JOINT INSTITUTE FOR NUCLEAR RESEARCH

Veksler and Baldin laboratory of High Energy Physics

FINAL REPORT ON THE START PROGRAM

Transverse momentum spectra of K_s^0 at mid-rapidity produced at 19.6 GeV Au-Au collisions

Supervisor:

Prof. Grigory Nigmatkulov

Student:

Aleksandra Marova, Russia

St Petersburg University

Participation period:

July 17 – September 03,

Summer Session 2022

Dubna, 2022

Contents

Abstract

As part of the work, experimental data from the Beam Energy Scan Program (BES) at the Relativistic Heavy Ion Collider (RHIC) (STAR experiment) were considered. The transverse momentum spectra of K_s^0 at mid-rapidity ($|y| < 0.5$) from Au+Au collisions at energy $\sqrt{s_{NN}} = 19.6$ GeV and at centralities 0-5% was fitting with Tsallis-Levy function, power-Law function and m_t-exponential function.

Chapter 1

Introduction

Within the framework of this project, the data from the Beam Energy Scan Program (BES) at the Relativistic Heavy Ion Collider (RHIC) (STAR experiment) for the production of strange hadrons, such as K_s^0 mesons in Au+Au collisions at $\sqrt{s_{NN}} = 19.6$ GeV and at centralities 0-5%, were considered [1].

The main purpose of scanning is to study the quantum chromodynamics (QCD) phase diagram. Systematic analysis of Au+Au collisions from $\sqrt{s_{NN}} = 39$ GeV down to 7.7 GeV at mid-rapidity ($|y| < 0.5$) in BES Program could help to achieve the following goals: 1) to find the QCD critical point where the first order phase transition at finite baryon chemical potential μ_B ends and to identify the phase boundary of the first order phase transition; 2) to locate the collision energy where deconfinement begins. For these goals, the study of strange hadrons is very well suited. The precise measurement of strange hadrons yields in heavy ion collisions in the BES may lead to a better understanding of strangeness production mechanisms in nuclear collisions and a more constrained extraction of the chemical freeze-out parameters. [1].

It was required for the specified range of centralities and for the specified energy to fit the transverse momentum spectra (the dependence of the $\frac{d^2N}{2\pi N_{evt} p_t dy dp_t}$ on the

transverse momentum). Several functions were chosen for fitting.

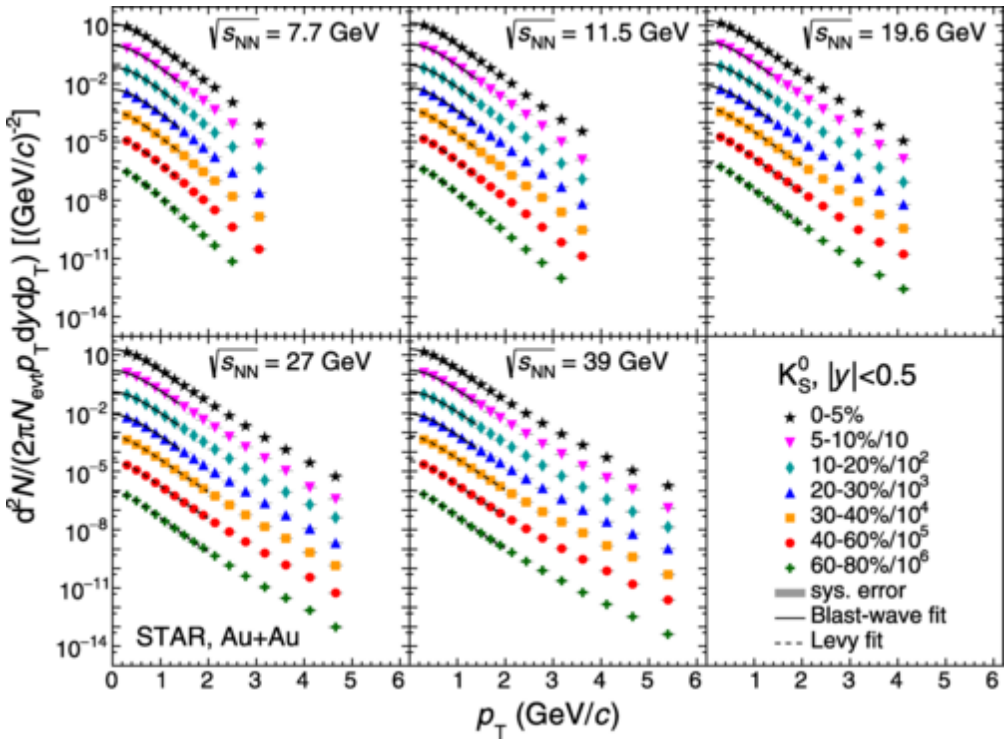

Figure 1.1: The transverse momentum spectra of K^0_s at mid-rapidity ($|y| < 0.5$) from Au+Au collisions at different centralities and energies ($\sqrt{s_{NN}} = 7.7 - 39$ GeV)

. The data points are scaled by factors of 10 from central to peripheral collisions for clarity. The vertical gray bands represent the systematic errors, which are small hence the bands look like horizontal bars. The blast-wave model (or Levy function) fit results are shown in the fit range and the low p_T extrapolation range as solid (dashed) lines for the centrality bins within 0–30% (30–80%) [1].

Functions selected for fitting:

1. The (two-component) Tsallis-Levy function [2].:

$$f_L = Cp_T(1 + \frac{\sqrt{p_T^2 + m_0^2} - m_0}{nT})^{-n}, \tag{1.1}$$

where C - normalization; p_T - transverse momentum; m_0 - mass; n - parameter for

fit; T - temperature.

2. Power-Law function [3]:

$$f = B(1 + \frac{p_T}{p_0})^{-n}, \tag{1.2}$$

where B - normalization; p_T - transverse momentum; p_0 and n - parameters for fit.

3. m_t-exponential function [3]:

$$f = Aexp(\frac{-m_T}{T}), \quad where \quad m_T = \sqrt{p_T^2 + m^2} \tag{1.3}$$

where A - normalization; m_T - transverse mass; p_T - transverse momentum; m - particle mass; T - temperature.

The following parameters are taken for fitting:
K_s^0 mesons mass: 0.4976 GeV/c^2 (fixed parameter);
The temperature of the chemical freeze-out is taken as the temperature value: 0.15-0.17 GeV (fixed in this range).
The rest of the fitting parameters must be defined.

Chapter 2

Project goals

The main goal of the project was to consider the transverse momentum spectra of K_s^0 at mid-rapidity ($|y| < 0.5$) from Au+Au collisions at energy $\sqrt{s_{NN}} = 19.6$ GeV and at centralities 0-5%. It is required to fit the experimental data with a certain function.

To complete this task, at the first stage of work, it was necessary to gain skills in working in the ROOT package, as well as complete a number of training tasks in C++.

