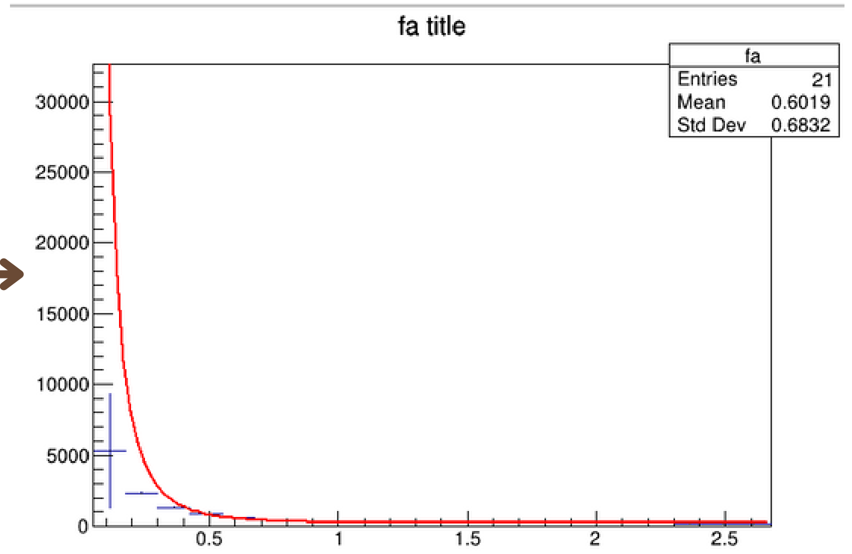
```
Minimizer is Linear / Migrad
Chi2                      =      15.7892
NDf                       =           11
p0                        =      7676.36   +/-   371.248
p1                        =       -39040   +/-   2952.39
p2                        =      96366.4   +/-   9913.83
p3                        =      -141337   +/-   18506.7
p4                        =       133083   +/-   21243.6
p5                        =     -82655.6   +/-     15608
p6                        =      33709.3   +/-   7365.77
p7                        =     -8686.79   +/-   2160.04
p8                        =      1282.56   +/-   358.235
p9                        =     -82.6626   +/-   25.6682
```

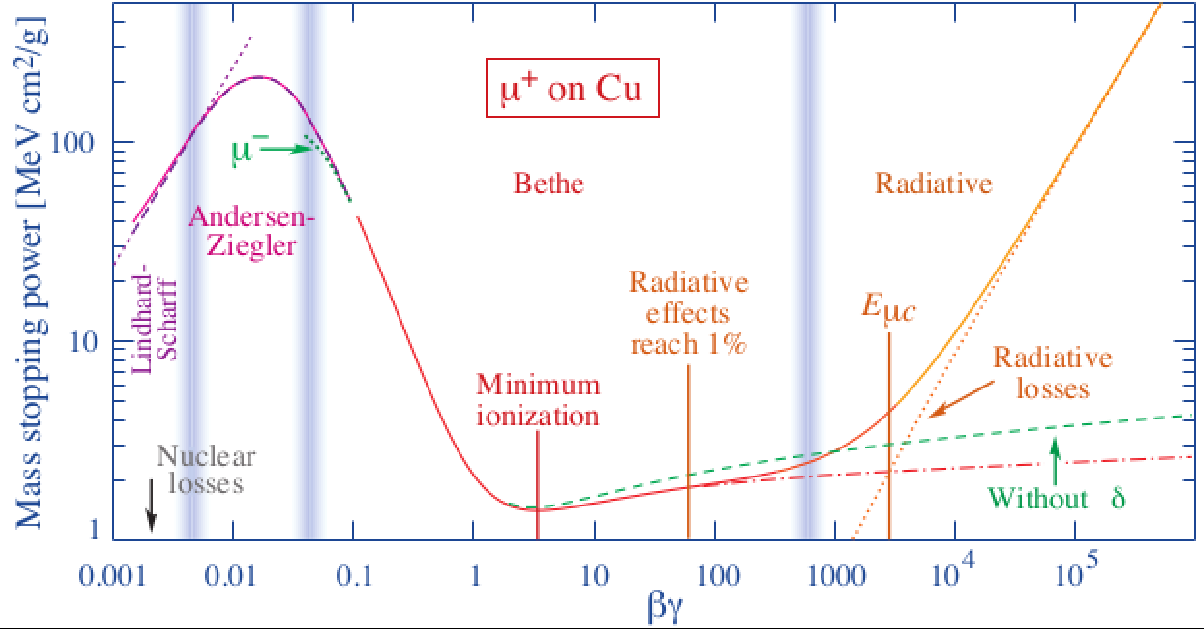
4

- We use only 7 parameters (P0 to P6) because that's what the function f(x) requires, which will allow us to achieve a better fit.
- The macro plotProton.C will be edited to add the parameters and the fitting function f(x).

