Spin Alignment of Vector Mesons and Angular Momentum in high energy collisions

Viraj Thakkar*

National Institute of Science Education and Research

(Dated: April 24,2017)

Abstract

In this report, we have calculated the angular momentum for ultra-relativistic collisions using Optical and Monte Carlo Glauber Model for Au-Au collsion at $\sqrt{s_{NN}} = 200\ GeV/c^2$. The angular distribution formula for spin alignment of Vector Mesons is derived. From the data analysis of simulated pp collisions at 7 TeV, the spin density matrix element ρ_{00} was calculated by fitting $dN/dcos\,\theta$ vs $cos\,\theta$. The invariant mass of Vector Meson K^* was also calculated.

CONTENTS

ANGULAR MOMENTUM IN HEAVY ION COLLISONS

When heavy ions collide at ultra-relativistic speeds, the overlapping region of the colliding nucleus leads to QGP. A large part of initial angular momentum is carried away by spectator fragments while a fraction of it is left to the interaction region. We want to calculate Angular Momentum of the interacting region as a function of impact parameter using Glauber Model.

OPTICAL GLAUBER MODEL CALCULATION OF ANGULAR MOMENTUM

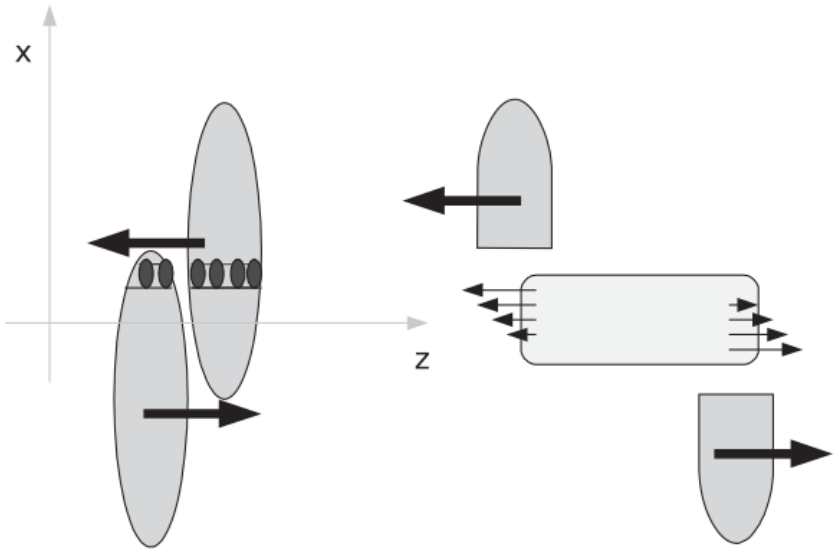

FIG. 1. The geometrical view of nuclear collision . As we can see from the diagram, a large part of initial angular momentum is carried by the non-interacting fragment particles.[3]

The net momentum density per unit area at the point (x, y) in the overlap region is given by

$$\frac{dP}{dxdy} = [T(x - b/2, y) - T(x + b/2, y)]\frac{\sqrt{s_{NN}}}{2} \tag{1}$$

where $T(x, y)$ is the thickness function defined by :

$$T(x, y) = \int dz\, n(x, y, z) \tag{2}$$

where $n(x, y, z)$ is the probability density function to find a nucleon at (x, y, z)
Considering a hard sphere model for the nuclear density.

$$n(r) = \begin{cases} n_o & \text{if } r < R \\ 0 & \text{if } r \geq R \text{ where } R \text{ is radius of nucleus} \end{cases}$$

We have angular momentum of the interacting region along the y axis as[3]:

$$\vec{J}(b) = \iint dxdy\, x\, [T(x - b/2, y) - T(x + b/2, y)] \frac{\sqrt{s_{NN}}}{2} \hat{j} \tag{3}$$

For hard sphere model, as $T(x, y)$ is proportional to $\sqrt{R^2 - r^2}$ (r is distance from center of the nucleus), we get the following equation :

$$J(b) = \iint 2\, n_o dxdy\, x\, [\sqrt{R^2 - y^2 - (x - b/2)^2} - \sqrt{R^2 - y^2 - (x + b/2)^2}] \frac{\sqrt{s_{NN}}}{2} \tag{4}$$

We can calculate this integral numerically after performing the integral with appropriate limits in the overlapping region.2

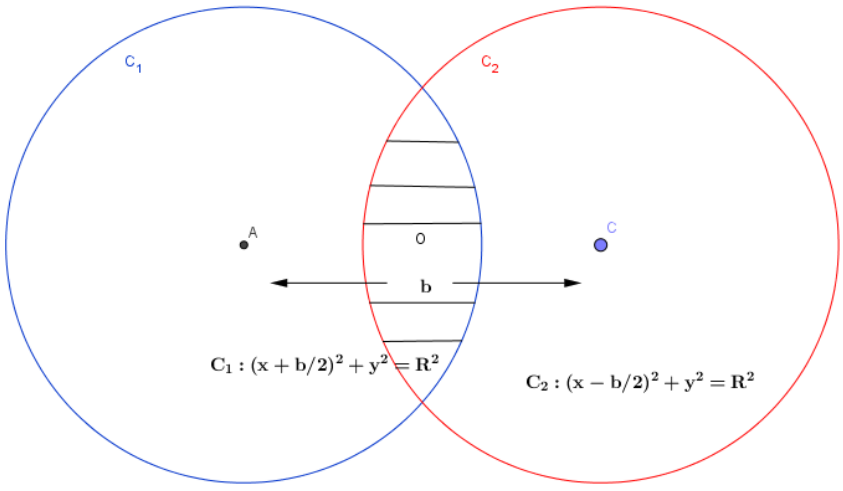

FIG. 2. Overlapping region subjected to constraints by the curve C_1 and C_2 as shown in the figure. Thus, for a given y: the limits of x will be $-\sqrt{R^2 - y^2} + b/2$ to $\sqrt{R^2 - y^2} - b/2$. y varies from $-\sqrt{R^2 - b^2/4}$ to $\sqrt{R^2 - b^2/4}$

MONTE CARLO GLAUBER MODEL CALCULATION OF ANGULAR MOMENTUM

The Angular Momentum for Monte Carlo Glauber Model was calculated using position and momenta of participating nucleons (using simulated data) in the overlapping region for various values of impact parameter (b).

$$\vec{\mathbf{L}} = \vec{\mathbf{r}} \times \vec{\mathbf{P}}$$

Consider the first nucleus traveling along the $+z$ axis. Then the nucleons in the first nucleus will have momentum of $+P_z$ In terms of component , we write

$$\vec{L}_x = yP_z\hat{i} - xP_z\hat{j}$$
$$\vec{L}_y = yP_z\hat{i} - xP_z\hat{j}$$

The net angular momentum will be given by

$$L = \sqrt{L_x^2 + L_y^2}$$

Similarly we calculate for the other nucleon with P_z replaced by $-P_z$.
Summing the angular momentum over all the participating nucleons, we can calculate the net angular momentum for varying impact parameter (b). Where impact parameter b is the perpendicular distance between the center of two colliding nucleus.