Chapter 3

Learning tasks in ROOT

3.0.1 Task 1

Task: *Write a script that creates a TCanvas with the title "Functions", creates and draws two functions -sin(x)*x and cos(x)*x in the x range from -10 to 10. Draw the first function in black, the second is red. Save the drawing in .png format in the normal form and in the logarithmic scale along the y-axis.*
First solution method: plotting through TF1 and plotting a legend with TLegend.

```
void task1R_2() {
  Double_t x;
  TF1* f1 =new TF1("f1", "-sin(x)*x", -10, 10);
  TF1* f2 =new TF1("f2", "cos(x)*x", -10, 10);
  TCanvas *c1 = new TCanvas("c1","Functions",/*200,10,600,400*/1);
  f1->Draw();
  f2->Draw("same");
  c1->SetGrid();
  f1->GetYaxis()->SetRangeUser(-10, 10);
  f1->GetXaxis()->SetTitle("X");
  f1->GetYaxis()->SetTitle("Y");
  f1->SetLineColor(1);
  f2->SetLineColor(2);
  c1->BuildLegend();
  f1->SetTitle("");
```

```
  c1->Update();
  c1->SaveAs("graph1_1.png");
  TCanvas *c2 = new TCanvas("c2","Functions in log",1);
  f1->Draw();
  f2->Draw("same");
  c2->SetGrid();
  f1->SetTitle("-sin(x)*x");
  f1->GetYaxis()->SetRangeUser(-10, 10);
  f1->GetXaxis()->SetTitle("X");
  f1->GetYaxis()->SetTitle("log(Y)");
  f1->SetLineColor(1);
  f2->SetLineColor(2);
  c2->BuildLegend();
  c2->SetLogy();
  f1->SetTitle("");
  c1->Update();
  c2->SaveAs("graph1_2.png");
}
```

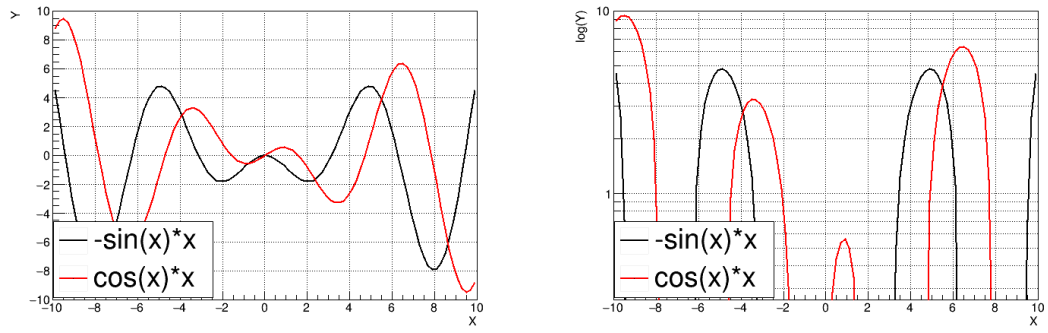

Figure 3.1: Task 1. Graphs in the normal form and in the logarithmic scale along the y-axis.

Second solution method: plotting by points (via TGraph by merging parts of graphs in TMultiGraph) and building a legend using TLegend.

```
void task1R() {
  Int_t n = 20, m = 20;
  Double_t  x1[n], x2[n],t1[m], t2[m], y1[n], y2[n], z1[m], z2[m];
```

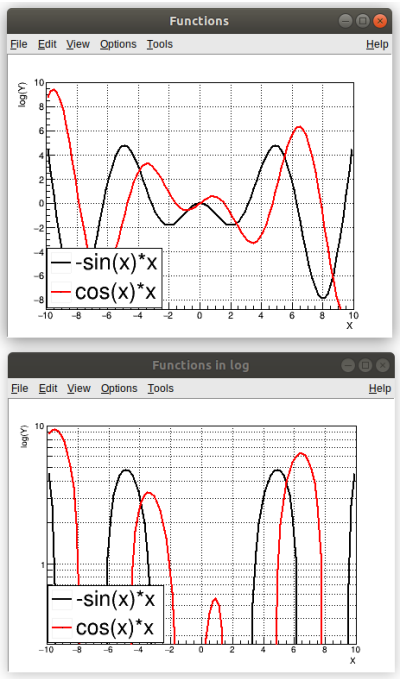

Figure 3.2: Task 1. TCanvas with functions.

```
for (Int_t i=0; i<n; i++) {
  x1[i] = -i*0.5;
  y1[i] = -sin(x1[i])*x1[i];
  x2[i] = -i*0.5;
  y2[i] = cos(x2[i])*x2[i];}
for (Int_t i=0; i<m; i++) {
  t1[i] = i*0.5;
  z1[i] = -sin(t1[i])*t1[i];
  t2[i] = i*0.5;
  z2[i] = cos(t2[i])*t2[i];}
TCanvas *c1 = new TCanvas("c1","Graphs",/*200,10,600,400*/1);
TGraph *gr1 = new TGraph(n,x1,y1);
TGraph *gr2 = new TGraph(n,x2,y2);
TGraph *gr3 = new TGraph(n,t1,z1);
TGraph *gr4 = new TGraph(n,t2,z2);
TMultiGraph *mg = new TMultiGraph();
mg->Add(gr1,"cp");
mg->Add(gr2,"cp");
```

```
  mg->Add(gr3,"cp");
  mg->Add(gr4,"cp");
  mg->Draw("a");
  c1->SetGrid();
  mg->GetXaxis()->SetTitle("X");
  mg->GetYaxis()->SetTitle("Y");
  gr1->SetLineColor(1);
  gr1->SetLineWidth(2);
  gr1->SetMarkerStyle(8);
  gr2->SetLineWidth(2);
  gr2->SetMarkerStyle(4);
  gr2->SetLineColor(2);
  gr3->SetLineWidth(2);
  gr3->SetMarkerStyle(8);
  gr4->SetLineWidth(2);
  gr4->SetMarkerStyle(4);
  gr4->SetLineColor(2);
  gr1->SetName("gr1");
  gr2->SetName("gr2");
  TLegend* leg = new TLegend(0.9,0.8,0.7,0.9);
  leg->AddEntry("gr1","-sin(x)*x","lp");
  leg->AddEntry("gr2","cos(x)*x","lp");
  leg->Draw();
  c1->Update();
  c1->SaveAs("graph1.png");
}
```

3.0.2 Task 2

Task: *Write a script that should do the following:*
a. Create TCanvas
*b. Fill in the values for x and y with an array of n = 20 points, where x[i] =i*0.1, and y[i]=10*sin(x[i]+0.2)*

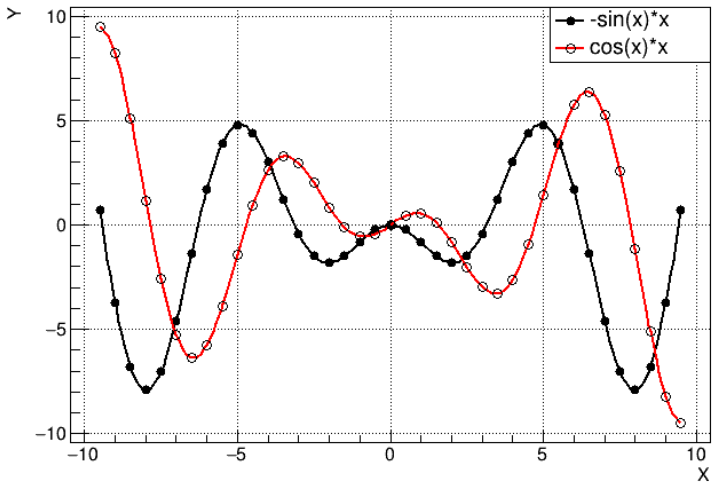

Figure 3.3: Task 1. Point Graphing (via TGraph by merging parts of graphs in TMultiGraph).

c. Set Plot Style: Empty Stars

d. Draw and save the constructed graph in .png format

Solution:

```
void task2R() {
  Int_t n = 20;
  Double_t  x[n], y[n];
  for (Int_t i=0; i<n; i++) {
    x[i] = i*0.1;
    y[i] = 10*sin(x[i]+0.2);}
  TCanvas *c1 = new TCanvas("c1","Graph",1);
  TGraph *gr1 = new TGraph(n,x,y);
  gr1->GetXaxis()->SetTitle("X");
  gr1->GetYaxis()->SetTitle("Y");
  gr1->SetLineColor(1);
  gr1->SetLineWidth(1);
  gr1->SetMarkerStyle(30);
  gr1->SetMarkerSize(1.4);
  gr1->Draw("ACP");
  c1->SetGrid();
```

```
  TLegend* leg = new TLegend(0.9,0.2,0.7,0.1);
  leg->AddEntry(gr1,"10*sin(x+0.2)","lp");
  leg->Draw();
  //c1->Update();
  c1->SaveAs("graph2.png");
}
```

