Yerevan State University
Department of applied mathematics and informatics
Chair of programming and information technologies

Graduation Thesis

for Bachelor Degree

Title: Creation of a repository of data on the nucleon charge-exchange reactions for data mining, presentation and analysis.

Student: Papikyan Vardanush
Supervisor: Nazaryan Zaven

Yerevan 2010

Contents

1 Introduction. Nucleon charge exchange reactions. Experiment-theory contradiction 3

2 Actuality of the creation of the experimental data repository. Requirements to the repository. Choosing ROOT technologies 6

3 Overview of the ROOT system 6

4 Tree technologies of ROOT for handling with data 11

5 Choice of the algorithm of repository Building 12

6 Description of macro 13

Appendix. Complete macro 26

References 34

1 Introduction. Nucleon charge exchange reactions. Experiment-theory contradiction

Nucleon charge-exchange (NCE) reactions of inclusive neutron production in the proton-fragmentation kinematical region of proton-proton and pion-proton collisions

$$pp \rightarrow nX \qquad (1)$$

and inclusive proton production in the neutron-fragmentation region of positive pion-neutron and proton-neutron collisions

$$\pi^+ n \rightarrow pX \qquad (2)$$

$$pn \rightarrow pX \qquad (3)$$

have always been subject to detailed experimental and theoretical studies aimed at understanding of the interaction mechanisms and internal structure of nucleons.

An intensive experimental investigation of these reactions in a large energy range (from beam energy of 12 GeV in the fixed target experiments to c.m.s. energy of 53.0 GeV in the collider experiments) has been performed in early 70-ths and 80-ths at Fermi National Laboratory (USA) and PS and ISR accelerators of European Laboratory for Fundamental Research (CERN, Switzerland) [1-10].

A detailed theoretical analysis of these data was done in the works [11–15]. The approach was based on the picture of the dominance of the pion exchange in NCE inclusive reactions at small values of momentum transfer from initial to the final nucleon and large values of the momentum of the final nucleon. This dominance is due to the small mass of pion and its strong coupling to the nucleons.

The one-pion-exchange (OPE) Feynman diagram for the process (1) is shown in the Figure 1. The incident beam proton emits a positive pion and and converts to the neutron. The emitted pion interacts with the target proton creating the inclusive system *X*

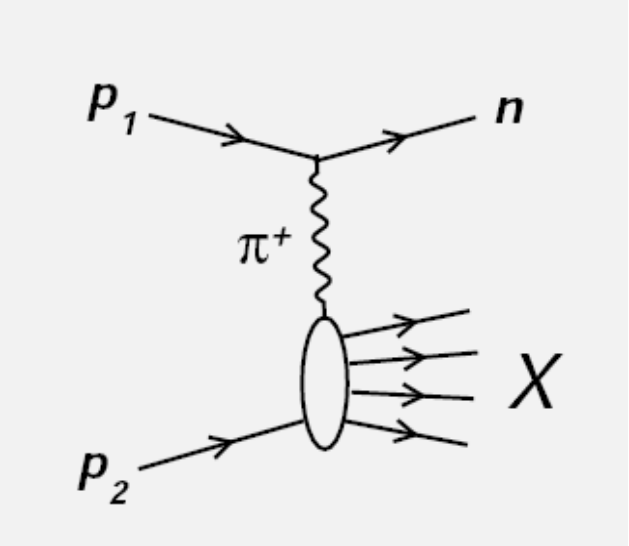

Figure 1. Feynman diagram of the OPE contribution to reaction (1)

The analysis of [11-15] has shown that the majority of data on the reactions (1) - (3) could be well described within the OPE theoretical approach. Meanwhile the high-energy data of ISR on the reaction (1) have revealed a controversial situation. The data at non-zero values of the transversal momentum transfer are lying much lower than the predictions of OPE, while the data at zero value of this momentum are in a good agreement with theory.

Recently, three experiments presented their studies of the inclusive spectra of neutrons. Two of them measured the reaction (1). The beam energy of CERN's NA49 experiment was of 158 GeV [16, 17], while PHENIX experiment of Relativistic Heavy Ion Collider at Brookhaven National Laboratory (USA) was operating at initial c.m.s energy of 200 GeV [18]. The results of NA49 experiment disagree with the standard pion exchange

mechanism and with the zero-angle spectra of ISR [3] while the scarce data of PHENIX (three points only) seem to follow the results of [3].

Production of neutrons in the proton fragmentation region of electroproduction processes was studied by ZEUS and H1 experiments [19–22] at Hadron Electron Ring Accelerator of DESY, (Germany), where the neutrons were produced in the interaction of virtual photon with proton

$$\gamma^* p \rightarrow Xn \qquad (4)$$

The reaction (4) was studied at different values of initial energy virtuality of photon γ* and neutron's momenta. The shape of neutron spectra at large longitudinal and small transversal momentum of neutrons measured in ZEUS experiment reproduces the pion-exchange mechanism predictions, however the absolute values of cross section are noticeably lower than theoretical expectations.

Several theoretical attempts have been undertaken to explain the neutron spectra observed at HERA. So, the authors of [23] considered rescattering mechanism, which makes neutrons spectra to migrate from the large momenta region to the low momenta one. However, the applicability of this mechanism to the nucleon charge-exchange spectra measured in hadron collisions is very questionable.

Hence, one observes a puzzling situation with the description of NCE spectra within pion exchange approach. As distinct from the large variety of the other reactions were a clear universality of pion exchange takes place, the measured NCE processes reveal a considerable deviation from this universality and a detailed global analysis of these processes is needed, with the final goal to develop adequate theoretical picture.

A group of physicists from the Yerevan Physics Institute has undertaken the work on this global analysis. This thesis constitutes a part of this work.

2 Actuality of the creation of the experimental data repository. Requirements to the repository. Choosing the ROOT technologies.

An important step on the way of the understanding of the NCE spectra consists in the creation of a reliable repository of all existing experimental data on the reactions (1) - (4). Such a repository is highly demanded by the physics community. The repository should satisfy the following requirements:

1. Easy retrieval of necessary set of experiment data
2. Possibility of graphical presentation of data
3. Possibility of comparing the theoretical model(s) with the data

The ROOT system [24] is capable to satisfy these requirements.

3 Overview of the ROOT system

ROOT is an object-oriented framework aimed principally at solving the data analysis challenges of high-energy physics. It is created at CERN (The European Organization for Nuclear Research) [25] by a big team of programmers and physicists. Nowadays ROOT is widely used by a very large community of specialists all over the world. ROOT provides storing, processing and analysis of experimental data, 1D, 2D and 3D graphical presentation of these data, Monte Carlo simulation, minimization, etc. ROOT has also, a rich mathematical library allowing to perform complex theoretical calculations. It is anticipated that LHC experiments will produce a few tens Petabytes (10^{15} bytes) of data per year and this data will be handled using ROOT technologies.

3.1 CINT

CINT is a command line C/C++ interpreter, embedded in ROOT system. CINT commands always start with dot. For example, to exit from ROOT one types **.q**
One can also type C++ expressions on the command line. For example:

```
root [0] double value = 4
root [1] sqrt(value)
(const double)2.00000000000000000e+00
```

One can use the command line to execute multi-line commands. To begin a multi-line command you must type a single left curly bracket {, and to end it you must type a single right curly bracket }. For example, let us calculate the sum of the natural numbers from 1 to n:

```
root [0] {
end with '}', '@':abort > float n, sum = 0;
end with '}', '@':abort > cout << "Input n ";
end with '}', '@':abort > cin >> n;
end with '}', '@':abort > sum = ((1 + n) / 2) * n;
end with '}', '@':abort > cout << "sum =  " <<sum<<endl;
end with '}', '@':abort > }
Input n 10
sum =  55
(class ostream)2146336
```

Instead of typing commands on command line, one usually stores code in file (aka macro). The macros can be unnamed and named.

