PID in the NICA experiment using MVA

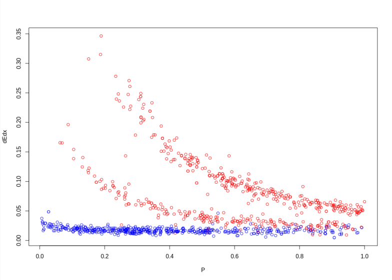
4th Computing/Analysis Workshop of the MexNICA Collaboration

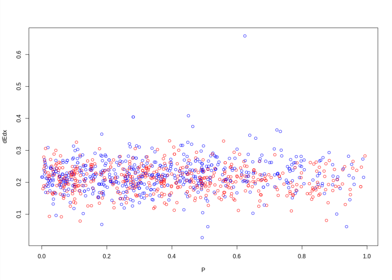
Julio César Maldonado González

June 15th, 2022

Bethe-Bloch for heavy-ions

Signal dE/dx vs p in the MPD-TPC detector for Bi-Bi collisions of $\sqrt{s} = 11$ GeV,





Simulation and reconstruction data

- Detector: MPD (TPC, TOF)
- Input file: rectestf140.root file (DST)
- Event generator: UrQMD
- Bi-Bi a 11 GeV (MB)
- of events: 10k
- Macro: CompareSpectra.C, anaDST.C

Read reconstruction files

We read the reconstruction files from the tree in the *.root* file with the information of events and tracks:

```
for (Int_t nf = 0; nf <1; nf++){      // files name rectestf14#.root, from nf to nf++

  // Input files
  sprintf(name,"/home/julio/Research/Taller3/12_PID-with-different-detectors/rectestf14%d.root",nf);
  cout <<" Openning file " << name << endl;
  TString fileName = name;

  // Define Chain to tree mpdsim
  TChain *mpdSim = new TChain("mpdsim");
  mpdSim->Add(fileName);   //put the filename

  // Set branches
  MpdEvent *mpdEvents = NULL;
  mpdSim->SetBranchAddress("MPDEvent.", &mpdEvents);
  TBranch *DSTBranch = mpdSim->GetBranch("MPDEvent.");

  TClonesArray *mcTracks = NULL;
  mpdSim->SetBranchAddress("MCTrack", &mcTracks);
  TBranch *MCBranch = mpdSim->GetBranch("MCTrack");
```

```
///////////////////////////////////////////////////////////////////////////
// MC tracks loop

Int_t mcNtracks = mcTracks->GetEntries();

//cout << " *** Event # " << i+1 << " No. of MC tracks = " << mcNtracks << endl;

for (Int_t j = 0; j < mcNtracks; j++){

  Ntr++;

  MpdMCTrack *mctrack = (MpdMCTrack*)mcTracks->At(j);

  MotherID = mctrack->GetMotherId();
  PDGID = mctrack->GetPdgCode();

  PtMC = mctrack->GetPt();
  PtMC = TMath::Abs(PtMC);

  Pxyz.SetXYZ(mctrack->GetPx(),mctrack->GetPy(),mctrack->GetPz());
  EtaMC = Pxyz.PseudoRapidity();

  NMC++;

  hEtaMC->Fill(EtaMC);
  hPtMC->Fill(PtMC);

} //end MC track loop
```

```cpp
//////////////////////////////////////////////////////////////////
// MPD tracks loop

TClonesArray *mpdTracks = mpdEvents->GetGlobalTracks();
Int_t mpdNtracks = mpdTracks->GetEntriesFast();

for (Int_t k = 0; k < mpdNtracks; k++){

  Ntrmpd++;

  MpdTrack *mpdtrack = (MpdTrack*)mpdTracks->UncheckedAt(k);

  ID = mpdtrack->GetID();

  MpdMCTrack *mctrack1 = (MpdMCTrack*)mcTracks->UncheckedAt(ID);

  MotherID = mctrack1->GetMotherId();

  Pt = mpdtrack->GetPt();
  Pt = TMath::Abs(Pt);

  Eta = mpdtrack->GetEta();

  N++;

  hEta->Fill(Eta);
  hPt->Fill(Pt);
```

```
    Px = mpdtrack->GetPx();
    Py = mpdtrack->GetPy();
    Pz = mpdtrack->GetPz();

    P = TMath::Sqrt(Pz*Pz + Px*Px + Py*Py);

    dEdx = mpdtrack->GetdEdXTPC();

    hEta->Fill(Eta);
    hPt->Fill(Pt);

    hdEdx->Fill(dEdx);

    hdEdxP->Fill(P,dEdx);

  } // MPD track loop

  hN->Fill(N);
```

Probability function

The probability of a particle i, if s signal is observed,

$$P(i|s) = \frac{r(s|i)C_i}{\sum_k r(s|k)C_k}$$

$r(s|i)$ probability density function of the observed signal s of the detector, if a particle $i(e, \mu, \pi, K, p, ...)$ is detected.