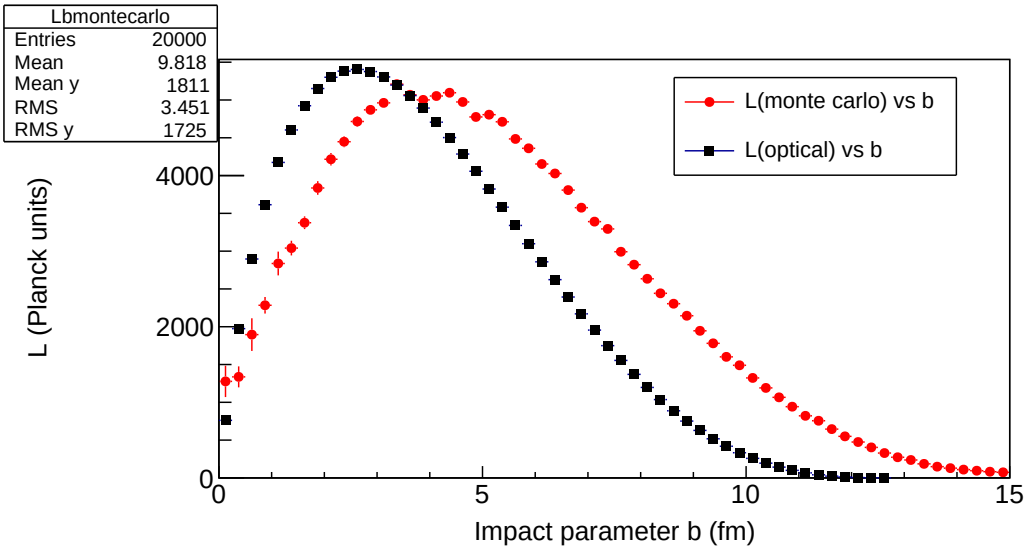

FIG. 3. Angular Momentum comparison obtained from Optical Glauber Model (for hard sphere) and Monte Carlo Glauber Model for Au-Au collsion at $\sqrt{s_{NN}} = 200\ GeV/c^2$.Thus we observe that there is large angular momentum of the interaction region.The plots are comparable by both the methods.

ANGULAR DISTRIBUTION FORMULA FOR VECTOR MESONS

Spin Alignment

Due to large angular momentum of the overlapping region of the nucleus, there is spin-orbit coupling effect which leads to spin alignment of vector mesons produced in nucleon-nucleon collision. This leads to deviation in the value of spin density matrix element ρ_{00} from 1/3 .

Event reaction

$$\gamma + \mathbf{N} \rightarrow \ \mathbf{V} + \mathbf{N} \tag{5}$$

Where γ is polarized photon, N is nucleon and V is Vector Meson. The Vector meson further decomposed into resonance particles.For example,

$$\mathbf{K}^* \rightarrow \mathbf{K}^+ + \pi^- \tag{6}$$

Notations:

CMS frame:

The four momenta of incoming photon and outgoing vector meson is denoted as k and q respectively.

We define the coordinate system by using the 3-momentum vectors ($\vec{k}$ and $\vec{q}$) as follows:

$$\hat{Z} = \frac{\vec{k}}{|\vec{k}|} \tag{7}$$

$$\hat{Y} = \frac{\vec{k} \times \vec{q}}{|\vec{k} \times \vec{q}|} \tag{8}$$

$$\hat{X} = \frac{(\vec{k} \times \vec{q}) \times \vec{k}}{|(\vec{k} \times \vec{q}) \times \vec{k}|} \tag{9}$$

Production Plane:

The plane produced by vectors $\vec{k}$ and $\vec{q}$ is called production plane. The normal to production plane is given by $\vec{k} \times \vec{q}$

Helicity frame:

In the helicity frame, V is at rest . z direction is chosen opposite to direction of outgoing nucleon. The y direction is along the normal to the production plane. The x direction is then given by $\hat{x} = \hat{y} \times \hat{z}$ Ref Fig.4

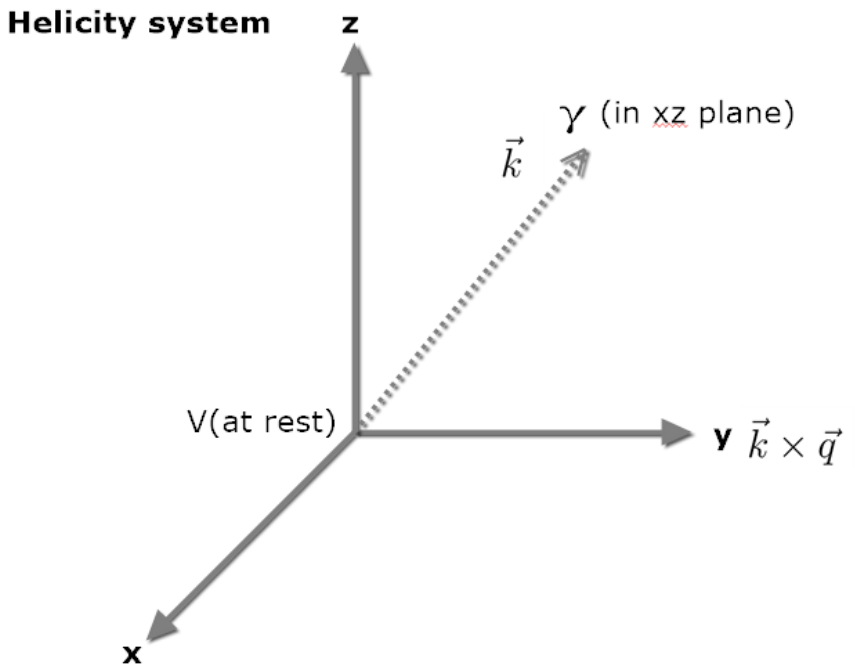

FIG. 4. Helicity System. The photon is in xz plane. Y axis is perpendicular to the production plane. V is at rest .

Decay angles: The Vector meson further decays into other particles as shown in Eq.6.

The polar and azimuthal angles are as shown is Fig.5. The vector $\hat{\pi}$ denotes the decay direction of the particle formed by the Vector Meson decay in the Helicity frame. We will consider two particle decay of Vector meson for this context. In general $\hat{\pi}$ in spherical coordinates is given by

$$\hat{\pi} = sin\,\theta\, cos\,\phi\,\hat{x} + sin\,\theta\, sin\,\phi\,\hat{y} + cos\,\theta\,\hat{z}$$

Then it can be easily seen that

$$cos\,\theta = \hat{\pi}.\hat{z}$$
$$cos\,\phi = \hat{y}.(\hat{z} \times \hat{\pi})$$
$$sin\,\phi = -\hat{x}.(\hat{z} \times \hat{\pi})$$

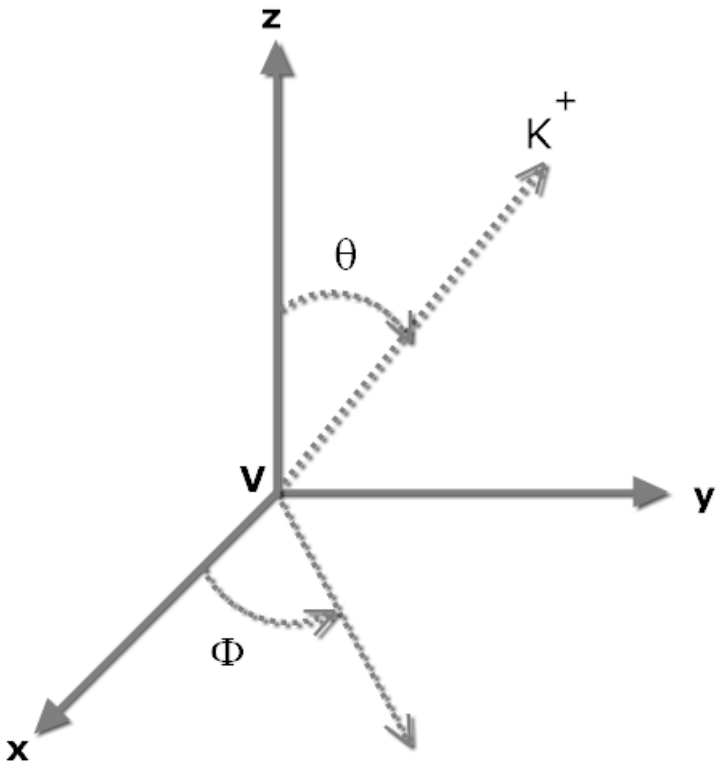

FIG. 5. Decay Distribution angles made by daughter particle in the rest frame of the mother Vector Meson.