Unnamed macros contain a set of C++ and CINT commands enclosed in curly brackets. Unnamed macros are loaded and executed with **.x** CINT command

x. filename

Named macros contain one or more function definitions. They must not be enclosed in curly brackets.
To execute a named macro, one first loads this macro with .L command, and than type the name of function inside of macro. For example:

root [0] .L filename

root [1] main()

3.2 Coding convention specifics of ROOT

Classes begin with T:	**TObject**
Non-class types end with _t	**Int_t**
Data members begin with f:	**fTree**
Member functions begin with a capital:	**Loop()**
Constants begin with k:	**kInitialSize**
Global variables begin with g:	**gSystem**
Static data members begin with fg	**fgTokenClient**
Enumeration types begin with E:	**EColorLevel**
Locals and parameters begin with a lower case:	**nbytes**
Getters and setters begin with Get and Set:	**GetFirst(), SetLast()**

3.3 Architecture of ROOT

The backbone of the ROOT architecture is a layered **class hierarchy** with, currently, around 1200 classes. These classes are grouped in about 60 frameworks (libraries) divided in 19 main categories (modules) (see Figure 2)

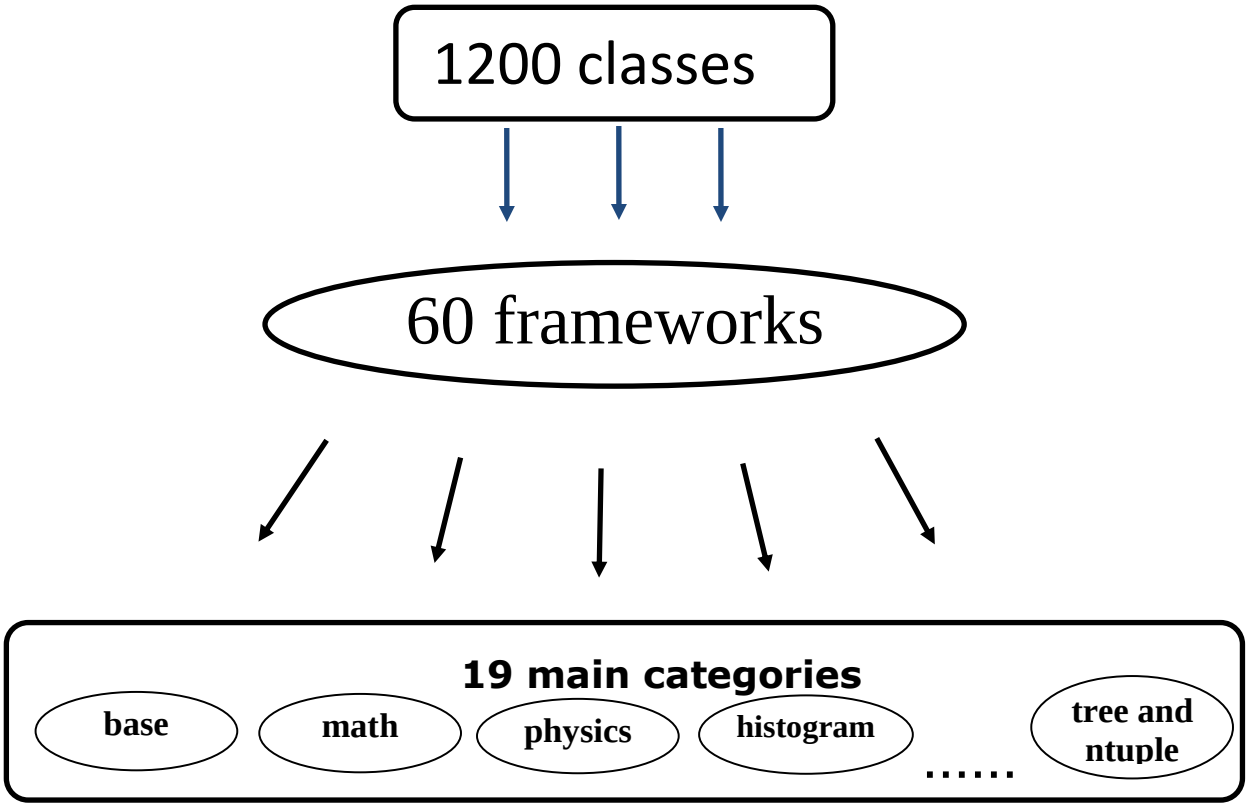

Figure 2 Schematic view of ROOT architecture

3.4 Some main categories

Base category: This category consists of the classes that provide the functionality of the ROOT. For example, the **TObject** class (mother of all ROOT objects) provides default behaviour for all objects in the ROOT system, therefore most of the classes inherit from **TObject**. It provides

protocol (abstract) member functions for object I/O, error handling, sorting, inspection, printing, drawing, etc. The other class, **TFile** provides a hierarchical sequential and direct access persistent object store. Using the **TBrowser** one can browse all ROOT objects and see the content of the selected classes.

Container category: Its classes provide general purpose data structures like arrays, lists, sets, maps, etc.

Matrix category: Provides classes for matrices and vectors.

Math category: Enables access to rich mathematical library of CERN.

Histogram category: Provides classes for advanced statistical data analysis, like 1D, 2D and 3D histogramming of short, long, float or double values, with fixed or variable bin sizes, profile histograms and formula evaluation.

Tree category: One of the classes of this category is **TTree** class. It allows to store large quantities of same-class objects/data/. A **TTree** can hold all kind of data - objects, arrays and simple types. The class is optimized to reduce disk space and enhance access speed. Using a rich graphical environment of ROOT, data written to trees can be represented through different kind of plots for drawing histograms and graphs.

Tree category is used in this Diploma to store the NEC data and handle with these data.

4 Tree technologies of ROOT for handling with data

4.1 TTree and TBranch classes

A **TTree** object consists of a list of branches. The organization of branches allows to optimize the data for their convenient use. The branch class is called TBranch. If variables are not to be used together, they are normally placed on separate branches. A variable on a TBranch is called a leaf. If the variables are related, such as the coordinates of a point, it is most efficient to keep coordinate values in the leaves of the one branch.
The branch type differs by what is stored in it. A branch can hold an entire object, a list of simple variables, contents of a folder, contents of a TList, or an array of objects.

4.1.1 Branching of a tree

The branching-level of a tree can take values from 1 to 99. If the split-level is set to 1, the whole object is written in its entirety to one branch. The **TTree** will then look like the one on the left, with one branch and one leaf holding the entire event object. If the TTree object has more than one branches, with a single leaf on each branch, then it will look as one shown on the right picture (5-branch object). A branched tree is faster to read because variables of the same type are stored consecutively and the type does not have to be read each time.

4.2 ROOT file

A ROOT file is a compressed binary file in which we can save objects of any type. The data can be saved into one or several ROOT files. ROOT trees spread over several files can be chained and accessed as a unique object, allowing for loops over huge amounts of data.
Actually, one ROOT file can look quite similar to the file system provided by your operative system, because it can be internally structured in different folders (or "*directories*"), each of them containing your data or other folders. Hence, one can navigate through an open ROOT file (using the **TBrowser**) practically in the same way as one can browse the files by moving from a folder to another one.

5 Choice of the algorithm of repository building

The NCE reactions have been studied at collider (when two accelerated beams collide) and fixed target (when a beam is impinging on a fixed target) accelerators. The data on the reaction (1) have been obtained in both fixed target and collider experiments. The beam momentum in fixed target experiments was 12 and 24 GeV/c, while the values of collision energy in the ISR collider experiments were 22.5, 30.6, 44.6, 53.0 GeV. These experiments analyzed the dependence of neutron spectra on Feynman scaling variable x at different fixed values of transverse momentum (from 0.0 to 1.0 GeV/c). RHIC collider analyzed the x dependence at 200 GeV for zero transverse momentum.
The reactions (2) and (3) can be studied in fixed target experiment only. The distribution of protons over x and transferred momentum square was studied in the reaction (2) with the pion beam momentum of 100 GeV/c and in the reaction (3) with the proton beam momentum of 400 GeV/c.