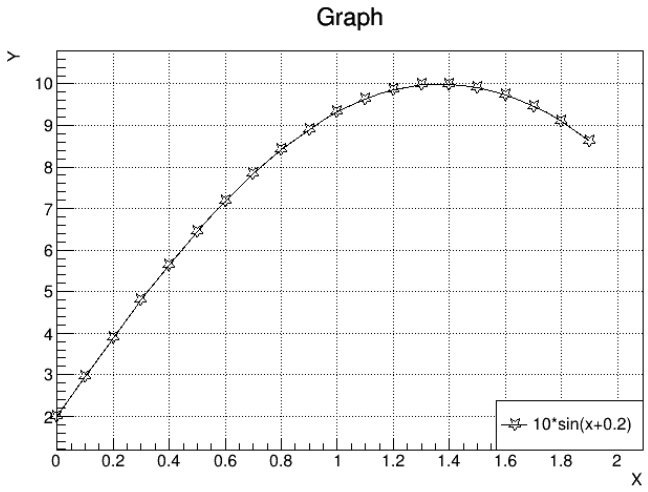

Figure 3.4: Task 2. Function graph.

3.0.3 Task 3

Task: *Build a histogram filled with the value of the function f(x) = exp(x)+100*gaus(x, sigma). Where sigma = 2. The number x is a random number distributed evenly in the range (-20, 20). Set the number of steps (points) for generation yourself.*
Solution:

```
void task3R_23() {
  TCanvas *c = new TCanvas("c", "c", 1);
  TH1F* h1 = new TH1F("h1", "distribution", 100, -20, 20); //creating hist. 100 bins
  TF1 *func1 = new TF1("funct1", "exp([0]+[1]*x)",-20,20); //creating func.
  func1->SetParameters(10,-0.1);
  TF1 *gauss = new TF1("gauss", "gaus",-20,20); //creating Gauss
  gauss->SetParameters(1.,-2.,1.6);
  TH1F* h2 = new TH1F("h2", "distribution", 100, -20, 20); //creating hist. 100 bins
```

```
 TH1F* h3 = new TH1F("h3", "distribution", 100, -20, 20); //creating hist. 100 bins
 h1->FillRandom("funct1",10E6); //fill hist. by funct1
 h2->FillRandom("gauss",10E5); //fill hist. by Gauss
 h1->Add(h2);
 h1->Draw();
 h1->GetXaxis()->SetTitle("X");
 h1->GetYaxis()->SetTitle("Y");
}
```

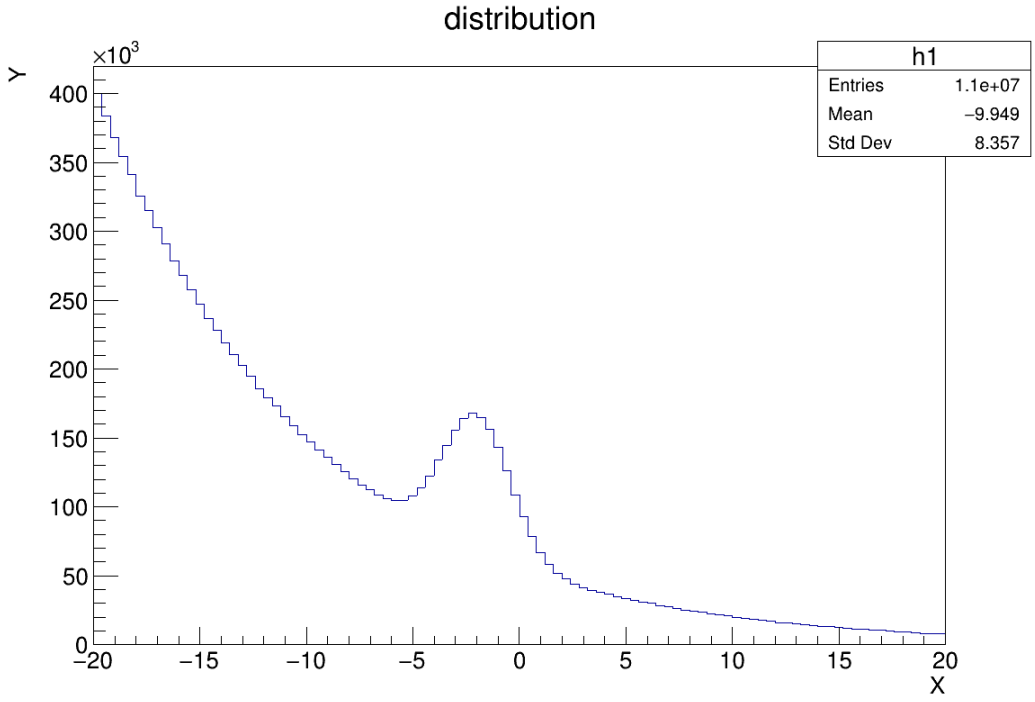

Figure 3.5: Task 3. Exponential "background" and Gaussian "signal".

3.0.4 Task 4

Task: *The constructed graph of the function from Task 3 should be fitted with the same function. The fit function parameters, Pearson's χ^2 test, and the number of degrees of freedom should be displayed.*

Solution (by Fit Panel): We have a function that is described by a Gaussian and an exponent. Gauss is described by three parameters: normalization, mean

and standard deviation. And the exponent is described by two parameters (the coefficient at the exponent and offset). Thus, we have a five-parameter function that we can set in the fitting panel after plotting the function. In the parameters, we set the same values that were used to build the function in the previous task. As a result, you can see (Fig. 8) that only the normalization has changed.

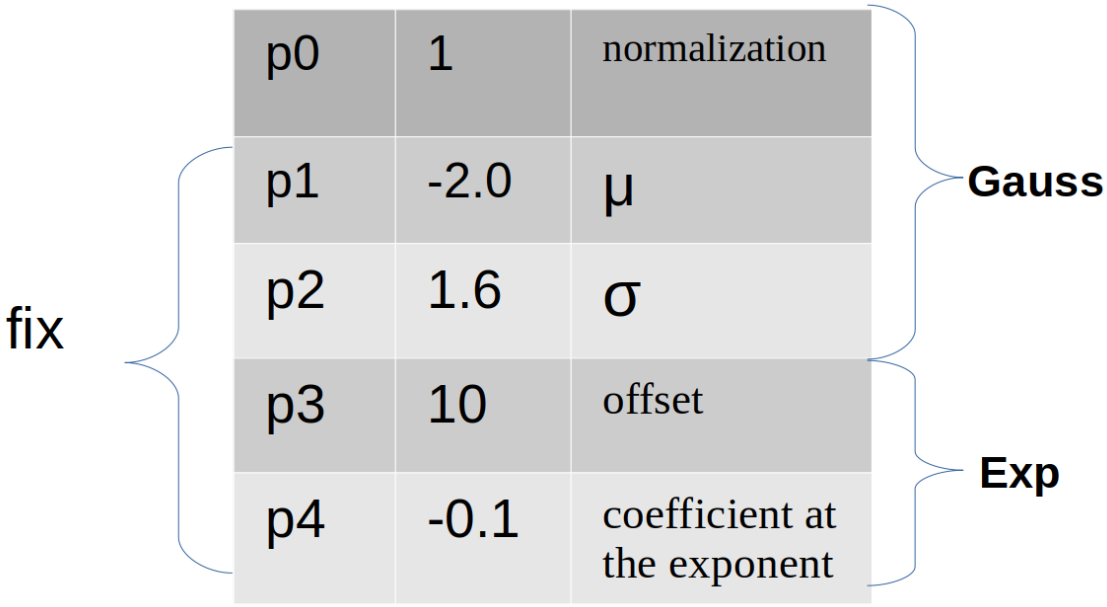

Figure 3.6: A five-parameter function for fit.