$$f(x)=\frac{p_0}{x^2}\times\left(p_1\log(x^2)-p_2x^2-p_3x-p_4-p_5x^3\right)+p_6$$

```
TF1 *dEdx_mean = new TF1("dEdx_mean", " [0] / x / x * ([1] * log(x * x) - [2]
* x * x - [3] * x - [4] - [5] * x * x * x) + [6]", 0.050, 2.675);

  dEdx_mean->SetParameter(0,7676.36);
  dEdx_mean->SetParameter(1,-39040);
  dEdx_mean->SetParameter(2,96366.4);
  dEdx_mean->SetParameter(3,-141337);
  dEdx_mean->SetParameter(4,133083);
  dEdx_mean->SetParameter(5,-82655.6);
  dEdx_mean->SetParameter(6,33709.3);

  fa->Fit(dEdx_mean, "R");

  fa->Draw();
  dEdx_mean->Draw("same");
  }
```





3.-PARAMETRIZE DE/DX BANDS FOR PARTICLE SPECIE

3.1 FIT FUNCTIONS

3.2 CUTS IN THE DE/DX DISTRIBUTIONS

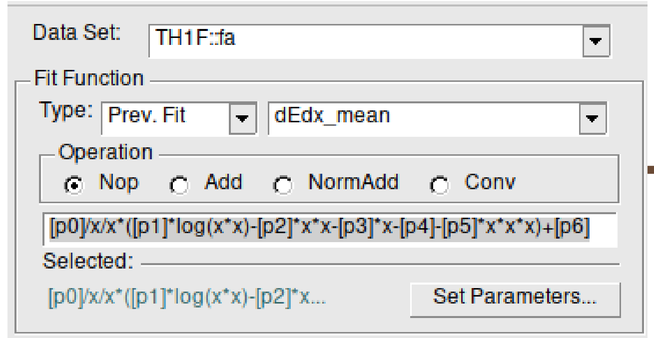
3.3 GAUSSIAN FIT FOR CUTTING

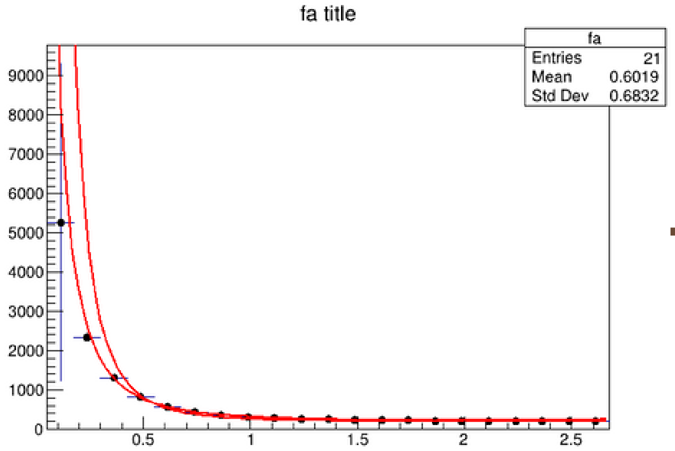
3.4 DEGREES OF FREEDOM AND PARAMETERS

3.5 FIT FUNCTION FOR EACH DE/DX

FIT FUNCTION FOR EACH DE/DX

5 The steps 3 and 4 will be repeated to obtain a better estimation of the parameters, but the option chosen will be the f(x) function introduced in the macro.