The reaction (4) has been studied in the electron-proton and positron-proton collisions at center-of-mass system energy of 318 GeV. The data have been taken at several values of $\gamma^* p$ system center-of-mass energy,

virtuality of photon, as well as at different values of x and transverse momentum of the produced neutrons

The types of accelerators and experiments presented above as well as the structure of the data obtained in these experiments define the architecture of our repository and the corresponding search algorithm (see Section 6.2).

6 Description of macro

Schematically the macro consists of three main blocks (Figure 3). In section 6.1-6.3 the functionality of these blocks is described.

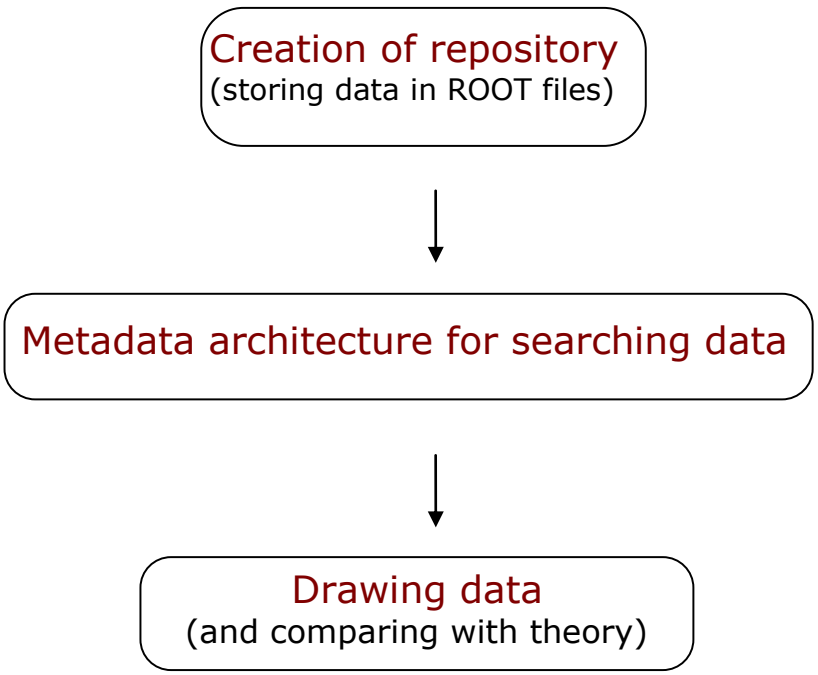

Figure 3. Schematic view of the architecture of macro

6.1 Creation of repository

Initially, the information on the data is compiled (written) by hand in the form of ASCII files. Below we give an example of such a file. It contains the

characteristics of the reaction *pp->nX* measured at ISR collider in the collision of protons at the energy of 52.8 GeV, for the neutrons with the zero value of transverse momentum. The dependence of the reaction cross section on the Feynman x-variable has been measured.

0.15	0.0	10.9	1.4
0.20	0.0	10.0	1.3
0.25	0.0	9.9	1.2
0.30	0.0	10.0	1.2
0.35	0.0	10.4	1.1
0.4	0.0	10.9	1.1
0.45	0.0	11.4	1.0
0.5	0.0	12.2	1.0
0.55	0.0	13.3	1.1
0.6	0.0	14.7	1.1
0.65	0.0	17.0	1.3
0.7	0.0	19.8	1.4
0.75	0.0	23.6	1.6
0.775	0.0	25.5	1.6
0.8	0.0	26.7	1.6
0.825	0.0	26.7	1.6
0.85	0.0	25.7	1.7
0.875	0.0	23.8	1.8
0.9	0.0	19.3	2.6
0.925	0.0	13.8	2.6
0.95	0.0	8.3	2.6
0.975	0.0	2.7	2.6

Hence, the ASCII file represents a record of the values of four variables. The values of *x* (x-variable), transverse momentum (pt-variable) and cross section (f-variable) are written in the first, second and third columns, correspondingly. The errors of the cross section measurements (deltaf-variable) are recorded in the fourth column.

The part of macro concerning the repository creation consists of three functions:

CreateExpDataTree, CreateBranch and FillData.

The following steps are done in the CreateExpDataTree function:

- The ExpData.root ROOT file is created

  ```
  TFile *f = new TFile(Form("%sExpData.root",gDir.Data()),"RECREATE");
  ```

  "RECREATE" options creates file, and overwrites the existing file;

- A tree is created

  ```
  TTree *tree = new TTree("T","Experimental data for different experiments");
  ```

- CreateBranch function is called and the branches are created, one branch per experiment

  ```
  branchName = "Experiment_for_ppnX_ISR_30GeV_pt02";
  fileName = "ppnX_ISR_pt0.2_Engler_et_al30GeV.txt";
  CreateBranch (tree,branchName, fileName);
  ```

CreateBranch function consists of the following steps:

- In order to register the values of the variables x, pt, f and deltaf , a tree is created with the leaves x, pt, f and deltaf

  ```
  tree->Branch(branchName,&ExpData.x,"x/F:pt:f:deltaf");
  ```

- The ASCII file corresponding to a given tree is open

  ```
  in.open (gDir + fileName);
  ```

. The ASCII file is read by the FillData function (with the help of the ifstrem type in object)

The FillData function returns the integer type numEntries number, which is equal to the number of the values of the variables (the number of the lines in ASCII file) measured in a given experiment

```
numEntries = FillData(in,&ExpData,tree);
```

As a result, an auxiliary branch (Num) with one leaf is created in the CreateBranch . The leaf contains the value of the variable numEntries

```
treeBranch(Form("%s_Num", branchName,&numEntries,"numEntries/I");
```

At the last step, the object created in CreateExpDataTree is written to the file ExpData.root and the file is closed

```
tree->Write();
f->Close();
```

6.2 Search for the data

The part of the macro concerning the organization of the data mining looks as follows:

The needed experimental data (branches) are searched with the help of the function SearchTBranch. In this function, the four pointers to the four arrays (a, b, c and d) are declared

The aray "a" contains the types of accelerators

```
Char_t* a[ 2 ] = { "collider", "fixed target" };
```

The array "b" contains the types of reactions

```
Char_t* b[ 4 ] = { "pp->nX", "pn->pX", "pin->pX", "gammap->nX" };
```

In the array "c", one keeps the values of the energy

```
Char_t* c [ 4 ] = { "23", "30", "44", "53" };
```

The values of the transverse momentum are written to the array "d"

```
Char_t* d [ 5 ] = { "0", "0.2", "0.4", "0.6", "0.8"};
```

The search for a given experimental data is done as follows:

- One loads the macro

  root [ ] .L macro.C

- The SearchTBranch function is executed

  root [ ] SearchTBranch( )

- The following lines are typed on the screen

  Choose accelerator type

  1 collider

  2 fixed target

In order to get the collider data, one should enter "1", while for the fixed target data one should type "2". The entered number is written to the first cell of a 4-element array "z" of integer type.

- The following lines are then displayed on the screen

  Choose type of reaction

  1 pp->nX

  2 pn->pX

  3 pip->pX

  4 gammap->nX

- Similarly, the values of the energy and transverse momenta integer identifiers have to be chosen. They are stored in the other cells of the array "z".

In one other two-dimensional array "f" one stores the names of branches. Using the values of "z" as indices for "f", one gets the name of the searched branch. This name is returned to ReadData function for the presentation of data.