Solution (by macro for fitting): It would necessary to write a macro that would automatically fit this function and draw a Gaussian ("signal").

```
void task3R_24() {
  TCanvas *c = new TCanvas("c", "c", 1);
  TH1F *noise = new TH1F("noise", "distribution", 100, -20, 20); //creating hist. 100 bins
  TF1 *func1 = new TF1("funct1", "exp([0]+[1]*x)",-20,20); //creating func.
  func1->SetParameters(10,-0.1);
  TF1 *gauss = new TF1("gauss", "gaus",-20,20); //creating Gauss
  gauss->SetParameters(1.,-2.,1.6);
  // Functions for fitting
  TF1 *int1 = new TF1("int1", "expo",-20,-4);
  TF1 *int2 = new TF1("int2", "gaus",-4.5,1);
```

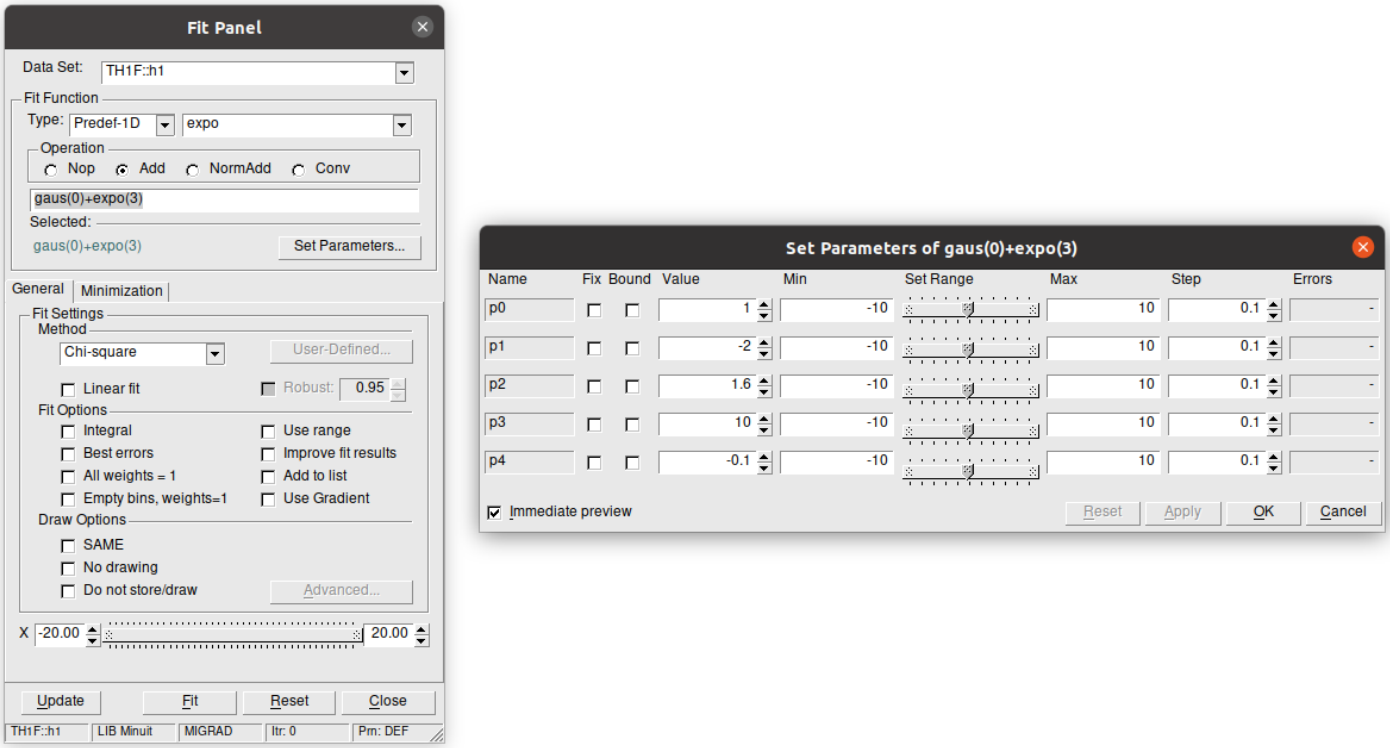

Figure 3.7: Task 4. Setting parameters in Fit Panel.

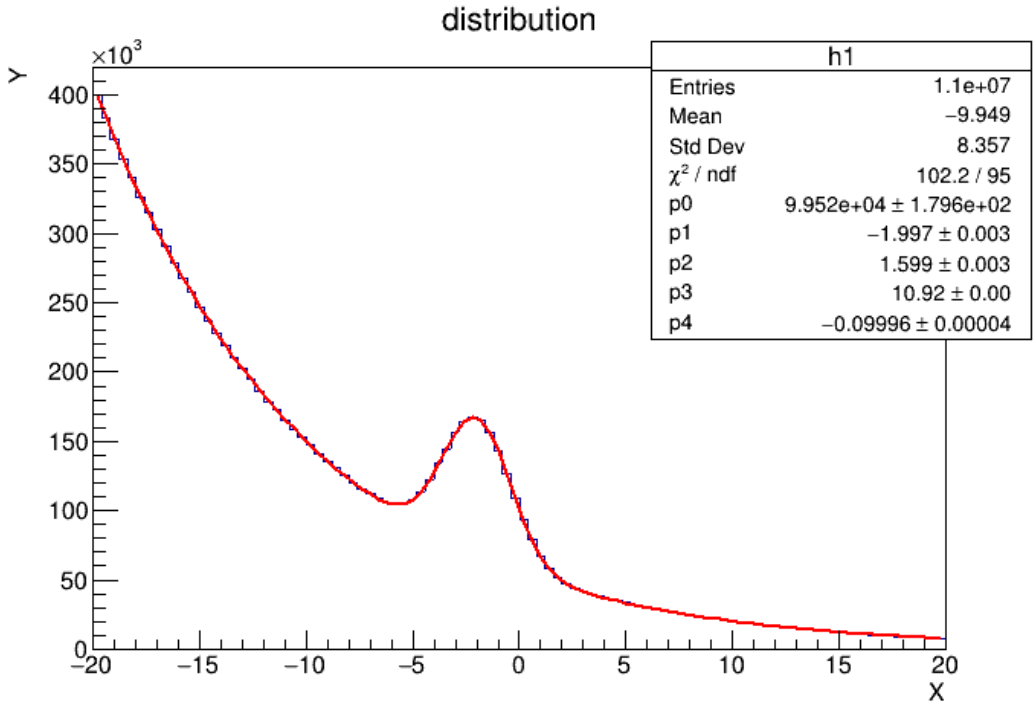

Figure 3.8: Task 4. Fitting a graph with five parametric function.

```
  TF1 *int3 = new TF1("int3", "expo",1.5,20);
  TF1 *inttotal = new TF1("inttotal", "expo(0)+gaus(2)+expo(5)",-20,20);
  inttotal->SetParameter(0,10);
  inttotal->FixParameter(1,-0.1);
  inttotal->SetParameter(2,1.);
  inttotal->SetParameter(3,-2.);
  inttotal->SetParameter(4,1.6);
  inttotal->SetParameter(5,10);
  inttotal->FixParameter(6,-0.1);
  TH1F* sign = new TH1F("sign", "distribution", 100, -20, 20); //creating hist. 100 bins
  TH1F* noiseandsign = new TH1F("noiseandsign", "distribution", 100, -20, 20);//creating hist.
  noise->FillRandom("funct1",10E6); //fill hist. by funct1
  sign->FillRandom("gauss",10E5); //fill hist. by Gauss
  noiseandsign->Add(noise,sign);
  noiseandsign->Draw();
  noiseandsign->Fit(inttotal);
  sign->Draw("same");
  sign->Fit(int2);
  noiseandsign->GetXaxis()->SetTitle("X");
  noiseandsign->GetYaxis()->SetTitle("Y");
 }
```