Derivation of Formula

The scattering cross section is given by

$$\frac{dN}{dcos\,\theta\,d\phi} = W(cos\,\theta, \phi) = M\rho(V)M^{+} \tag{10}$$

Where M is the decay amplitude.

$\rho(V)$ is the spin space density matrix of the vector meson.

Now inserting a complete set of basis in terms of helicity of vector meson ($|\lambda_V>$), we get

$$\frac{dN}{dcos\,\theta\,d\phi} = \sum_{\lambda_V \lambda_V^{'}} <\theta, \phi|M|\lambda_V> \rho(V)_{\lambda_V \lambda_V^{'}} <\lambda_V^{'}|M^{+}|\theta, \phi> \tag{11}$$

Using

$$<\theta, \phi|M|\lambda_V> = C\sqrt{\frac{3}{4\pi}}\, D^{1^*}_{\lambda_V 0}(\phi, \theta, -\phi) \tag{12}$$

Where C is a constant and is set to 1 for normalized decay distributions.

D are the Wigner Rotation Functions .

Using Eq.12, The formula becomes

$$W(cos\,\theta,\phi) = \frac{3}{4\pi}\sum_{\lambda_V\lambda_V'} D^{1^*}_{\lambda_V 0}(\phi,\theta,-\phi)\,\rho(V)_{\lambda_V\lambda_V'}\,D^{1}_{\lambda_V' 0}(\phi,\theta,-\phi) \tag{13}$$

Eigenvalues for Helicity are $\lambda_V = -1, 0, +1$. Thus, expanding the summation, we get

$$= \frac{3}{4\pi}\Bigg[\sum_{\lambda_V}\left(D^{1^*}_{\lambda_V 0}(\phi,\theta,-\phi)\,\rho(V)_{\lambda_V 1}D^{1}_{10}(\phi,\theta,-\phi)\right) + \left(D^{1^*}_{\lambda_V 0}(\phi,\theta,-\phi)\,\rho(V)_{\lambda_V 0}D^{1}_{00}(\phi,\theta,-\phi)\right) + \left(D^{1^*}_{\lambda_V 0}(\phi,\theta,-\phi)\,\rho(V)_{\lambda_V -1}D^{1}_{-10}(\phi,\theta,-\phi)\right)\Bigg]$$

$$= \frac{3}{4\pi}\left[\left(D^{1^*}_{10}\rho_{11}D^{1}_{10} + D^{1^*}_{00}\rho_{01}D^{1}_{10} + D^{1^*}_{-10}\rho_{-10}D^{1}_{10}\right) + \left(D^{1^*}_{10}\rho_{10}D^{1}_{00} + D^{1^*}_{00}\rho_{00}D^{1}_{00} + D^{1^*}_{-10}\rho_{-10}D^{1}_{00}\right) + \left(D^{1^*}_{10}\rho_{1-1}D^{1}_{-10} + D^{1^*}_{00}\rho_{0-1}D^{1}_{-10} + D^{1^*}_{-10}\rho_{-1-1}D^{1}_{-10}\right)\right]$$

Using:

$$D^{1}_{10}(\phi,\theta,-\phi) = -\frac{1}{\sqrt{2}}sin\,\theta e^{-i\phi}$$
$$D^{1}_{00}(\phi,\theta,-\phi) = cos\,\theta$$
$$D^{1}_{-10}(\phi,\theta,-\phi) = \frac{1}{\sqrt{2}}sin\,\theta e^{i\phi}$$

Substituting, we get:

$$= \frac{3}{4\pi}\Bigg[\left(\frac{1}{2}sin^2\,\theta\rho_{11} - \frac{1}{\sqrt{2}}sin\,\theta\,cos\,\theta e^{i\phi}\rho_{10} - \frac{1}{2}e^{-i2\phi}sin^2\theta\rho_{-10}\right) + \left(-\frac{1}{\sqrt{2}}sin\,\theta cos\,\theta e^{i\phi} + \rho_{00}cos^2\,\theta + \frac{1}{\sqrt{2}}sin\,\theta cos\,\theta e^{-i\theta}\rho_{-10}\right) + \left(-\frac{1}{2}e^{i2\phi}sin^2\,\theta\rho_{1-1} + \frac{1}{\sqrt{2}}sin\,\theta cos\,\theta e^{i\theta}\rho_{0-1} + \frac{1}{2}sin^2\,\theta\rho_{-1-1}\right)\Bigg]$$

Now integrating over ϕ from 0 to 2π , and using

$$\int_0^{2\pi} e^{i\phi}d\phi = 0 \text{ and } \int_0^{2\pi} e^{i2\phi}d\phi = 0$$

$$\frac{dN}{dcos\,\theta} = \frac{3}{2}\left(\frac{1}{2}sin^2\theta(\rho_{11} + \rho_{-1-1}) + \rho_{oo}cos^2\theta\right)$$

Using the condition,

$$\sum_n \rho_{nn} = \rho_{11} + \rho_{00} + \rho_{-1-1} = 1$$

We get(Reference paper[1]),

$$W(\theta) = \frac{3}{4}\left[(1-\rho_{00}) + (3\rho_{00}-1)cos^2\,\theta\right] \tag{14}$$

Thus, when there is no Spin Alignment , $\rho_{00} = 1/3$ and the angular distribution is independent of θ .

RESONANCES:

Resonance particles or resonances are extremely short lived particles. The lifetime of these particles is on the order of 10^{-23} s . They almost travel at the speed of light and thus could travel only about 10^{-15} meters or about the diameter of proton. Hence it is very difficult to predict the occurrence of such particles.
Thus we measure the scattering cross section of particles. If we make a plot of the cross-section of the particles versus the total energy of the particles , we observe that the graphs have peaks and valleys.This implies that the cross-section of the colliding particle changes as a function of the total energy in the collision. When graphs of the different possible outcomes of the same collision are compared, we find that the peak cross-sections occur at the same energies for each possibility.
An explanation for this is that the peaks are evidence for actual particles that are formed as intermediate steps in the collision. Hence, the presence of resonance particles adds to the cross-section of the particles in the collision thus making it more likely. Thus, the peaks are interpreted as the presence of different resonance particles each with different decay lifetime. Thus, we can calculate energy or mass of the resonance particle by locating the peak in the scattering cross section curve.

NON- RELATIVISTIC BREIT WIGNER FUNCTION

We will derive the Non- Relativistic Breit Wigner function which is used to model resonance curves.

According to Fermi's Golden Rule from Quantum Mechanics, the transition rate W from

one inital state $|i\rangle$ to some final state $|f\rangle$ is given by :

$$W = \frac{2\pi}{\hbar}|M_{if}|^2\,\rho_f(E)_{E\approx E_i} \tag{15}$$

where $M_{if} = \langle f|\,U\,|i\rangle = \int\,\psi_f^* U\psi_i\,dV$, U is the interaction perturbation operator .