6.3 Data presentation

The part of macro concerning the organization of data presentation is as follows:

In the ReadData function:

- One opens the ROOT file

```
TFile* f = new TFile(gDir + "ExpData.root");
```

- One gets the tree stored in the ROOT file

```
TTree* tree = (TTree*)f->Get("T");
```

- One reads (following the value of the argument returned to the function) the branch, together with corresponding to it Num auxiliary branch

```
ExpData_t Exp1Data;
Int_t num
TBranch* b = tree->GetBranch(Form("%s_Num", branchName));
b->SetAddress(&num);
b->GetEntry(0);
b = tree->GetBranch(branchName);
b->SetAddress(&Exp1Data);
```

- In order to present the data together with their errors, one uses TGraphError class

```
TGraphErrors* graph1 = new TGraphErrors(num,x1,f1,deltax1,deltaf1);
```

- Finally, for the desired display of data, one applies the Draw method of ROOT

```
graph1->Draw("A*");
```

The values of the x (f) variable are put on the x (y) axis.

Figure 4 demonstrates an example of the macro's application. The data on the reaction *pp* –> *nX* obtained in 52.8 GeV ISR experiment, at fixed zero neutron transverse momentum are presented.

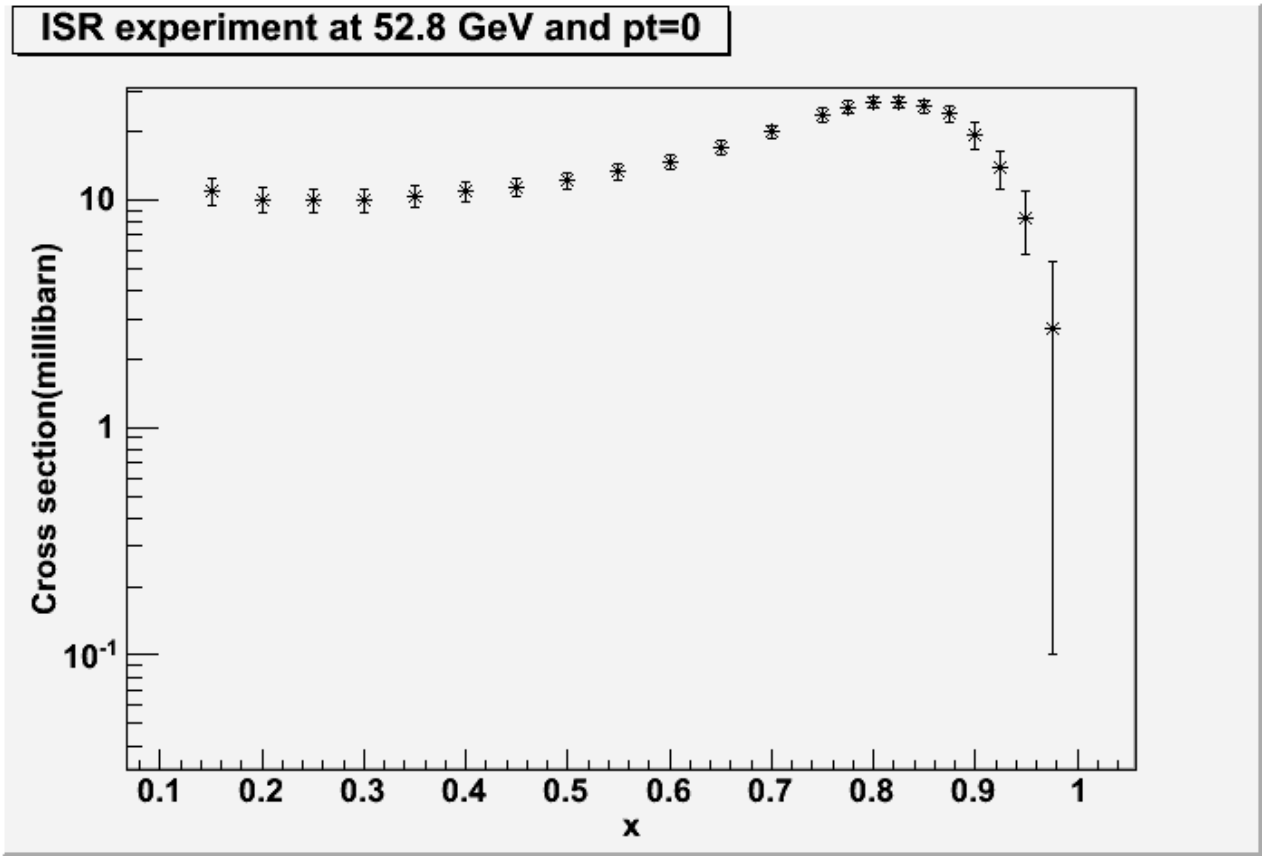

Figure 4 Dependence of the cross section of the neutron production in *pp* –> *nX* reaction. The data are obtained in the *pp* collisions at energy of 52.8 GeV, for the zero-angle neutron production.

6.4 Part of the macro concerning theoretical model and theory-experiment comparison

In this Section, the theoretical OPE model is described as well as the corresponding part of macro, devoted to the comparison of the theoretical predictions with the data of [1,3] obtained in pp collisions at the ISR energies.

6.4.1 Theoretical model

In the OPE model of [14], the invariant cross section of the process (1) has the following form

$$f = E_n \frac{d^3\sigma}{d^3 p_n} = \frac{G^2_{\pi NN} \mid t \mid}{(2\pi)^3 p_{in}\sqrt{s}} \mid F(s, M_X^2, t) \mid^2 \; p_\pi \sqrt{M_X^2}\, \sigma^{\pi^+ p}_{tot}(M_X^2) \qquad (5)$$

The quantities in the formula (5) are as follows:

$$s = (p_1 + p_2)^2 \qquad (6)$$

is the s-channel invariant, with p_1 and p_2 being the 4-dimensional momenta of the colliding protons (see Figure 1);

$$t = (p_n - p_1)^2 \qquad (7)$$

is the t-channel invariant variable, square of the 4-dimensional momentum transfer from the produced neutron to the incoming proton. p_n is the 4-momentum of the neutron and E_n is its energy in the center of mass system (c.m.s.) of the reaction (1);

$$M_X^2 = (p_1 + p_2 - p_n)^2 \qquad (8)$$

is the square of the invariant mass of the X-system;

$G_{\pi NN}$ is the dimensionless constant of the interaction of pion with nucleons:

$$G^2_{\pi NN} / 4\pi = 14.6 \qquad (9)$$

p_{in} is the initial momentum in the overall c.m.s.;

$$p_{in} = \sqrt{(s - 4M_N^2)} / 2 \qquad (10)$$

with M_N=0.94 GeV being the nucleon mass;

p_π is the value of virtual pion momentum in the c.m.s. of the pion and p_2,

$$p_\pi = \sqrt{(M_X^2 - M_N^2 - M_\pi^2)^2 - 4M_N^2 M_\pi^2} / 2M_X \qquad (11)$$

with pion mass M_π = 0.14 GeV;

$\sigma_{tot}^{\pi^+ p}(M_X^2)$ is the total cross section of the $\pi^+ p$ interaction;

The function entering (5) is the pion Green function. In the OPE model of [14] it is parameterized in the following way:

$$F(s, M_X^2, t) = \frac{\pi\alpha'}{2\sin(\pi\alpha'(t - M_\pi^2)/2)} \exp\{\Lambda(t - M_\pi^2)\} \qquad \text{for } |t|<|t_0| \quad (12)$$

$$F(s, M_X^2, t) = \frac{\pi\alpha'}{2\sin(\pi\alpha'(t_o - M_\pi^2)/2)} \exp\{\Lambda(t - M_\pi^2) + R_2^2(t - t_o)\}$$

for $|t|>|t_0|$ (13).