Solution (by macro for fitting): Also it is necessary to modify the written macro to draw two functions: exponent ("background") and a Gaussian ("signal").

```
void task3R_25_2() {
  // Add read histogram from file
  TFile *f = TFile::Open("oFile1.root","open");
  TH1F *h = (TH1F*)f->Get("histo");
  TF1 *inttotal = new TF1("inttotal", "expo(0)+gaus(2)",-15,10);
  inttotal->SetParameters(10.,-0.1,1.,-2.,1.6);
  h->Fit(inttotal,"MRE");
  Double_t par[5]; // array
  inttotal->GetParameters(&par[0]);
```

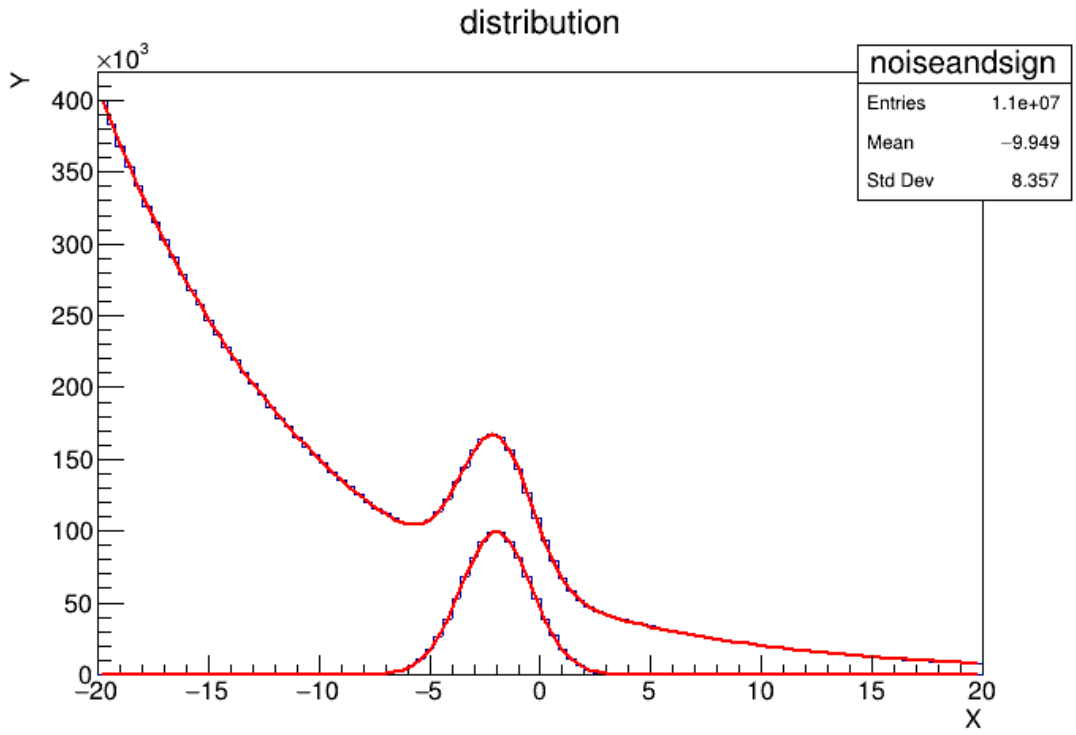

Figure 3.9: Task 4. The result of a fitting using a macro. Separately, the Gaussian.

```
 FCN=103.137 FROM MIGRAD    STATUS=CONVERGED     231 CALLS         232 TOTAL
                     EDM=1.14083e-09    STRATEGY= 1  ERROR MATRIX UNCERTAINTY
2.9 per cent
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           1.02248e+01   1.01726e-03  -3.31312e-06   7.46952e-03
   2  p1          -1.00000e-01     fixed
   3  p2           9.95368e+04   1.78479e+02   2.30506e+00  -9.81175e-08
   4  p3          -1.99704e+00   2.94662e-03  -2.62133e-05  -1.64578e-02
   5  p4           1.59945e+00   2.80616e-03   7.78620e-06  -1.98258e-02
   6  p5           1.02248e+01   1.01726e-03  -3.31312e-06   7.46946e-03
   7  p6          -1.00000e-01     fixed
 FCN=28.8146 FROM MIGRAD    STATUS=CONVERGED      57 CALLS          58 TOTAL
                     EDM=9.34741e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     9.94955e+04   1.21876e+02   2.65234e-01  -2.30603e-08
   2  Mean        -1.99770e+00   1.60387e-03   4.27614e-06   8.51401e-02
   3  Sigma        1.60382e+00   1.13460e-03   5.13237e-07  -3.27866e-02
root [1] Info in <TCanvas::Print>: file /home/alexandra/Tasks/task3R_24.png has
been created
```

Figure 3.10: Task 4. Parameters obtained as a result of fitting.

```
cout<<par[0]<<"   "<<par[1]<<"   " <<par[2]<<endl;
// Add the creation of a signal and background function,
// set the initial parameters from inttotal,
// change the colors of the functions and draw them.
```

```
  TF1 *fon = new TF1("fon", "exp([0]+[1]*x)",-20,20); //creating func.
  TF1 *sign = new TF1("sign", "gaus",-20,20); //creating Gauss
  fon->FixParameter(0, par[0]);
  fon->FixParameter(1, par[1]);
  sign->FixParameter(0, par[2]);
  sign->FixParameter(1, par[3]);
  sign->FixParameter(2, par[4]);
  fon->SetLineColor(3);
  fon->Draw("same");
  sign->Draw("same");
}
void task3R_25() {
  task3R_25_1();
  task3R_25_2();
}
```

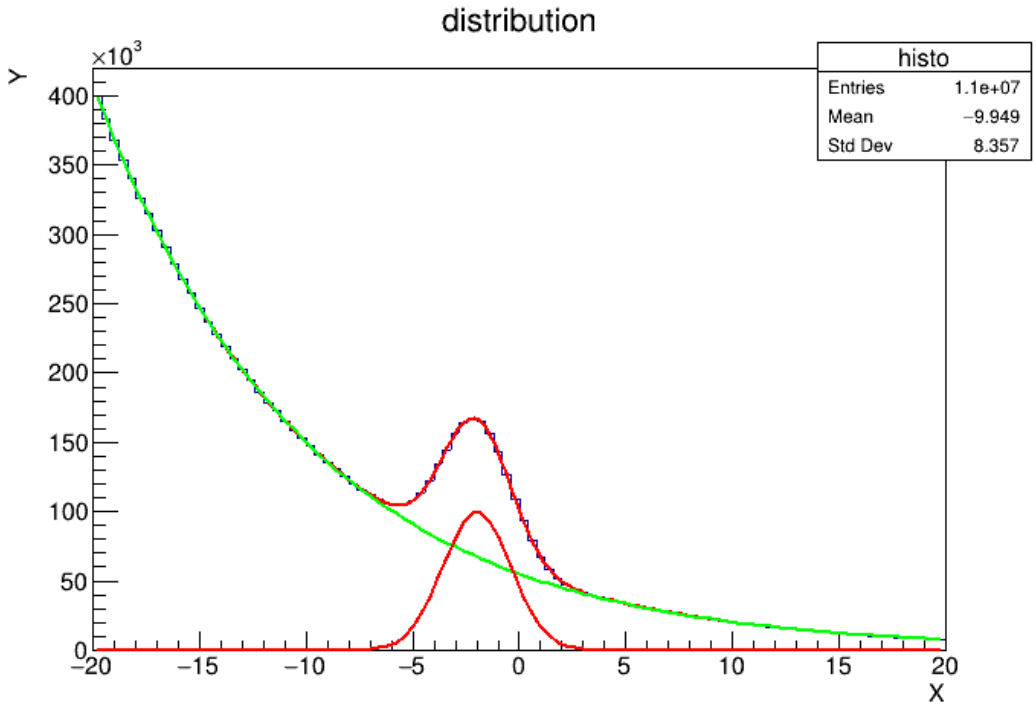

Figure 3.11: Task 4. Fitting a function with drawing "signal" and "background".