The Energy density of final states is given by:

$$\rho_f(E) = \frac{dN}{dE}$$

That is the number of states available to product particles per unit interval of Energy. We know;

$$\text{Mean lifetime} = \frac{1}{\text{Rate}}$$

Hence we define mean lifetime of a decaying state as:

$$\tau = \frac{1}{W} \tag{16}$$

Now according to uncertainty principle, the Energy width Γ follows $\Gamma\tau \sim \hbar$. Hence we define

$$\Gamma = \frac{\hbar}{\tau} = \hbar W = 2\pi|M_{if}|^2\rho_f \tag{17}$$

For an exponential decay of particle A,

$$N_A(t) = N_A(0)exp\left(-\frac{\Gamma}{\hbar}t\right) \tag{18}$$

where $N_A(t)$ are the number of particles at time t , $N_A(0)$ are the initial number of particles.

Now let us assume the wavefunction of non-stationary state of energy $E_R(=\omega_R\hbar)$, where E_R is the resonance energy.

$$\psi(t) = \psi(0)e^{-i\omega_R t}f(t)$$

As $\langle\psi(t)\,|\,\psi(t)\rangle \sim e^{-\frac{\Gamma}{\hbar}t}$ or $\langle\psi(t)\,|\,\psi(t)\rangle \sim e^{-\frac{t}{\tau}}$,we conclude that $f(t) \sim e^{-\frac{t}{2\tau}}$

Hence,

$$\psi(t) = \psi(0)e^{-it\omega_R}e^{-\frac{t}{2\tau}}$$

In natural units, $\hbar = c = 1$,

$$\psi(t) = \psi(0)e^{-t\left(iE_R+\frac{\Gamma}{2}\right)}$$

To find the probability of amplitude of state with energy E, we take the Fourier Transform

$$g(\omega) = \int_0^\infty \psi(t) e^{i\omega t} dt \ \ , \omega = E/\hbar$$

Hence we get amplitude as a function of E,

$$g(E) = \int_0^\infty \psi(0) e^{-t\left(i(E_R - E) + \frac{\Gamma}{2}\right)} dt$$

After integrating and putting the limits ,we get

$$g(E) = \frac{k}{\frac{\Gamma}{2} + i(E_R - E)} \text{ for some constant } k$$

The cross section $\sigma(E)$ follows

$$\begin{aligned} \sigma(E) &= g^*(E) g(E) \\ &= \frac{1}{(E_R - E)^2 + \frac{\Gamma^2}{4}} \end{aligned}$$

Hence,

$$\sigma(E) = \sigma_{max} \frac{\frac{\Gamma^2}{4}}{(E - E_R)^2 + \frac{\Gamma^2}{4}} \tag{19}$$

with E_R the resonant energy and Γ is the Full Width at Half Maximum. Thus, at $E = E_R$, $\sigma(E) = \sigma_{max}$

CHECK FOR SPIN ALIGNMENT IN PP COLLISIONS AT 7 TEV FROM SIMULATED DATA.

We have been given data corresponding to Energy and momentum of primary daughter particles formed as a result of decay of Vector Mesons . We have to perform the following task :

1. Calculate invariant mass and $cos\,\theta$ for each like pair of unlike sign ($K^+\pi^-$ and $K^-\pi^+$) and like sign ($K^+\pi^+$ and $K^-\pi^-$) daughter particles. To calculate the angle between one of the daughter particle and the z axis, the daughter particle must be boosted to the rest frame of the mother Vector Meson. Make 2D scatter plot of Invariant Mass vs $cos\theta$ for all such pairs.

2. For each $cos\theta$ bin, Make 1D Histogram projections to get Number of counts vs Invariant Mass.

3. Signal Extraction by using like sign technique to eliminate background .

4. Fitting functions to the resonance signals obtained and calculating the yield for each $cos\theta$ bin.

5. Plot $dN/dcos\theta$ vs $cos\theta$ and obtain the value of spin density matrix element ρ_{00}

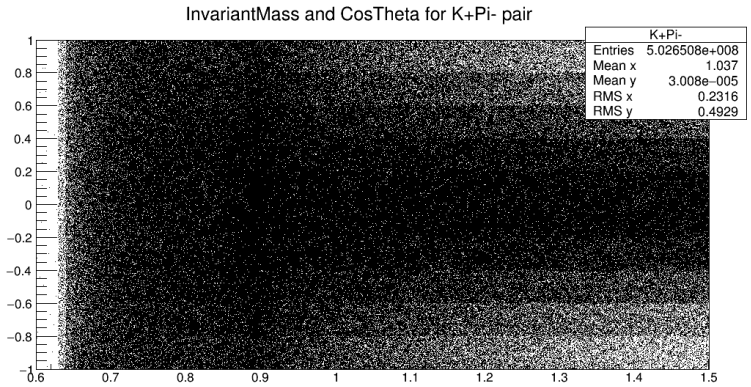

FIG. 6. Invariant mass vs $cos\theta$ 2D scatter plot for unlike sign $K^+\pi^-$

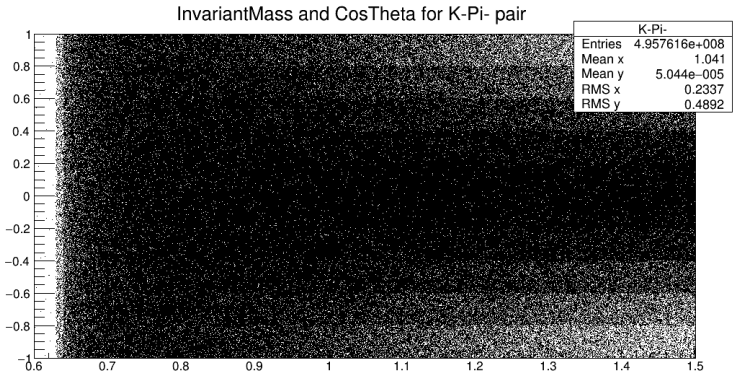

FIG. 7. Invariant mass vs $cos\theta$ 2D scatter plot for like sign $K^-\pi^-$

Signal Extraction and Resonance

The signal from unlike sign pairs $K^+\pi^-$ & $K^-\pi^+$ is added together to make unlike pair signal as shown is Fig.8. The background signal (obtained from $K^+\pi^+$ and $K^-\pi^-$ pairs) is combined using the like sign technique.

$$N_{Like-sign}(m) = 2 \times \sqrt{N_{K^+\pi^+}(m) \times N_{K^-\pi^-}(m)} \tag{20}$$

Where N is the number of entries in a $cos\theta$ bin with its center at the $K\pi$ pair invariant mass m. The final true signal is obtained by subtracting the background signal from the total unlike pair signal.[4]

$$N_{K^{*0}}(m) = N_{K^+\pi^-}(m) + N_{K^-\pi^+}(m) - 2 \times \sqrt{N_{K^+\pi^+}(m) \times N_{K^-\pi^-}(m)} \tag{21}$$

After getting the true signal, we observer a resonance peak corresponding to K^* See Fig.9 . This resonance curve is fitted with a non-relativistic Breit Wigner function plus a polynomial of degree 2. The degree 2 polynomial is added to take care of remaining non accountable background left. Finally, yield is obtained from the fit parameters from the function for each $cos\theta$ bin.