In (12) and (13) $\Lambda = R_1^2 + \alpha' \ln(s/s_o)$, $R_1^2 = 0.3\text{GeV}^{-2}, R_2^2 = 0.74\text{GeV}^{-2}$

α'=1 GeV^{-2}, s_0=1 GeV^2, t_0= –0.7 GeV^2.

In this thesis, we will compare the predictions of the OPE model with the data of the ISR experiments. These experiments were studying the dependence of the inclusive cross section f on neutron's Feynman scaling variable x at several fixed values of transverse momenta p_n^T of neutrons.

The Feynman scaling variable x is defined as

$$x = p_n^L / p_{in}, \text{ that is } p_n^L = x p_{in} \qquad (14)$$

where p_n^L is the longitudinal component of the neutron 3-momentum.

In (5), the following variables depend on the x and p_n^T: E_n, p_π, t and M_X^2.

The explicit expression for E_n in terms of x and p_n^T is

$$E_n = \sqrt{(x\, p_{in})^2 + (p_n^T)^2 + M_N^2} \qquad (15)$$

The variables t and M_X^2 are rewritten through p_n^L, p_n^T *and* E_n as:

$$t = (E_n - E_{in})^2 - (p_n^L - p_{in})^2 - (p_n^T)^2 \quad (16)$$

$$M_X^2 = s - 2\sqrt{s}\,E_n + M_N^2 \quad (17)$$

The formula for p_π is given by (11).

The expressions (11), (16) and (17) defines all the variables entering (5) as functions of x and p_n^T.

In the considered kinematical region of 1.0 >x>0.6 and the initial ISR collision energies, the value of the variable M_X is >20 GeV. At this values, the total cross section of the the $\pi^+ p$ interaction is constant, equal to 22 millibarn.

6.4.2 Programming the theoretical part

The part of macro corresponding to the description, within the OPE model, of the dependence of the cross section of the process *pp->nX* on the scaling variable x consists in the construction of function f (see (5)) through a consecutive construction of all variables of (5) depending on x and p_n^T. Below we give the part of macro concerning construction of E_n

```
Defining function E (Energy of created particle)------------
Double_t E(Double_t xf, Double_t pt)
{
return (sqrt((xf*PA)*(xf*PA)+pt*pt+MN2))
}
```

Constructing also the other variables, one finally gets the invariant cross section as function of x and p_n^T

```
//**Invariant Rho-function for pp->nX as function of xf and pt **
Double_t RHOxfpt(Double_t xf, Double_t pt)
{
Double_t RHOxfpt = 1.0/(pow(2.*PI,3)*PA*U)*G2*(-t(xf,
pt))*Greenpi(xf,pt)*Greenpi(xf,pt)*
ppivirt(xf,pt)*sqrt(s1(xf, pt))*stotpiplusn;


return(RHOxfpt);
```

```
}
```

In order to compare the theoretical predictions with the experiments at fixed p_n^T (p_n^T =0.0 in the considered case), one first defines the cross section at p_n^T =0.0

```
//---Invariant Rho-function for pp->nX as function of xf at pt=0.--
Double_t RHOxfptzero(Double_t xf)
{
Double_t RHOxfptzero = RHOxfpt(xf, 0.0);
return(RHOxfptzero);
}
Int_t main()
{
```

and then one proceeds to the function of one variable (x), using the ROOT constructor of the function of one argument, allowing to draw the dependence on x

```
TF1 * f = new TF1("found", "RHOxfptzero(x)", 0.6, 0.95);
f->Draw();
gPad->SetLogy();
return 0;
}
```

In a similar way, one can construct the theoretical distributions over x for any fixed value of the neutron transverse momentum.

6.4.3 Theory-experiment comparison

Predictions of the OPE model and the experimental data are plotted in the Figure 5. The comparison shows that OPE mechanism describes well the data at large values (> 0.6) of Feynman scaling variable *x.*

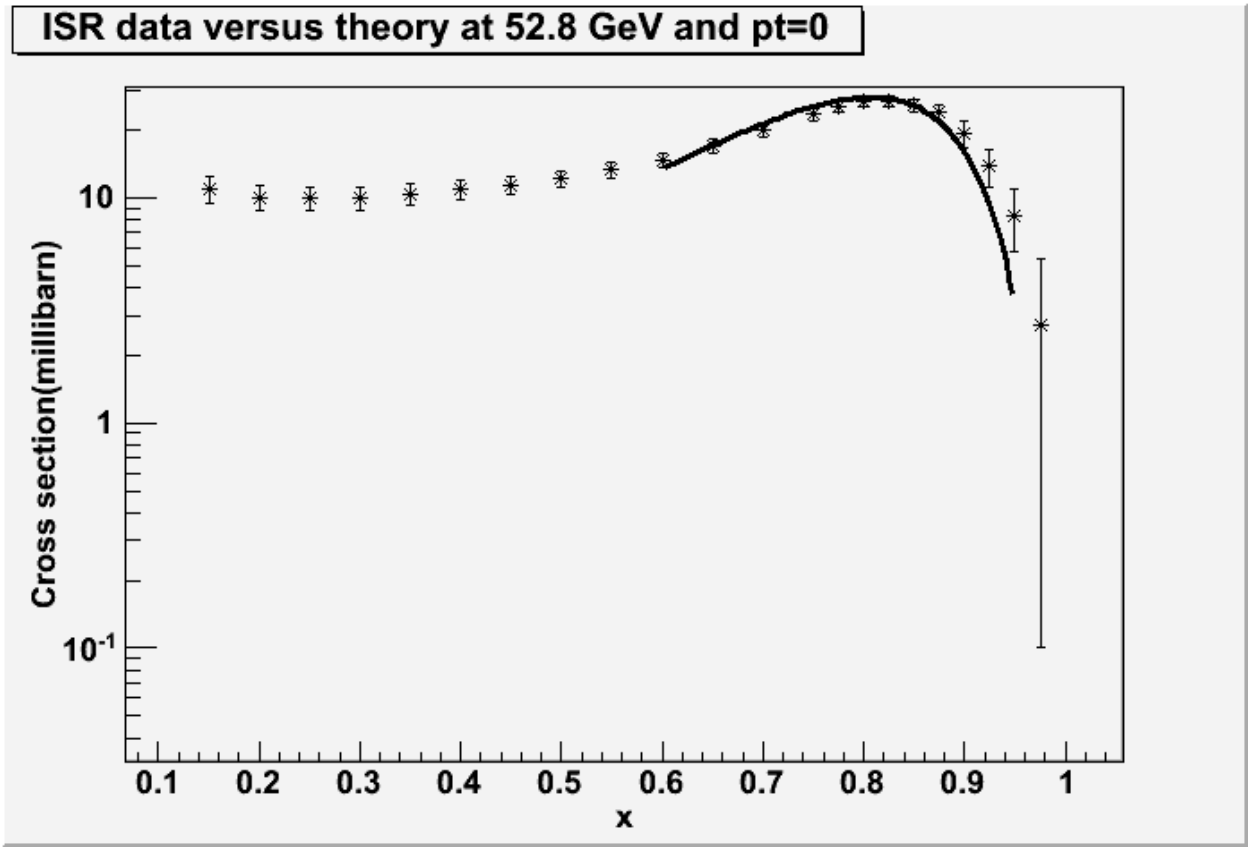

Figure 5 Inclusive cross section of *pp* -> *nX* process as a function of *x* at the collision energy of 52.8 GeV and p_n^T = 0.0 GeV. The stars represent experimental data of [3] and the solid line corresponds to the theoretical calculations.

Figure 6 demonstrates the comparision of the experimental data and theoretical predictions for the non-zero, p_n^T = 0.4 GeV, value of the produced neutrons. The data are obtained [1] at ISR collider in the pp collisions at energy of 52.8 GeV. As it was noticed in the Introduction, in the region of $x > 0.6$ the measured values of the cross section lie considerably (2-3 times) lower than the OPE predictions.