Chapter 4

Fitting of the data of the STAR experiment for the production of strange hadrons, such as K_s^0 mesons in Au+Au collisions at $\sqrt{s_{NN}} = 19.6$ GeV and at centralities 0-5%

Experimental data (the transverse momentum spectra of K_s^0 mesons) were taken from the ROOT-file attached to the article [1]. The file contains the following data: graph for the mean value, graphs with statistical and systematic errors. In the ROOT program, three histograms were constructed with the corresponding binning: for the mean value and for the errors. Then the data from the ROOT file was transferred to the created histograms. After that, another histogram was built: the mean value plus the error, which was defined as $Err = \sqrt{Err_{stat}^2 + Err_{sys}^2}$. Then this histogram was fitted with several functions that were discussed in the Introduction section (formulas 1.1-1.3). Following are the parameterizations for these functions.

1. Tsallis-Levy function

This parameterization of Tsallis-Levy function was used for fitting:

$$f = \frac{x[0]([1]-1)([1]-2)}{[1][2]([1][2]+[3]([1]-2))}(1+\frac{\sqrt{[3]^2+x^2}}{[1][2]})^{-[1]}, \tag{4.1}$$

where x - transverse momentum p_T; [0] - normalization; [1] - fit parameter n; [2] - temperature T; [3] - particle rest mass m_0.

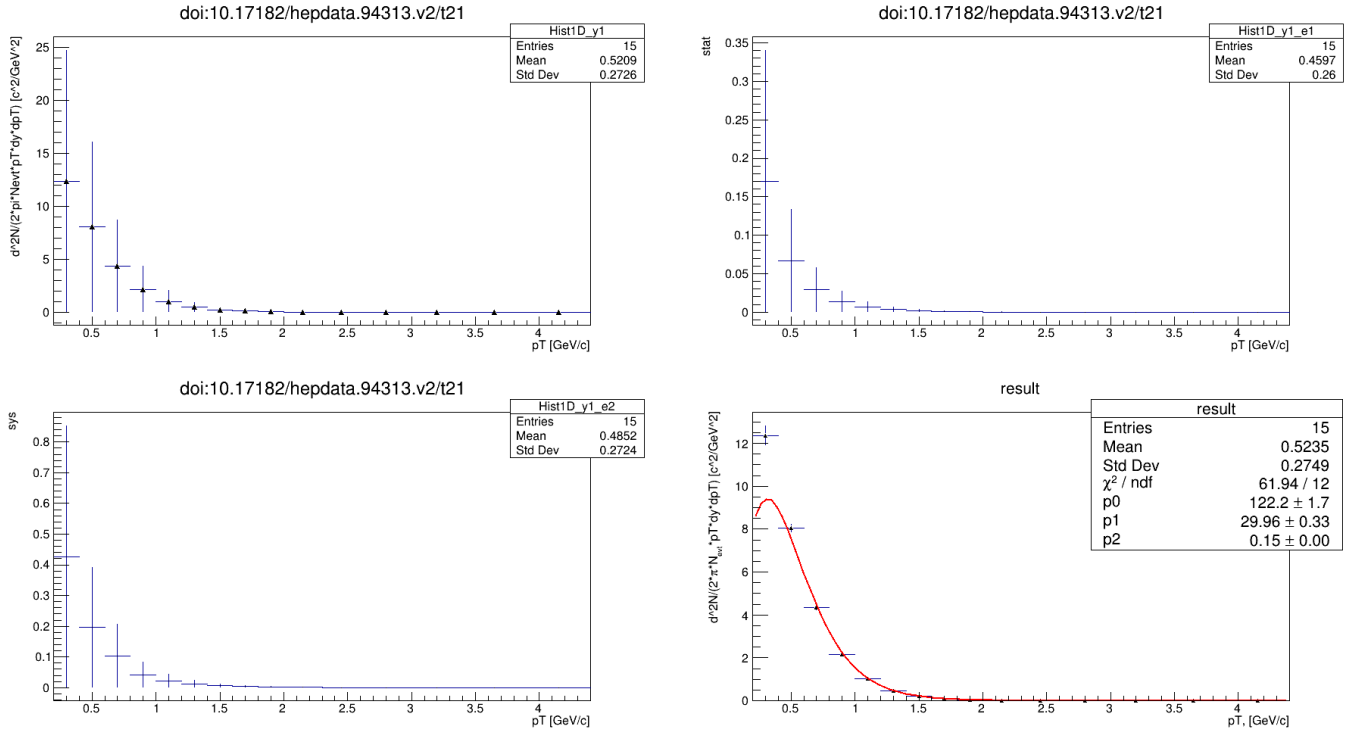

Figure 4.1: Experimental data: mean value, statistical error, systematic error and histogram with mean value and Err error and fit.

Written macro for extracting data from a ROOT-file, splitting a histogram into bins, and also fitting with the Tsalis-Levi function:

```
// Function definition
  Double_t TsallisLevyFunc (Double_t *x, Double_t *par) {
  double norm_num = double((*x)*par[0]*(par[1]-1.)*(par[1]-2.)); //normalization numerator
```

```
  //double norm_num = double(par[0]*(par[1]-1.)*(par[1]-2.)); //normalization denominator
  double norm_den = double(par[1]*par[2]*(par[1]*par[2]+par[3]*(par[1]-2.)));
  //normalization denominator
  double Function_num = double(TMath::Sqrt(par[3]*par[3]+(*x)*(*x)));
  //function numerator
  double Function_den = double(par[1]*par[2]); //function denominator
  return (norm_num/norm_den)*TMath::Power((1+(Function_num/Function_den)),-par[1]);
  // creating the (two-component) Tsallis-Levy function for fitting
}
void FittingData5()
{
  //Checking and opening a file
  TFile *readFile = TFile::Open("HEPData-ins1738953-v2-root.root");
  if (!readFile) {
    cout << "File didn't open"<< endl;
    return -1;
  }
  //readFile->GetListOfKeys()->Print();
  //Creating histograms with mean, statistical and systematic error
  Float_t xbins[16] = {0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.3,
  2.6, 3.0, 3.4, 3.9, 4.4};
  TH1F *result = new TH1F("result", "result", 15, xbins); //creating hist. 15 bins
  // TH1F *h = (TH1F*)readFile->Get("Hist1D_y1");
  TH1F *mean = new TH1F();
  TH1F *stat = new TH1F();
  TH1F *sys = new TH1F();
  // Creating arrays for writing
  Float_t DataMean[15];
  Float_t DataStat[15];
  Float_t DataSys[15];
  Float_t DataErr[15];
  Int_t i;
  // Creating a ROOT file and writing a histogram to it
```

```
TFile* DataFile = new TFile("HEPData-ins1738953-v2-root.root", "READ"); // "o" in "oFile1"
for out
mean = (TH1F*)DataFile->Get("/K0S pT spectrum, Au+Au 19.6 GeV, 0-5%/Hist1D_y1");
stat = (TH1F*)DataFile->Get("/K0S pT spectrum, Au+Au 19.6 GeV, 0-5%/Hist1D_y1_e1");
sys = (TH1F*)DataFile->Get("/K0S pT spectrum, Au+Au 19.6 GeV, 0-5%/Hist1D_y1_e2");
// Write files to an array
for (i=1; i<16; i++) {
  DataMean[i] = mean->GetBinContent(i);
  DataStat[i] = stat->GetBinContent(i);
  DataSys[i] = sys->GetBinContent(i);
  DataErr[i] = TMath::Sqrt(DataStat[i]*DataStat[i]+DataSys[i]*DataSys[i]);
  result->SetBinContent(i,DataMean[i]);
  result->SetBinError(i,DataErr[i]);
  cout<<DataMean[i]<<endl;
  // cout<<result<<endl;
  //gStyle->SetErrorX(DataErr[i]);
}
//Adding a histogram reading from an already open file
TCanvas *c = new TCanvas("c", "c", 1);
c->Divide(2,2);
c->cd(1);
mean->Draw();
// mean->Draw("AC");
mean->SetMarkerStyle(22);
mean->SetMarkerSize(0.95);
c->cd(2);
stat->Draw();
c->cd(3);
sys->Draw();
c->cd(4);
// Fitting
double par[4]= {0.01, 45, 0.16, 0.4976}; // 0- normalization, 1- n, 2- T, 3- m.; // array
  // inttotal->GetParameters(&par[0]);
```

```
  TF1 *FitFunction = new TF1("FitFunction","TsallisLevyFunc",0.2,4.4,4);
// range limits and number of parameters
  FitFunction->SetParameter(0, par[0]);
  FitFunction->SetParameter(1, par[1]);
  FitFunction->SetParLimits(2,0.15,0.17);
  FitFunction->FixParameter(3, par[3]);
  result->Fit(FitFunction,"MRE");
  TCanvas *c1 = new TCanvas("c1", "c1", 1);
  result->Draw("E");
  result->SetMarkerStyle(22);
  result->SetMarkerSize(0.8);
  result->GetXaxis()->SetTitle("pT, [GeV/c]");
  result->GetYaxis()->SetTitle("d^2N/(2*#pi*N_{evt}*pT*dy*dpT) [c^2/GeV^2]");
   return 0;
}
```