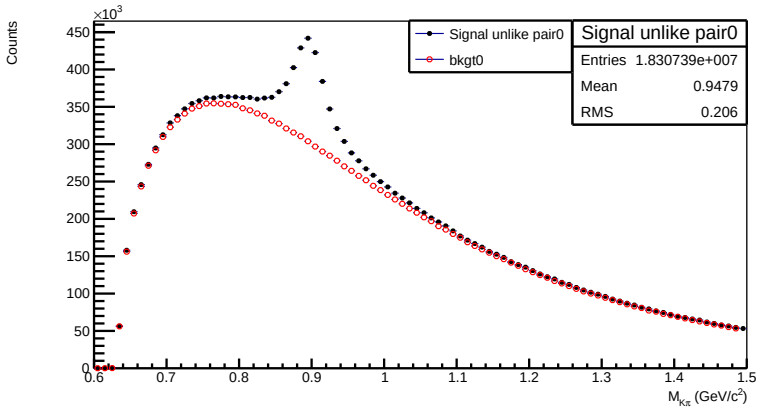

FIG. 8. Signal Extracted from the first $cos\theta$ bin.The figure shows the number of counts of the unlike pair combinations(black): $K^+\pi^-$ & $K^-\pi^+$ along with total background signal(red). The peak can clearly be seen around K^* invariant mass of 0.896 GeV/c^2. The total background is obtained using the like-sign technique.

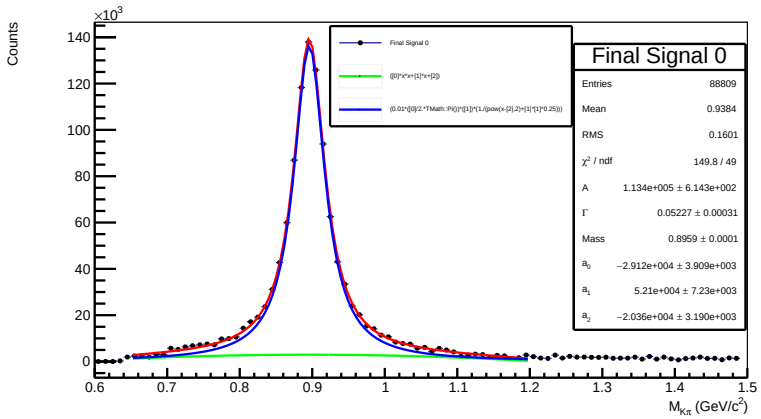

FIG. 9. Final signal obtained after background substraction for the first $cos\theta$ bin. The final signal is fitted with a non-relativistic Briet Wigner function plus a polynomial of degree 2. The yield is obtained from the fit parameters. Black represents the final signal. Blue curve represents the Breit Wigner Fuction and Green curve represents the degree 2 polynomial background.

Density matrix element and Invariant Mass Calculation

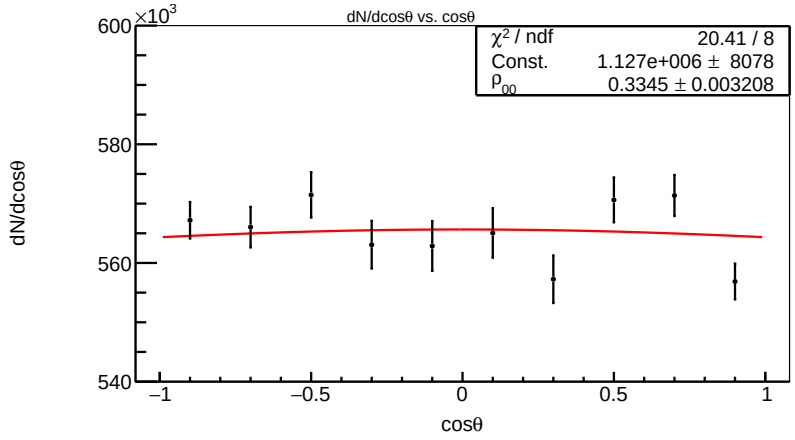

FIG. 10. The yield obtained from the fit parameters for all the signals is plotted against the respective $cos\theta$ bin midpoint. The angular distribution function14 is fitted to it and ρ_{00} matrix element is calculated. $\rho_{00} \approx \frac{1}{3}$

This tells us that there is no spin alignment along any direction.

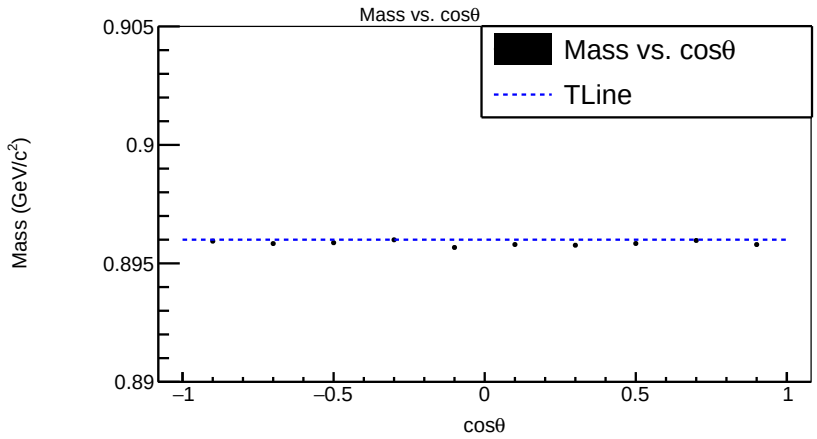

FIG. 11. The yield obtained from the pit parameters for all the signals is plotted against it's respective $cos\theta$ bin midpoint. The invariant mass of mother vector meson (K^*) is 0.896 GeV/c^2

.

RESULTS AND CONCLUSION

- The invariant Mass of K^* was obtained to be around 0.896 GeV/c^2
- $\rho_{00} \approx 1/3$ which is in agreement with the no spin alignment value as pp collision does not have any angular momentum.
- The angular momentum for ultra-relativistic collisions was calculated using Glauber Model. The plots given by Optical and Monte Carlo Glauber Model are comparable.

ACKNOWLEDGMENTS

Supervisor:

Prof. Bedangadas Mohanty

Thanks to :