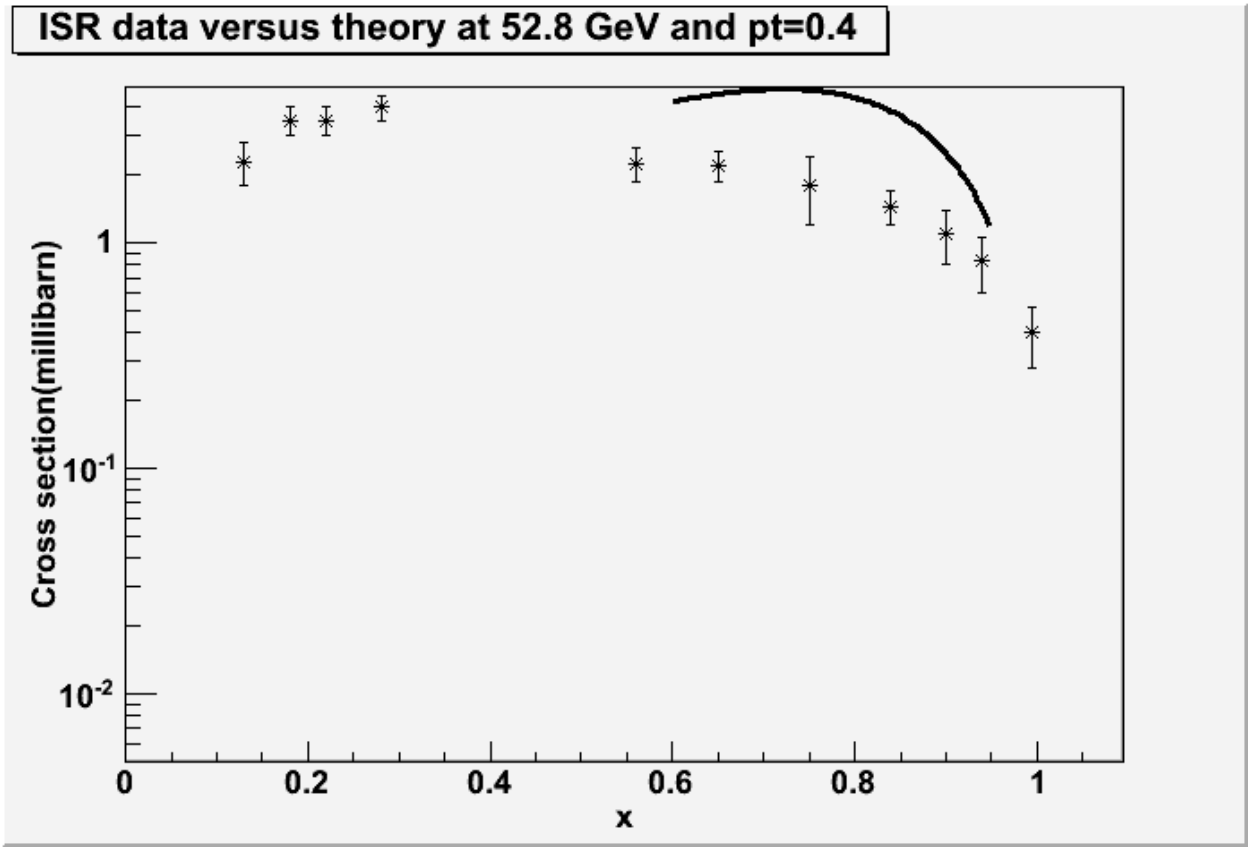

Figure 6 Inclusive cross section of *pp -> nX* process as a function of *x* at the collision energy of 52.8 GeV and p_n^T = 0.4 GeV. The stars represent experimental data of [1] and the solid line corresponds to the theoretical calculations.