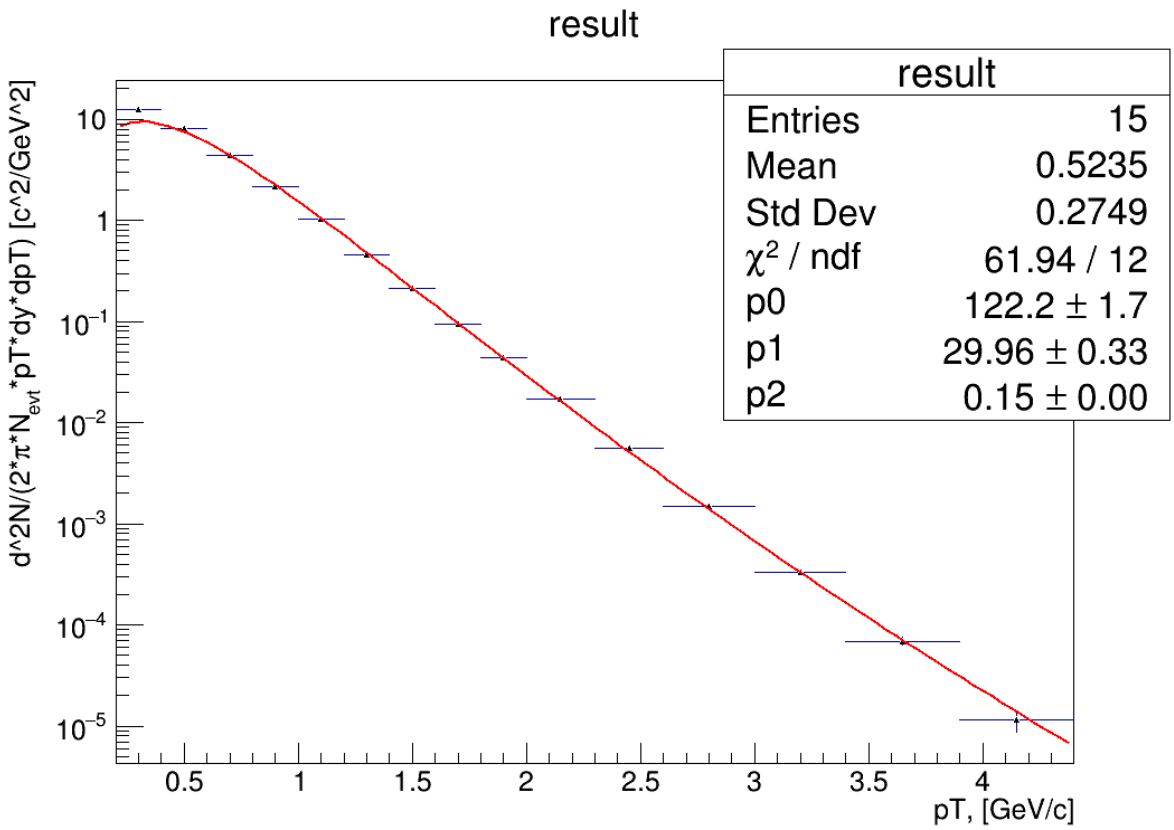

Figure 4.2: The result of fitting the transverse momentum spectra with the Tsalis-Levy function.

2. Power-Law function

Parameterization for this function (see formula 1.2):

$$f = [0](1 + \frac{x}{[2]})^{-[1]}, \tag{4.2}$$

where [0] - normalization; x - transverse momentum; [1] and [2] - parameters for fit (n and p_0 respectively).
The macro code is exactly the same as for fitting with the Tsalis-Levi function, but with the difference that a different function for the fit and other parameters are specified. Here are only those parts of the code that differ:

```
// Function definition
  Double_t PowerLawFunc (Double_t *x, Double_t *par) {
  double bracket = double(1+((*x)/par[2])); //
  return (par[0])*TMath::Power(bracket,-par[1]);
}
  // creating the Power-Law function for fitting
  .....
  // Fitting
   double par[3]= {100, 30, 10}; // 0- normalization B, 1- n, 2- p0; // array
  // inttotal->GetParameters(&par[0]);
  TF1 *FitFunction = new TF1("FitFunction","PowerLawFunc",0.2,4.4,3); // range limits and
  number of parameters
  FitFunction->SetParameter(0, par[0]);
  FitFunction->SetParameter(1, par[1]);
  FitFunction->SetParameter(2, par[2]);
```