C_i frecuency of the observed particle.

Using the tracks information of the Monte Carlo (*PDGID*)

```
if (TMath::Abs(PDGID) == 211 ){  //pion

  Npi++;

  hpiEta->Fill(Eta);
  hpiPt->Fill(Pt);

  Px = mpdtrack->GetPx();
  Py = mpdtrack->GetPy();
  Pz = mpdtrack->GetPz();
  P = TMath::Sqrt(Pz*Pz + Px*Px + Py*Py);

  dEdx = mpdtrack->GetdEdXTPC();
  hpidEdx->Fill(dEdx);
  hpidEdxP->Fill(P,dEdx);

  hpiEtaMC->Fill(EtaMC);
  hpiPtMC->Fill(PtMC);

  M2 = mpdtrack->GetTofMass2();
  hpiM2->Fill(M2);

}
```

dE/dx and dE/dx vs P for pions

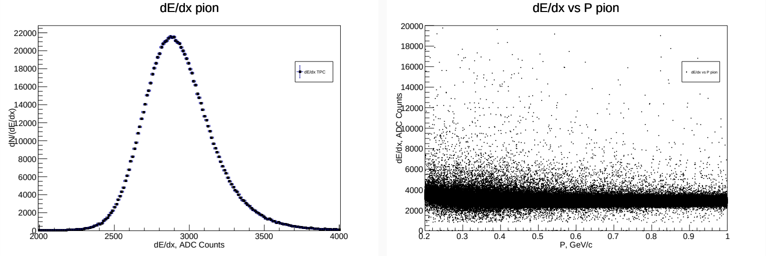
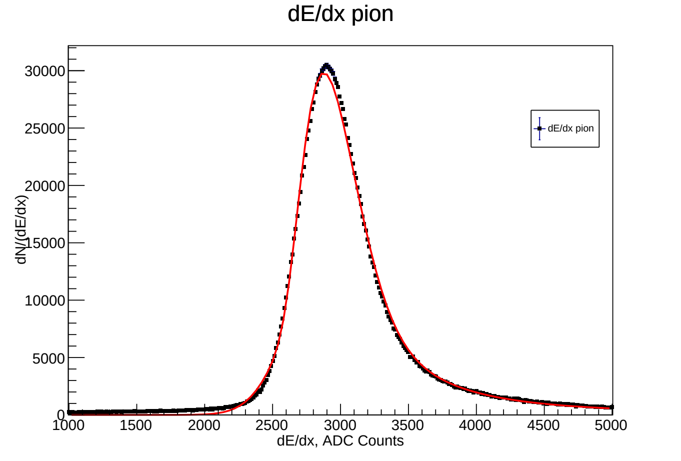


Figure 1: dE/dx en TPC para 10 mil eventos tomando partículas primarias y secundarias.

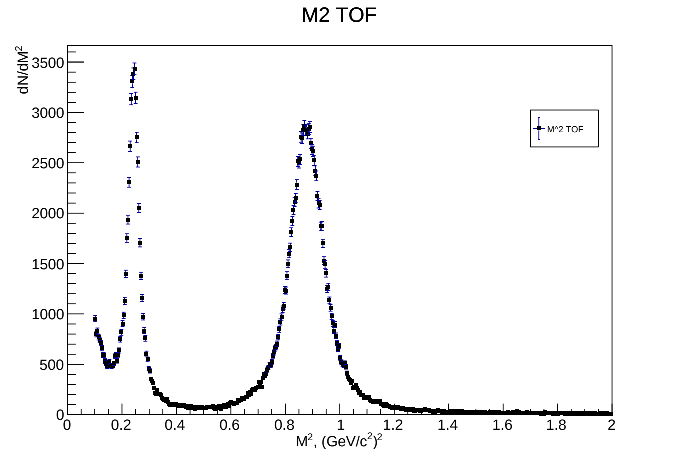
Ajuste al histograma dE/dx

```
//pion

Double_t parpi[6];

TF1* fitgausspi = new TF1("fitgausspi","gaus(0)",1000.,2900.00);
TF1* fitlandaupi = new TF1("fitlandaupi","landau(0)",2900.00,5000.);
TF1* fittotalpi = new TF1("fittotalpi","gaus(0)+landau(3)",1000.,5000.);

fittotalpi->SetLineColor(2);

hpidEdx->Fit(fitgausspi,"R");
hpidEdx->Fit(fitlandaupi,"R+");

fitgausspi->GetParameters(&parpi[0]);
fitlandaupi->GetParameters(&parpi[3]);

fittotalpi->SetParameters(parpi);

hpidEdx->Fit(fittotalpi,"R");
```