Appendix. Complete macro

// main.C

```
#include <iostream>
#include "RHOxfpt04_53GeV_07.04.2010.C"
TString gDir;
void CreateBranch (TTree*, Char_t*, Char_t*, TString );
void CreateExpDataTree();
void ReadData(Char_t*);
Int_t FillData(std::ifstream&,struct ExpData_t*, TTree*);
void SearchTBranch ();
void DrawMyTGraph();

struct  ExpData_t {
        Float_t         x;
        Float_t         pt;
        Float_t         f;
        Float_t         deltaf;

        };

    int main() {
    CreateExpDataTree();
    return 0;
    }
// Create a tree with experimental data.
void CreateExpDataTree() {

    gDir = gSystem->UnixPathName(gInterpreter->GetCurrentMacroName());
    gDir.ReplaceAll("main10_05.C","");
    gDir.ReplaceAll("/./","/");

    // Create a new ROOT file
    TFile *f = new TFile(Form("%sExpData.root",gDir.Data()),"RECREATE");

    // Create a tree
    TTree *tree = new TTree("T","Experimental data for different
experiment");
    Char_t* branchName;
    Char_t* fileName;

    branchName = "Experiment_for_ppnX_ISR_22GeV_pt02";
    fileName = "ppnX_ISR_pt0.2_Engler_et_al22GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_30GeV_pt02";
           fileName = "ppnX_ISR_pt0.2_Engler_et_al30GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_44GeV_pt02";
           fileName = "ppnX_ISR_pt0.2_Engler_et_al44GeV.txt";
    CreateBranch (tree,branchName, fileName);


    branchName = "Experiment_for_ppnX_ISR_53GeV_pt02";
           fileName = "ppnX_ISR_pt0.2_Engler_et_al53GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_30GeV_pt04";
           fileName = "ppnX_ISR_pt0.4_Engler_et_al30GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_44GeV_pt04";
```

```
        fileName = "ppnX_ISR_pt0.4_Engler_et_al44GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_53GeV_pt04";
        fileName = "ppnX_ISR_pt0.4_Engler_et_al53GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_30GeV_pt06";
        fileName = "ppnX_ISR_pt0.6_Engler_et_al30GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_44GeV_pt06";
        fileName = "ppnX_ISR_pt0.6_Engler_et_al44GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_53GeV_pt06";
        fileName = "ppnX_ISR_pt0.6_Engler_et_al53GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_30GeV_pt08";
        fileName = "ppnX_ISR_pt0.8_Engler_et_al30GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_44GeV_pt08";
        fileName = "ppnX_ISR_pt0.8_Engler_et_al44GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_53GeV_pt08";
        fileName = "ppnX_ISR_pt0.8_Engler_et_al53GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_30GeV_pt0";
        fileName = "ppnX_ISR_pt0_30GeV.txt";
    CreateBranch (tree,branchName, fileName);

    branchName = "Experiment_for_ppnX_ISR_53GeV_pt0";
        fileName = "ppnX_ISR_pt0_53GeV.txt";
    CreateBranch (tree,branchName, fileName);

       // Write the tree to the file
    tree->Write();
    f->Close();

}
    // Create a branch
void CreateBranch (TTree* tree, Char_t* branchName, Char_t* fileName) {

    //cout<<branchName <<endl<<fileName<<endl;

    ExpData_t ExpData;
    tree->Branch(branchName,&ExpData.x,"x/F:pt:f:deltaf");
    ifstream in;
    //cout<<dir+fileName<<endl;
    in.open (gDir + fileName);
    Int_t numEntries;
    numEntries = FillData(in,&ExpData,tree);
    //cout<<numEntries<<endl;
    in.close();
    tree->Branch(Form("%s_Num", branchName),&numEntries,"numEntries/I");
        tree->Fill();

    }
```

```
// read the ASCII file, put the data into the tree and return number of
entries

Int_t FillData(std::ifstream& in,struct ExpData_t* ExpData, TTree* tree)
{


Int_t numEntries=0;

    while (1) {

        // read x
        in >> ExpData->x;
        if (in.eof()) break;

        // read pt
        in >> ExpData->pt;

        // read f
        in >> ExpData->f;

        // read feltaf
        in >> ExpData->deltaf;

        //cout << ExpData->x << " " << ExpData->pt << " " << ExpData->f << "
" << ExpData->deltaf << endl;

                // Count the number of entries read
        ++numEntries;

        // Put the read data to the tree
        tree->Fill();
    }

    return numEntries;
}



// Search necessary branch of tree
void SearchTBranch () {

    Int_t n = 2, m = 4, r = 5;
    Char_t* a[ 2 ]  = { "collider", "fixed target" };
    //cout<<b[0]<<endl<<b[1]<<endl;

    Char_t* b[ 4 ] = { "pp->nX", "pn->pX", "pip->nX", "gammap->nX" };
    //cout<< a[0]<<endl<<a[1]<<endl<<a[2]<<endl<<a[3]<<endl;

    Char_t* c [ 4 ] = {  "23","30", "44", "53" };
     //cout<<c[0]<<endl<<c[1]<<endl;

    Char_t* d [ 5 ] = { "0", "0.2", "0.4", "0.6", "0.8"};


    Int_t z[ 4 ] ;
    Int_t j;

    cout << "Choose accelerator type " << endl << "1 " << a[0] << endl << "2
" << a[1] << endl;
        cin >> j;
    //cout<<"j= "<<j<<endl;

    while(j != 1 && j != 2) {
```

```
          cout << "You entered wrong data. You can choose 1 or 2. Try again.
" << endl;
        cout << "Choose accelerator type " << endl << "1 " << a[0] << endl <<
"2 " << a[1] << endl;
        cin >> j;
    }

    if( j == 1 ) {
        cout<< "There are not data for collider yet\n";
        exit(1);
        }
        else z[0] = j;


    cout << "Choose type of reaction " << endl << "1 " << b[0] << endl <<"2 "
<< b[1] << endl << "3 " << b[2] << endl << "4 " << b[3] << endl;
    cin >> j;

        while( j < 1 || j > 4) {
          cout << "You entered wrong data. You can choose 1,2,3 or 4. Try
again. " << endl;
        cout << "Choose type of reaction " << endl << "1 " << b[0] << endl
<<"2 " << b[1] << endl << "3 " << b[2] << endl << "4 " << b[3] << endl;
        cin >> j;
    }

     z[ 1 ] = j ;


    if ( z[1] == 2 || z[1] == 3 || z[1] == 4 ) {
        cout<< "There are not data  for " << b[1] << " yet\n";
        exit(1);
     }

      cout << "Choose energy " << endl << "1 " << c[0] << endl << "2 " <<
c[1] << endl << "3 " << c[2] << endl << "4 " << c[3] <<endl;
      cin >> j;


        while(j < 1 || j > 4) {
          cout << "You entered wrong data. You can choose 1,2,3 or 4. Try
again. " << endl;
        cout << "Choose energy " <<  endl << "1 " << c[0] << endl << "2 " <<
c[1] << endl << "3 " << c[2] << endl << "4 " << c[3] <<endl;
        cin >> j;
    }
        z[2] = j;

    cout << "Choose transverse momentum" << endl << "1 " << d[0] << endl <<
"2 " << d[1] << endl << "3 " << d[2] << endl << "4 " << d[3]
    <<endl << "5 " << d[4] << endl;
       cin >> j;

        while(j < 1 || j > 5) {
          cout << "You entered wrong data. You can choose 1,2,3,4 or 5. Try
again. " << endl;
        cout << "Choose transverse momentum " <<  endl << "1 " << d[0] <<
endl << "2 " << d[1] << endl << "3 " << d[2] << endl << "4 " << d[3]
    <<endl << "5 " << d[4] << endl;
        cin >> j;
    }
        z[3] = j;

    if (z[3] == 1) {
```

```
    if( z[2] == 2 )
        ReadData("Experiment_for_ppnX_ISR_30GeV_pt0");

    if( z[2] == 4 )
        ReadData("Experiment_for_ppnX_ISR_53GeV_pt0");

        cout << "There are no data for transverse momentum equal to 0 " <<
endl;
        exit(1);
        }


     Char_t* f[4][4];

     f[0][0] = "Experiment_for_ppnX_ISR_22GeV_pt02";
     f[1][0] = "Experiment_for_ppnX_ISR_30GeV_pt02";
     f[2][0] = "Experiment_for_ppnX_ISR_44GeV_pt02";
     f[3][0] = "Experiment_for_ppnX_ISR_53GeV_pt02";
     f[1][1] = "Experiment_for_ppnX_ISR_30GeV_pt04";
     f[2][1] = "Experiment_for_ppnX_ISR_44GeV_pt04";
     f[3][1] = "Experiment_for_ppnX_ISR_53GeV_pt04";
     f[1][2] = "Experiment_for_ppnX_ISR_30GeV_pt06";
     f[2][2] = "Experiment_for_ppnX_ISR_44GeV_pt06";
     f[3][2] = "Experiment_for_ppnX_ISR_53GeV_pt06";
     f[1][3] = "Experiment_for_ppnX_ISR_30GeV_pt08";
     f[2][3] = "Experiment_for_ppnX_ISR_44GeV_pt08";
     f[3][3] = "Experiment_for_ppnX_ISR_53GeV_pt08";

         if(z[3] > 2 && z[2] == 1 ) {
        cout << "There are no data for energy equal to 23 " << endl;
        exit(1);
        }

     ReadData(f[z[2]-1][z[3]-2]);

}


//read data from ExpData.root

 void ReadData(Char_t* branchName) {

     ExpData_t Exp1Data;
    //cout<<branchName<<endl;
    // Opne ROOT file
    TFile* f = new TFile(gDir + "ExpData.root");

     TTree* tree = (TTree*)f->Get("T");

    Int_t num;
    //cout << Form("%s_Num", branchName);
    TBranch* b = tree->GetBranch(Form("%s_Num", branchName));
    b->SetAddress(&num);
    b->GetEntry(0);
    b = tree->GetBranch(branchName);
    b->SetAddress(&Exp1Data);
    //cout << num << endl;
    /*for(Int_t i = 0;i<num;++i)
    {
        b->GetEntry(i);
        //cout << i <<": " << Exp1Data.x << " " << Exp1Data.pt << " " <<
Exp1Data.f << " " << Exp1Data.deltaf << endl;
    }*/
```

```
    Double_t* x1 = new Double_t[num];
    Double_t* f1 = new Double_t[num];
    Double_t* deltax1 = new Double_t[num];
    Double_t* deltaf1 = new Double_t[num];


    for(Int_t i = 0;i < num;++i)
    {
        b->GetEntry(i);
        x1[i] = Exp1Data.x ;
        //cout<<x1[i]<<endl;
        f1[i] = Exp1Data.f;
        //cout<<f1[i]<<endl;
        deltaf1[i] = Exp1Data.deltaf;
        //cout<<deltaf1[i]<<endl;
        deltax1[i]=0;
        //cout<< deltax1[i] << endl;
        }

    TCanvas *c = new TCanvas("c", "c",14,45,700,500);
    TGraphErrors* graph1 = new TGraphErrors(num,x1,f1,deltax1,deltaf1);
    graph1->Draw("A*");
    c->SetLogy();
    }

void DrawMyTGraph() {

Char_t* branchName = "Experiment_for_ppnX_ISR_53GeV_pt04";
    TFile* f = new TFile(gDir + "ExpData.root");
     TTree* tree = (TTree*)f->Get("T");

    Int_t numEntries;
    //TBranch* b = tree->GetBranch("Experiment_for_ppnX_ISR_53GeV_pt0_Num");
    TBranch* b = tree->GetBranch(Form("%s_Num", branchName));
    b->SetAddress(&numEntries);
    b->GetEntry(0);

    ExpData_t Exp1Data;
    b = tree->GetBranch(branchName);
    b->SetAddress(&Exp1Data);


    //cout << num << endl;
    Double_t* x1 = new Double_t[numEntries];
    Double_t* f1 = new Double_t[numEntries];
    Double_t* deltax1 = new Double_t[numEntries];
    Double_t* deltaf1 = new Double_t[numEntries];

    Int_t i;

    for(i = 0;i<numEntries;++i)
    {
        b->GetEntry(i);
        x1[i] = Exp1Data.x ;
        //cout<<x1[i]<<endl;
        f1[i] = Exp1Data.f;
        //cout<<f1[i]<<endl;
        deltaf1[i] = Exp1Data.deltaf;
        //cout<<deltaf1[i]<<endl;
        deltax1[i]=0;
        //cout<< deltax1[i] << endl;
}

    TCanvas *c = new TCanvas("c", "c",14,45,700,500);
```

```
    TGraphErrors* graph1 = new
TGraphErrors(numEntries,x1,f1,deltax1,deltaf1);


    TF1 *MyFunc2  = new TF1("found", "RHOxfpt04(x)", 0.6, 0.95);
    TGraph* graph2 = new TGraph(MyFunc2);

    graph1->Draw("*A");
    graph2->Draw("same");
    c->SetLogy();
}
////////////////////////////////////////////////////////////////////////////

// RHOxfpt04_53GeV_07.04.2010.C
#include <iostream.h>
#include "TF1.h"
//
//----------Constants--------------------
Double_t PI=3.14159;
Double_t G2=183.4;
Double_t MN=0.94;
Double_t MPI=0.14;
Double_t MN2=MN*MN;
Double_t MPI2=MPI*MPI;
Double_t stotpiplusp=22.;
//
//-------Parameters-----------------------
Double_t T0=-0.7;
Double_t R12=0.3;
Double_t R22=0.74;
//
//----------Initial state-----------------
Double_t U;
U=52.8;
Double_t s;
s=U*U;
Double_t EA;
EA=U/2;
Double_t PA;
PA=sqrt(EA*EA-MN2);
//
//-----Defining function Q(x,y,z)(c.m.s momentum)------------
Double_t Q(Double_t x, Double_t y, Double_t z)
{
Double_t Q=sqrt((x-y-z)*(x-y-z)-4.*y*z)/(2.*sqrt(x));
return (Q);
}
//
//========Kinematics of final state=================================
//
//---- Defining function E (Energy of created particle)------------
Double_t E(Double_t xf, Double_t pt)
{
return (sqrt((xf*PA)*(xf*PA)+pt*pt+MN2));
}
//
//---- Defining function t (Square of 4-momentum transfer)---------
Double_t t(Double_t xf, Double_t pt)
{
return ((EA-E(xf,pt))*(EA-E(xf,pt))-(PA-xf*PA)*(PA-xf*PA)-pt*pt);
}
//
//----Defining function s1 (Invariant missing mass)-----------------
Double_t s1(Double_t xf,Double_t pt)
```

```
{
return (s+MN2-2.*U*E(xf,pt));
}
//
//---------------Defining c.m.s momenta of virtual pion-------------
Double_t ppivirt(Double_t xf, Double_t pt)
{
Double_t ppivirt=Q(s1(xf,pt),MN2,MPI2);
return(ppivirt);
}
//=====================================================================

//###############Constructing OPER cross section#####################

//_________________Building pion Green function_____________________

//--------Pion signature factor-----------------------------------------

Double_t Signfactpi(Double_t xf, Double_t pt)
{
return (PI/2./sin(PI/2.*(t(xf,pt)-MPI2)));
}
//-----------------Pion Green finction--------------------------------
//************ t >= T0 ******************************************

Double_t Greenpi(Double_t xf, Double_t pt)
{
if (t(xf,pt) >= T0)
  Double_t Greenpi=exp((R12+log(s/s1(xf,pt)))*(t(xf,pt)-
MPI2))*Signfactpi(xf,pt);
else
//************** t < T0 ************************************
Double_t Greenpi=exp((R12+log(s/s1(xf,pt)))*(t(xf,pt)-
T0))*(PI/2./sin(PI/2.*(T0-MPI2)))*exp(R22*(t(xf,pt)-T0));
return(Greenpi);
}
//Double_t Greenpi1(Double_t xf, Double_t pt)
//{
//Double_t Greenpi1=exp((R12+log(s/s1(xf,pt)))*(t(xf,pt)-MPI2))
//*Signfactpi(xf,pt);
//return(Greenpi1);
//}
//-------Virtual pion-nucleon scattering-------------------------



//*******Invariant Rho-function for pp->nX as function of xf and pt **
Double_t RHOxfpt(Double_t xf, Double_t pt)
{
Double_t RHOxfpt = 1.0/(pow(2.*PI,3)*PA*U)*G2*(-t(xf,
pt))*Greenpi(xf,pt)*Greenpi(xf,pt)*
ppivirt(xf,pt)*sqrt(s1(xf, pt))*stotpiplusp;

return(RHOxfpt);
}
//--------Invariant Rho-function for pp->nX as function of xf at pt=0.4


Double_t RHOxfpt04(Double_t xf)
{
Double_t RHOxfpt04 = RHOxfpt(xf, 0.4);
return(RHOxfpt04);
}
```