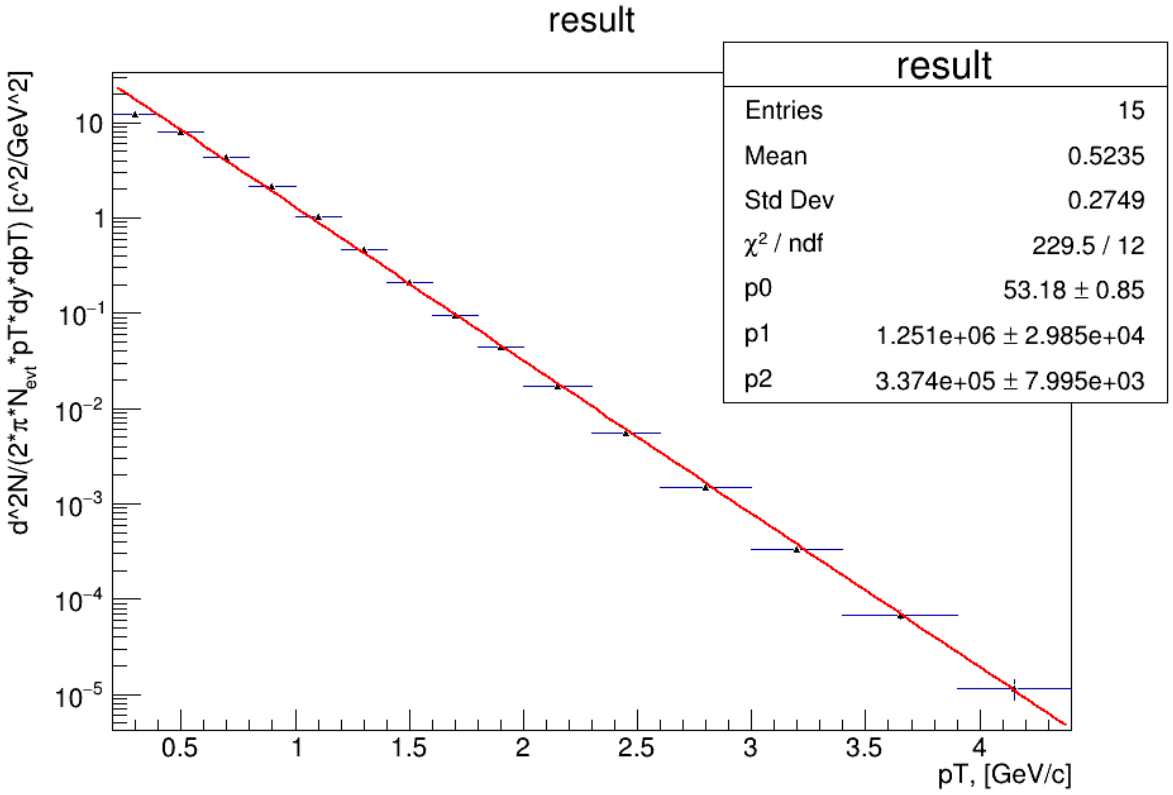

Figure 4.3: The result of fitting the transverse momentum spectra with the Power-Law function.

3. m_t-exponential function

Parametrization (see formula 1.3):

$$f = [0]exp(\frac{\sqrt{x^2 + [2]^2}}{[1]}), \tag{4.3}$$

where [0] - normalization; x - transverse momentum; [1] - temperature T; [2] - particle mass m.
Code fragment:

```
// Function definition
Double_t ExponentialFunc (Double_t *x, Double_t *par) {
  double transverseMass = double(TMath::Sqrt(par[2]*par[2]+(*x)*(*x)));
  // transverse mass mt = sqrt (pt^2+m^2)
  double powexp = double(-(transverseMass)/par[1]); // exp
  return (par[0])*TMath::Exp(powexp); // creating the exponential function for fitting
}
```

```
.....
// Fitting
 double par[3]= {200, 0.15, 0.4976}; // 0- normalization A, 1- T, 2 - K0 mass// array
  // inttotal->GetParameters(&par[0]);
TF1 *FitFunction = new TF1("FitFunction","ExponentialFunc",0.2,4.4,3); // range limits and
number of parameters
FitFunction->SetParameter(0, par[0]);
FitFunction->SetParLimits(1,0.15,0.17);
FitFunction->FixParameter(2, par[2]);
```

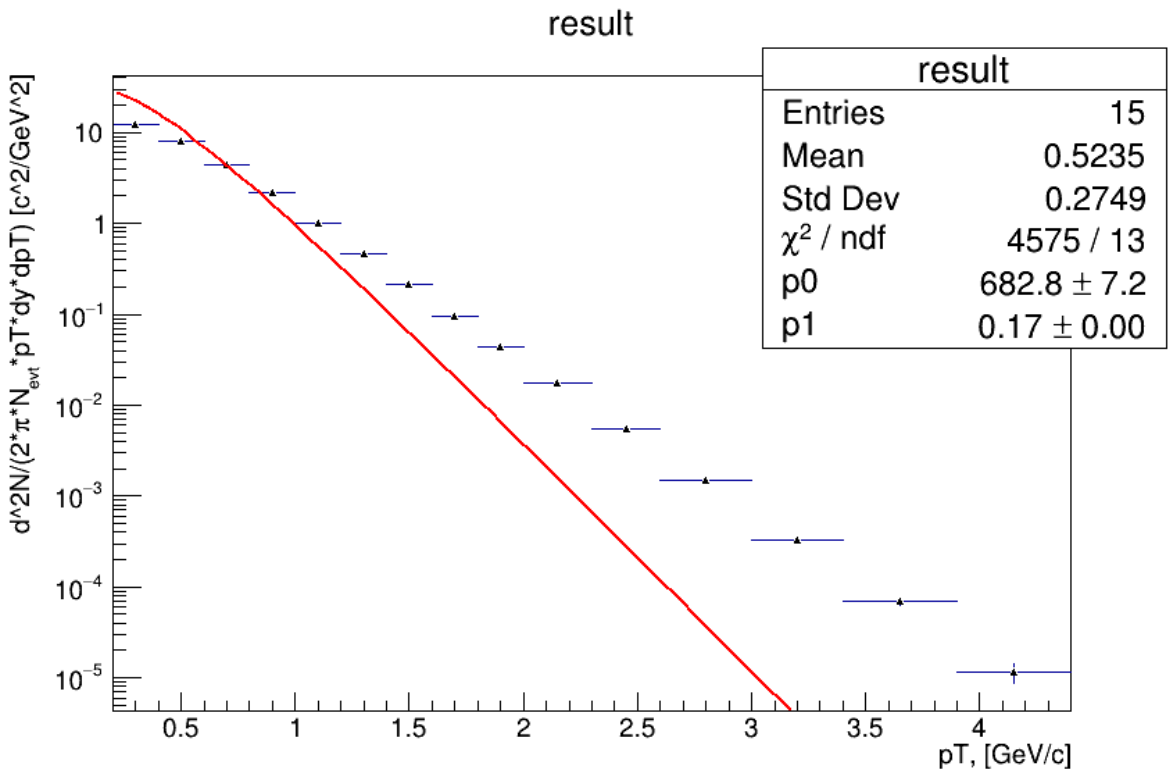

Figure 4.4: The result of fitting the transverse momentum spectra with exponential function.

Chapter 5

Results and Conclusion

1. As a result of the work performed, educational tasks in c ++ and ROOT were solved.

2. Also it was possible to fit the transverse momentum spectra of K_s^0 at mid-rapidity ($|y| < 0.5$) from Au+Au collisions at energy $\sqrt{s_{NN}} = 19.6$ GeV and at centralities 0-5% with several functions, find the fit parameters and χ^2, and compare it. As a result, we can conclude that the Tsallis-Levy function is the best way to describe experimental data.

Below are χ^2 for different functions:

Table 5.1: Comparison of fitting functions

Fitting function	χ^2/ndf	ndf
1. Tsallis-Levy function	5.16	12
2. Power-Law function	19.13	12
3. m_t-exponential function	351.92	13

Bibliography

[1] J. Adam et al. [STAR Collaboration], "Strange hadron production in Au+Au collisions at $\sqrt{s_{NN}}$ = 7.7, 11.5, 19.6, 27, and 39 GeV," arXiv:1906.03732v3 [nucl-ex], 2020.

[2] Q. Wang and F.-H. Liu, "Excitation function of initial temperature of heavy flavor quarkonium emission source in high energy collisions," arXiv:2005.04940v3 [hep-ph], 2020.

[3] B.I. Abelev et al. [STAR Collaboration], "Strange Particle Production in p+p Collisions at $\sqrt{s_{NN}}$ = 200 GeV," arXiv:nucl-ex/0607033v1, 2006.

Acknowledgements

I'd like to express gratitude for my supervisor Grigory Nigmatkulov and also for Aleksey Aparin for the great help in learning the ROOT, and also for the very useful comments, remarks and explanations. The knowledge and skills gained while participating in the START program will certainly be very useful for me in my future scientific work.

Also I would like to thank the organizers of the START program and the staff of the Veksler and Baldin laboratory of High Energy Physics for the opportunity to take part in this programm and for the excellent organization of the process of studying and working at JINR.