Mr. Ajay Dash and Mr. Vipul Bairathi

APPENDIX

Program for Finding Angular Momentum by Optical Glauber Model

```
#include <iostream>
#include <iomanip>
#include<cmath>
#include<fstream>
using namespace std;

//function prototype
double fun(double ,double );
double lim(double,double);
//double fune(double,double);
int main()
{



// cout<<"function is value is "<<fun(1,3)<<endl;
//float a,b;
const double R=6.38;
cout<<"The Radius of Gold is "<<R<<" fm "<<endl;
//cout<<"Enter the value of the radius of nucleus in fm :"<<endl;
//cin>>R;
//cout<<"Enter the value of impact parameter b (0<t<2R) = "<<endl;
//cin>>b;



//cout<<"The limits of integration are from "<<-lim(R,t)<<" to "<<lim(R,t)<<endl;
//   const float PI = 3.14159F; //type const float
unsigned long int n;
n=1000;
```

```
// cout<<"Enter the interval starting point a :"<<endl;
//cin>>a;
//cout<<"Enter the interval starting point b :"<<endl;
//cin>>b;
// cout<<"Enter the number of sub-intervals n (must be even) to integrate :"
    <<endl;
//cin>>n;
//float np=(n/2);

//cout<<"fun(t,0)="<< fun(t,0)<<endl;
//cout<<"limit = "<< lim(R,t)<<endl;




/* for loop for function values:
double y1;
for(int l=0;l<=n;l++)
{
y1=a+l*h;
cout<<setw(6)<<y1<<setw(12)<<fun(t,y1)<<endl;
}
*/
ofstream file;

file.open("ang mom gold optical.txt");
file.precision(10);
//file.setf(ios::scientific);
file.setf(ios::showpoint);
```

```
ofstream avbop;

avbop.open("A vs b optical Au.txt");
avbop.precision(10);
//file.setf(ios::scientific);
avbop.setf(ios::showpoint);
cout<<setw(20)<<"t=b/2R"<<setw(20)<<" b (fm) "<<setw(25)<<" Angular momentum
    (Gev) "<<endl;
file<<setw(20)<<"t=b/2R"<<setw(20)<<" b (fm) "<<setw(25)<<" Angular momentum
    (Gev) "<<endl;

double b=0;
double Nb=20000;
for(int u=0;u<=Nb;u++)
{b=(0+u*(1/Nb))*2*R;
double h=(lim(R,b))/n;
//cout<<"length of subinterval is h= "<<h<<endl;
//t=b/(2*R);

double a,c;
//a=-lim(R,t);
a=-lim(R,b);
c=lim(R,b);
//cout<<"limits are a to c : "<<a<<" to "<<c<<endl;

// sum of function value at end points
double send;
//send=fun(b,a)+fun(b,c); //dont use this, the angular momentum function fetches
    zero in denominator at end points
send=0;
```

```
//cout<<"sum end (send)="<<send<<endl;
//sum of function value sum at even points:
double seven=0;
double x1;
for(int k=2;k<=n;k=k+2)
{
x1=0+(k-1)*h;
seven=seven+fun(b,x1);
}
//cout<<"sum of even terms is (seven)= "<<seven<<endl;

//sum of function value at odd points;
double sodd=0;
double x2;
for(int j=3;j<n;j=j+2)
{
x2=0+(j-1)*h;
sodd=sodd+fun(b,x2) ;
}
//cout<<"sum of odd terms is (sodd) ="<<sodd<<endl;

double Is;
Is=200*(h/3)*( 4*seven + 2*sodd)*(3/(4*3.14159))*(1/pow(R,3));
cout<<setw(20)<<b/(2*R)<<setw(20)<<b<<setw(25)<<Is*1000/6.52485<<endl;
file<<setw(20)<<b/(2*R)<<setw(20)<<b<<setw(25)<<Is*1000/6.52485<<endl;
avbop<<setw(25)<<b<<setw(25)<<Is*1000/6.52485<<endl;
}
file.close();
avbop.close();
return 0;
```

```
}

//function declaration

double lim(double R,double b)
{
double limit;
limit=sqrt(R*R- (b*b/4) );
return limit;
}
double fun(double b,double y)
{  //const double PI = 3.14159265358979323846264338327950288419 7;
const double PI=3.14159;
const double R=6.38;
double a;
a=sqrt(R*R-y*y);

double fn;
//fn= N1(t,x)*N2(t,x)+ (1-x*x)*(0.5*PI- atan(N1(t,x)/N(t,x)));
// fn=-(2*t-sqrt(1-pow(x,2)))*sqrt(4*t*sqrt(1-(x*x))-4*t*t) + (1-x*x)*(PI*0.5 +
    atan((2*t-sqrt(1-pow(x,2)))/sqrt(4*t*sqrt(1-(x*x))-4*t*t)));
// fn=x*x*t;
double t1=(sqrt(2*a*b-b*b))*( 2*b-10*a );
//double t1= ( sqrt( abs( 2*a*b-b*b )*( 2*b-10*a ) )    );
double t3=atan( (b-a)/( sqrt(2*a*b-b*b) )      );
//double t3= atan(  ( b-a )/ ( sqrt( abs( 2*a*b-b*b ) ) )    );
fn=(b/12)*( t1 + 3*a*a*PI -6*a*a*t3 );
//-6*a*a*t3
// fn=atan((2*t-sqrt(1-pow(x,2)))/sqrt(4*t*sqrt(1-(x*x))-4*t*t));
//fn=(2*t-sqrt(1-pow(x,2)))*sqrt(4*t*sqrt(1-(x*x))-4*t*t);
```

```
//fn=sqrt(1-pow(x,2));
// fn=-(2*t-sqrt(1-pow(x,2)))*sqrt(fabs(4*t*sqrt(1-(x*x))-4*t*t)) +
    (1-x*x)*(PI*0.5 +
    atan((2*t-sqrt(1-pow(x,2)))/sqrt(fabs(4*t*sqrt(1-(x*x))-4*t*t))));
return fn;
}
```