dE/dx histogram and fit



Frequency of the particle



Probability with bayesian model

```
Int_t Cp = 332;
Int_t Cpi = 52;
Int_t Ck = 1626;

Int_t i;

parp[0] = 2.26248e+02;
parp[1] = 3.53863e+03;
parp[2] = 5.27756e+02;
parp[3] = 4.56528e+03;
parp[4] = 5.09123e+03;
parp[5] = 4.68042e+02;

parpi[0] = 4.08121e+04;
parpi[1] = 3.40529e+03;
parpi[2] = -2.19923e+02;
parpi[3] = 1.12808e+06;
parpi[4] = 2.86475e+03;
parpi[5] = -2.12680e+02;

park[0] = 5.75678e+03;
park[1] = 4.07096e+03;
park[2] = 5.22213e+02;
park[3] = 9.05859e+03;
park[4] = 3.38738e+03;
park[5] = 2.72836e+02;
```

```
  TFormula *rp = new TFormula("rp","[0]*TMath::Exp(-0.5*TMath::Power((x-[1])/[2],2))
+ [3]*TMath::Exp(-0.5*TMath::Power((x-[4])/[5],2)) + [6]*TMath::Exp(-0.5*TMath::Power((x-[7])/[8],2))
+ [9]*TMath::Landau(x,[10],[11])");
  TFormula *rpi = new TFormula("rpi","[0]*TMath::Exp(-0.5*TMath::Power((x-[1])/[2],2))
+ [3]*TMath::Exp(-0.5*TMath::Power((x-[4])/[5],2)) + [6]*TMath::Exp(-0.5*TMath::Power((x-[7])/[8],2))
+ [9]*TMath::Landau(x,[10],[11])");
  TFormula *rk = new TFormula("rk","[0]*TMath::Exp(-0.5*TMath::Power((x-[1])/[2],2))
+ [3]*TMath::Exp(-0.5*TMath::Power((x-[4])/[5],2)) + [6]*TMath::Exp(-0.5*TMath::Power((x-[7])/[8],2))
+ [9]*TMath::Landau(x,[10],[11])");

  ////////////////////////////////////////////////////////////////////////////////////

  rp->SetParameters(parp);
  rp->Print();

  rpi->SetParameters(parpi);
  rpi->Print();

  rk->SetParameters(park);
  rk->Print();


  double_t crp, crpi, crk;
  double_t Probp, Probpi, Probk;

  for(i=0;i<15000;i++){

    fscanf(fp,"%lf",&dEdx[i]);

    crp = rp->Eval(dEdx[i]);
    crpi = rpi->Eval(dEdx[i]);
    crk = rk->Eval(dEdx[i]);

    Probp = (crp*Cp)/(crp*Cp + crpi*Cpi + crk*Ck);
    Probpi = (crpi*Cpi)/(crp*Cp + crpi*Cpi + crk*Ck);
    Probk = (crk*Ck)/(crp*Cp + crpi*Cpi + crk*Ck);

    fprintf(fp2, "%f %f %f \n", Probp, Probpi, Probk);

  }
```

MVA model

Representation of the type of the particle for a neural network output with three classes,

0	1	2	Particle
0	0	1	$K^{\pm}$
0	1	0	$\pi^{\pm}$
1	0	0	p

MLP model training

We take the reconstruction data for a selected momentum range, and choose the features of the model,

```
model <- mlp(
  x,
  y,
  size = 5,
  maxit = 100,
  initFunc = "Randomize_Weights",
  initFuncParams = c(-0.3, 0.3),
  learnFunc = "Std_Backpropagation",
  learnFuncParams = c(0.2, 0),
)
```

MLP model testing

We can predict the the probability by particle type

```
predict(model, newdata = z, type = "Prob")
```

ID	ProbP	ProbPi	ProbK
1	0.00263	0.01511	0.99023
2	0.25084	0.00266	0.69811
3	0.03034	0.09871	0.90590

Confusion matrix for the three classes model

We take a threshold value of probability (0.5) to compare with the probability of the prediction,

	VP %	FN %	VN %	FP %
p	96.22	3.78	98.18	1.82
$\pi^{\pm}$	90.38	9.62	94.89	5.11
$K^{\pm}$	85.72	14.28	95.15	4.85

Comparing models

For the dataset $0.2 \leq P < 1.0$

	Bayesiano	MLP
VP	83.84 %	99.18 %
FN	16.16 %	0.82 %
VN	95.08 %	99.52 %
FP	4.92 %	0.48 %

For the dataset $1.8 \leq P < 2.6$

	Bayesiano	MLP
VP	0.8 %	90.38 %
FN	99.2 %	9.62 %
VN	99.37 %	94.89 %
FP	0.63 %	5.11 %