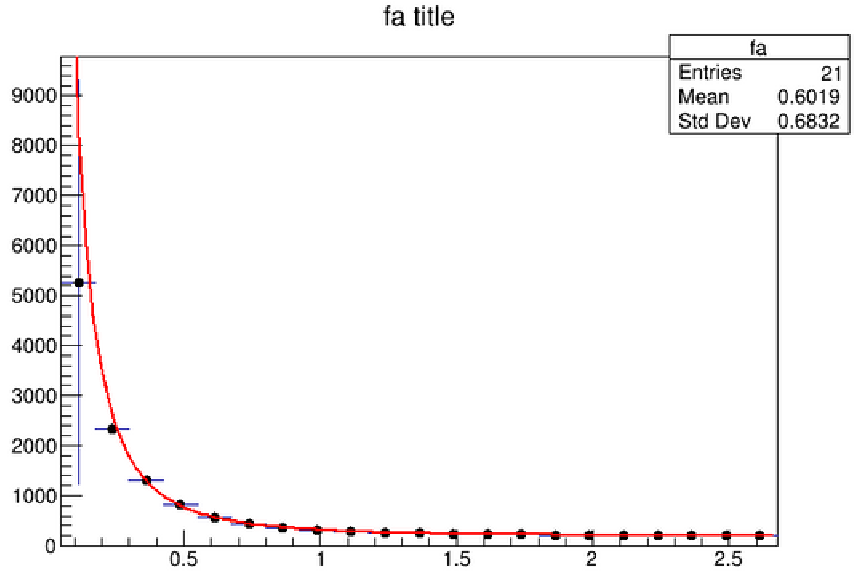


We obtain the final values for the parameters

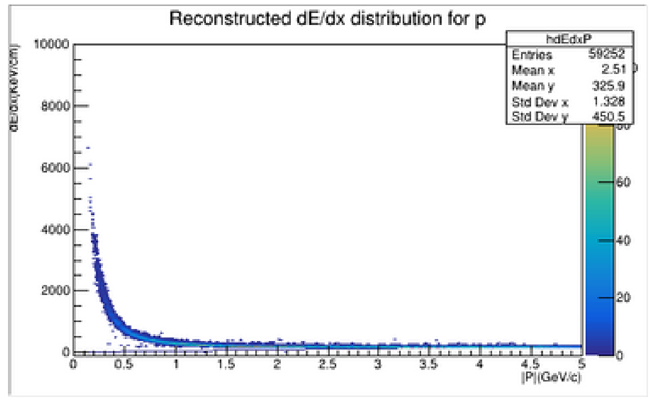
```
TFitEditor::DoFit - using function dEdx_mean  0x539dd00
 FCN=30.3397 FROM HESSE     STATUS=NOT POSDEF     50 CALLS         188 TOTAL
                     EDM=6.27277e-10    STRATEGY= 1      ERR MATRIX NOT POS-DEF
  EXT PARAMETER                APPROXIMATE        STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           1.89931e-04   3.98175e-06   2.51828e-10  -3.76397e+00
   2  p1           2.03523e+05   5.49913e+03   9.70479e-02  -1.43884e-08
   3  p2          -1.33570e+05   8.78392e+03   6.36911e-02   2.00450e-08
   4  p3           1.19415e+06   1.33680e+04   5.69417e-01   1.57012e-08
   5  p4          -1.57318e+06   1.52985e+04   6.83644e-01   9.33352e-09
   6  p5           3.47145e+04   2.77417e+03   1.65529e-02   2.38130e-08
   7  p6           2.24198e+02   1.66795e+00   7.28362e-05  -1.05567e-04
```

```
TF1 *dEdx_mean = new TF1("dEdx_mean", " [0] / x / x * ([1] * log(x * x) - [2]
* x * x - [3] * x - [4] - [5] * x * x * x) + [6]", 0.050, 2.675);

dEdx_mean->SetParameter(0,1.90080e-04);
dEdx_mean->SetParameter(1,2.03365e+05);
dEdx_mean->SetParameter(2,-1.33086e+05);
dEdx_mean->SetParameter(3,1.19323e+06);
dEdx_mean->SetParameter(4,-1.57195e+06);
dEdx_mean->SetParameter(5,3.46886e+04);
dEdx_mean->SetParameter(6,2.24271e+02);


fa->Fit(dEdx_mean, "R");

fa->Draw();
dEdx_mean->Draw("same");
}
```

Final adjustment for the dE/dx distribution, in this case for the proton.



The procedure will be the same for the distributions of the other particles.



RESULTS

According to the data obtained by the BOX generator, 6 particles belonging to light hadrons were identified. This identification was possible thanks to obtaining ionization energy loss histograms for each particle and their analysis, from which the parameter values necessary to perform the adjustment of the previously presented function were obtained, thus providing a better description of the trend that should follow the energy loss of each particle.