Program for Finding Angular Momentum by Monte Carlo Glauber Model

C file:

```
#define AngularMomAna_cxx
#include "AngularMomAna.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void AngularMomAna::Loop()
{
//Analysis Variables
Float_t En = 200/2.0; //in GeV units
Float_t Mn = 0.938; //in GeV units //assuming Mp = Mn = 0.938 GeV
Float_t Pn = TMath::Sqrt(En*En - Mn*Mn); //in Gev Units //Momentum of colliding
    nucleons(along z axis)
TFile *fout=new TFile("Angularmomvsb.root","recreate");
TProfile *Lxb=new TProfile("Lxb","Lx vs b",60,0,15);
TProfile *Lyb=new TProfile("Lyb","Ly vs b",60,0,15);
TProfile *Lb=new TProfile("Lb","L vs b",60,0,15);
TProfile *modLxb=new TProfile("|Lx|","|Lx| vs b",60,0,15);
TProfile *modLyb=new TProfile("|Ly|","|Ly| vs b",60,0,15);
//   In a ROOT session, you can do:
```

```
//      Root > .L AngularMomAna.C
//      Root > AngularMomAna t
//      Root > t.Loop();     // Loop on all entries
//

//    This is the loop skeleton where:
//   jentry is the global entry number in the chain
//   ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//   jentry for TChain::GetEntry
//   ientry for TTree::GetEntry and TBranch::GetEntry
//
//      To read only selected branches, Insert statements like:
// METHOD1:
//   fChain->SetBranchStatus("*",0); // disable all branches
//   fChain->SetBranchStatus("branchname",1); // activate branchname
// METHOD2: replace line
//   fChain->GetEntry(jentry);    //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
if (fChain == 0) return;
Long64_t nentries = fChain->GetEntriesFast();
Long64_t nbytes = 0, nb = 0;
ofstream fang;

fang.open("ang mom gold distribution root.txt");
fang.precision(10);
//file.setf(ios::scientific);
fang.setf(ios::showpoint);

ofstream avsb;
```

```
avsb.open("L vs b Au.txt");
avsb.precision(10);
//file.setf(ios::scientific);
avsb.setf(ios::showpoint);

//Event Loop
for (Long64_t jentry=0; jentry<nentries; jentry++)
{

//if(jentry > 10) break;

Long64_t ientry = LoadTree(jentry);
if (ientry < 0) break;
nb = fChain->GetEntry(jentry); nbytes += nb;
// if (Cut(ientry) < 0) continue;

Float_t Lx = 0.;
Float_t Ly = 0.;

for(int i=0; i<394; i++)
{
if(Stat[i] == 0) continue;
if(i < 197)
{
Lx = Lx + (Ny[i] * Pn)* (1.0/6.52485);
Ly = Ly + ((-1.0)*Nx[i] * Pn) *(1.0/6.52485);
}
else
{
```

```
Lx = Lx + (Ny[i] * (-1.0)*Pn) * (1.0/6.52485);
Ly = Ly + ((-1.0)*Nx[i] * (-1.0)*Pn)*(1.0/6.52485);
}
//cout<<"Nxy: "<<Nx[i]<<", "<<Ny[i]<<endl;

}

/*
Float_t L=0;
for(int i=0; i<394; i++)
{
if(Stat[i] == 0) continue;
if(i < 197)
{
if(Nx[i]>0)
{
L =L+ sqrt(Nx[i]*Nx[i]+ Ny[i]*Ny[i])*(-Pn)/2;
}
else
{
L =L+ sqrt(Nx[i]*Nx[i]+ Ny[i]*Ny[i])*(Pn/2);
}
}
else
{
// Lx = Lx + (Ny[i] * (-1.0)*Pn); // * (1.0/0.1975);
//  Ly = Ly + ((-1.0)*Nx[i] * (-1.0)*Pn); // *(1.0/0.1975);
if(Nx[i]>0)
{
L =L + sqrt(Nx[i]*Nx[i]+ Ny[i]*Ny[i])*(Pn)/2;
```

```
}
else
{
L =L+ sqrt(Nx[i]*Nx[i]+ Ny[i]*Ny[i])*(-Pn/2);
}
}
//cout<<"Nxy: "<<Nx[i]<<", "<<Ny[i]<<endl;

}
*/

/*
//cout<<Pn;
for(int i=0; i<197; i++)
{
// cout<<"x"<<Nx[i]<<"y"<<Ny[i]<<endl;
if(Stat[i]==0) continue;

Lx += Ny[i] * Pn;
Ly += -Nx[i] * Pn;
}

double Lx2=0;
double Ly2=0;

for(int i=197; i<394; i++)
{
if(Stat[i]==0) continue;
Lx2 += Ny[i] * (-1.0*Pn);
Ly2 += -Nx[i] * (-1.0*Pn);
```

```
}
*/

//cout<<"#Event: "<<jentry<<"\tImp: "<<Imp<<" Lx "<<Lx+Lx2<<" Ly "<<Ly+Ly2<<endl;
double L= sqrt(Lx*Lx + Ly*Ly);
Lb->Fill(Imp,abs(L));
Lxb->Fill(Imp,Lx);
modLxb->Fill(Imp,abs(Lx));
Lyb->Fill(Imp,Ly);
modLyb->Fill(Imp,abs(Ly));
if(jentry % 1000==0)
cout<<"#Event: "<<jentry<<"\tImp: "<<Imp<<" Lx "<<Lx<<" Ly "<<Ly<<" L "<<L<<endl;
fang<<"#Event: "<<jentry<<"\tImp: "<<Imp<<" Lx "<<Lx<<" Ly "<<Ly<<" L "<<L<<endl;
avsb<<setw(25)<<Imp<<setw(25)<<L<<endl;
cout<<"#Event: "<<jentry<<"\tImp: "<<Imp<<" L "<<L<<endl;
//fang<<"#Event: "<<jentry<<"\tImp: "<<Imp<<" L "<<L<<endl;
}//end of event loop
fout->Write();

}//end of program
```

Header file

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Sun Mar 26 18:46:51 2017 by ROOT version 5.34/36
// from TTree tr/Reconst ntuple
// found on file: AuAu200_SM_ntmax150_1.5mb_MinBias_rndmpsi_file0.root
//////////////////////////////////////////////////////////

#ifndef AngularMomAna_h
```

```
#define AngularMomAna_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>

// Header file for the classes stored in the TTree if any.

// Fixed size dimensions of array or collections stored in the TTree if any.

class AngularMomAna {
public :
TTree          *fChain;  //!pointer to the analyzed TTree or TChain
Int_t           fCurrent; //!current Tree number in a TChain

// Declaration of leaf types
Int_t           Event;
Int_t           Mult;
Int_t           Npartp;
Int_t           Npartt;
Int_t           Nesp;
Int_t           Ninesp;
Int_t           Nest;
Int_t           Ninest;
Float_t         Imp;  //Impact Parameter
Int_t           Na;
Int_t           Nb;
Int_t           Nab;
Float_t         Psi;
Float_t         Nx[394];  //[Nab] //X-pos of nucleons
```

```
Float_t         Ny[394];  //[Nab] //Y-pos of nucleons
Float_t         Nz[394];  //[Nab]
Int_t           Stat[394];  //[Nab] //0-No part., Non-zero : participant
Int_t           pnID[394];  //[Nab]
Int_t           PID[9337];  //[Mult]
Float_t         Px[9337];  //[Mult]
Float_t         Py[9337];  //[Mult]
Float_t         Pz[9337];  //[Mult]
Float_t         Mass[9337];  //[Mult]
Float_t         XX[9337];  //[Mult]
Float_t         YY[9337];  //[Mult]
Float_t         ZZ[9337];  //[Mult]
Float_t         TT[9337];  //[Mult]

// List of branches
TBranch        *b_Event;  //!
TBranch        *b_Mult;  //!
TBranch        *b_Npartp;  //!
TBranch        *b_Npartt;  //!
TBranch        *b_Nesp;  //!
TBranch        *b_Ninesp;  //!
TBranch        *b_Nest;  //!
TBranch        *b_Ninest;  //!
TBranch        *b_Imp;  //!
TBranch        *b_Na;  //!
TBranch        *b_Nb;  //!
TBranch        *b_Nab;  //!
TBranch        *b_Psi;  //!
TBranch        *b_Nx;  //!
TBranch        *b_Ny;  //!
```

```
   TBranch        *b_Nz;   //!
   TBranch        *b_Stat;   //!
   TBranch        *b_pnID;   //!
   TBranch        *b_PID;   //!
   TBranch        *b_Px;   //!
   TBranch        *b_Py;   //!
   TBranch        *b_Pz;   //!
   TBranch        *b_Mass;   //!
   TBranch        *b_XX;   //!
   TBranch        *b_YY;   //!
   TBranch        *b_ZZ;   //!
   TBranch        *b_TT;   //!

   AngularMomAna(TTree *tree=0);
   virtual ~AngularMomAna();
   virtual Int_t   Cut(Long64_t entry);
   virtual Int_t   GetEntry(Long64_t entry);
   virtual Long64_t LoadTree(Long64_t entry);
   virtual void    Init(TTree *tree);
   virtual void    Loop();
   virtual Bool_t  Notify();
   virtual void    Show(Long64_t entry = -1);
};

#endif

#ifdef AngularMomAna_cxx
AngularMomAna::AngularMomAna(TTree *tree) : fChain(0)
{
// if parameter tree is not specified (or zero), connect the file
```

```
// used to generate this class and read the Tree.
/*if (tree == 0) {
TFile *f =
    (TFile*)gROOT->GetListOfFiles()->FindObject("AuAu200_SM_ntmax150_1.5mb_MinBias_rndmpsi_file
if (!f || !f->IsOpen()) {
f = new TFile("AuAu200_SM_ntmax150_1.5mb_MinBias_rndmpsi_file0.root");
}
f->GetObject("tr",tree);

}
Init(tree);
*/

if (tree == 0) {
TChain *chain = new TChain("tr");
TString fNameList = TString("test.list");
cout <<" Load files from file: " << fNameList << endl;

ifstream fList((char*)fNameList);
if (!fList)
{
cout << "!!! Can't open file " << fNameList << endl;
return;
}

char lineFromFile[255];
while(fList.getline(lineFromFile, 250))
{
TString fileName = lineFromFile;
```

```
fileName = "" + fileName;
// Open this loop only when you need the information about loading files \ \

if(chain->Add((char*)fileName)){};
}
fList.close();
tree = chain;
}
Init(tree);

}

AngularMomAna::~AngularMomAna()
{
if (!fChain) return;
delete fChain->GetCurrentFile();
}

Int_t AngularMomAna::GetEntry(Long64_t entry)
{
// Read contents of entry.
if (!fChain) return 0;
return fChain->GetEntry(entry);
}
Long64_t AngularMomAna::LoadTree(Long64_t entry)
{
// Set the environment to read one entry
if (!fChain) return -5;
Long64_t centry = fChain->LoadTree(entry);
if (centry < 0) return centry;
```

```
if (fChain->GetTreeNumber() != fCurrent) {
fCurrent = fChain->GetTreeNumber();
Notify();
}
return centry;
}

void AngularMomAna::Init(TTree *tree)
{
// The Init() function is called when the selector needs to initialize
// a new tree or chain. Typically here the branch addresses and branch
// pointers of the tree will be set.
// It is normally not necessary to make changes to the generated
// code, but the routine can be extended by the user if needed.
// Init() will be called many times when running on PROOF
// (once per file to be processed).