References

1. J. Engler et al. Nucl. Phys. B84 (1975) 70
2. V. Blobel et al.. Nucl. Phys. B135 (1978) 379
3. W.Flauger, J.Monning. Nucl. Phys. B109 (1976) 347
4. R.Robinson et al. Phys. Rev. Lett. 34 (1975) 1475
5. J.Hanlon et al. Phys. Rev. Lett. 37 (1976) 967
6. Y.Eisenberg et al. Nucl. Phys. B135 (1978) 189
7. I.V. Azhinenko et al.. Yad. Fiz. 31(1980) 956
8. P.D.Higgins et al. Phys. Rev. D19, (1979) 731
9. F.T.Dao et al. Phys. Rev. Lett. 37 (1976) 967
10. M.R.Whalley et al., Preprint UM-HE 79149 (1979)
11. A.A. Grigoryan. Preprint ITEP-112 (1975)
12. K.G.Boreskov, A.A.Grigoryan and A.B.Kaidalov, Sov. J. Nucl. Phys. 24 (1976) 411
13. K.G.Boreskov, A.A.Grigoryan, A.B.Kaidalov and I.I.Levintov, Sov. J. Nucl. Phys. 27 (1978) 813
14. G.H.Arakelyan, A.A.Grigoryan. Sov. J. Nucl. Phys. 36 (1982) 211
15. K.G.Boreskov, A.B.Kaidalov and L.A.Ponomarev, Sov. J. Nucl. Phys. 19 (1974) 565
16. D.Varga et al. NA49 collaboration. , Euro. Phys. J. C33, s01, s515 (2004)
17. T.Anticic et al. NA49 collaboration. CERN-PH-EP/2009-006; hep-ex/0904.2708, 2009
18. M.Togawa, for RHIC-PHENIX collaboration. Talk, presented to Diffraction2008; PhD Theses, Kyoto University, 2008.
19. M.Derric et al. ZEUS collaboration, Phys. Lett. B384 (1996) 338
20. S.Chekanov et al. ZEUS collaboration, Phys. Lett. B610 (2005) 199
21. C.Adolff et al. H1 collaboration, Eur. Phys.J. C6, (1999) 587
22. A.Aktas et al. H1 collaboration, Eur. Phys.J. C41, (2005) 273
23. A.B.Kaidalov,. V.A.Khoze, A.D.Martin and M.G.Ryskin. Euro, Phys. J. C 47, 385 (2006); hep-ph/0602215
24. ROOT framework: http://root.cern.ch
25. CERN main page: http://public.web.cern.ch/public/