// Set branch addresses and branch pointers
if (!tree) return;
fChain = tree;
fCurrent = -1;
fChain->SetMakeClass(1);

fChain->SetBranchAddress("Event", &Event, &b_Event);
fChain->SetBranchAddress("Mult", &Mult, &b_Mult);
fChain->SetBranchAddress("Npartp", &Npartp, &b_Npartp);
fChain->SetBranchAddress("Npartt", &Npartt, &b_Npartt);
fChain->SetBranchAddress("Nesp", &Nesp, &b_Nesp);
fChain->SetBranchAddress("Ninesp", &Ninesp, &b_Ninesp);
fChain->SetBranchAddress("Nest", &Nest, &b_Nest);
```

```
fChain->SetBranchAddress("Ninest", &Ninest, &b_Ninest);
fChain->SetBranchAddress("Imp", &Imp, &b_Imp);
fChain->SetBranchAddress("Na", &Na, &b_Na);
fChain->SetBranchAddress("Nb", &Nb, &b_Nb);
fChain->SetBranchAddress("Nab", &Nab, &b_Nab);
fChain->SetBranchAddress("Psi", &Psi, &b_Psi);
fChain->SetBranchAddress("Nx", Nx, &b_Nx);
fChain->SetBranchAddress("Ny", Ny, &b_Ny);
fChain->SetBranchAddress("Nz", Nz, &b_Nz);
fChain->SetBranchAddress("Stat", Stat, &b_Stat);
fChain->SetBranchAddress("pnID", pnID, &b_pnID);
fChain->SetBranchAddress("PID", PID, &b_PID);
fChain->SetBranchAddress("Px", Px, &b_Px);
fChain->SetBranchAddress("Py", Py, &b_Py);
fChain->SetBranchAddress("Pz", Pz, &b_Pz);
fChain->SetBranchAddress("Mass", Mass, &b_Mass);
fChain->SetBranchAddress("XX", XX, &b_XX);
fChain->SetBranchAddress("YY", YY, &b_YY);
fChain->SetBranchAddress("ZZ", ZZ, &b_ZZ);
fChain->SetBranchAddress("TT", TT, &b_TT);
Notify();
}

Bool_t AngularMomAna::Notify()
{
// The Notify() function is called when a new file is opened. This
// can be either for a new TTree in a TChain or when when a new TTree
// is started when using PROOF. It is normally not necessary to make changes
// to the generated code, but the routine can be extended by the
// user if needed. The return value is currently not used.
```

```
return kTRUE;
}

void AngularMomAna::Show(Long64_t entry)
{
// Print contents of entry.
// If entry is not specified, print current entry
if (!fChain) return;
fChain->Show(entry);
}
Int_t AngularMomAna::Cut(Long64_t entry)
{
// This function may be called from Loop.
// returns 1 if entry is accepted.
// returns -1 otherwise.
return 1;
}
#endif // #ifdef AngularMomAna_cxx
```

Program for invariant mass calculation and cos theta

```
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include<math.h>


#include <TROOT.h>
```

```
#include <TChain.h>
#include <TFile.h>
//PID K+= 321
// PID Pi-=-211

void dNcos()
{
ifstream fin;
fin.open("PhojetPPCollision7TeV.txt",ios::in);
if(!fin)
{
cout<<"Can not open the file "<<endl;
return -1;
}
Char_t eve1[6],eve2[6];
Int_t nevt; //Event number
Int_t nptls,process;//# particles in an event
Int_t Pid; // PID of the particle
Int_t slno; //Serial # of the particle
Int_t prisec; //Primary (1)or secondary(2)
Int_t charge; // charge of the particle
Int_t totEvt; //Total no of events in the file
Float_t Px,Py,Pz,Energy; //four momentum of the particle

//start
fin>>eve1>>eve2>>totEvt;
cout<<"Total events"<<totEvt<<endl;
//Define 2D Histograms
TH2D *KpPim = new TH2D("K+Pi-", "InvariantMass and CosTheta for K+Pi- pair", 90,
    0.6, 1.5, 10, -1, 1);
```

```
TH2D *KmPip = new TH2D("K-Pi+", "InvariantMass and CosTheta for K-Pi+ pair", 90,
    0.6, 1.5, 10, -1, 1);
TH2D *KpPip = new TH2D("K+Pi+", "InvariantMass and CosTheta for K+Pi+ pair", 90,
    0.6, 1.5, 10, -1, 1);
TH2D *KmPim = new TH2D("K-Pi-", "InvariantMass and CosTheta for K-Pi- pair", 90,
    0.6, 1.5, 10, -1, 1);
double PxKp[400]; //K+ four vector
double PyKp[400];
double PzKp[400];
double EKp[400];

double PxKm[400]; //K- four vector
double PyKm[400];
double PzKm[400];
double EKm[400];



double PxPim[400]; //Pi- four vector
double PyPim[400];
double PzPim[400];
double EPim[400];

double PxPip[400]; //Pi+ four vector
double PyPip[400];
double PzPip[400];
double EPip[400];

//Definitions of vectors,variables for double "for loop" combinations
TLorentzVector vKp;
TLorentzVector vPim;
```

```
TLorentzVector vMother;
TVector3 b; //Boost
TVector3 dau; //daughter 3 momentum vector
double coss;
double M;

TLorentzVector vKm;
TLorentzVector vPip;
TLorentzVector vMother2;
TVector3 b2; //Boost
TVector3 dau2; //daughter 3 momentum vector
double coss2;
double M2;

TLorentzVector vMother3;
TVector3 b3; //Boost
TVector3 dau3; //daughter 3 momentum vector
double coss3;
double M3;
TLorentzVector vMother4;
TVector3 b4; //Boost
TVector3 dau4; //daughter 3 momentum vector
double coss4;
double M4;

//Define z axis ; z= P cross (0,0,1) direction of pp beam
TVector3 z;
//************************************************************EVENT
    LOOP*****************************************************
for(Int_t ievt = 0; ievt < totEvt; ievt++)
```

```
{
fin>>nevt>>nptls>>process;
cout<<"Processing Event # = "<<nevt<<" no. of Particles = "<<nptls<<endl;
int c321=0;
int c211m=0;
int c321m=0;
int c211=0;



for(Int_t ipart =0;ipart < nptls; ipart++)
{
fin>>slno>>prisec>>Pid>>charge>>Px>>Py>>Pz>>Energy;

if(TMath::Sqrt(Px*Px + Py*Py )<0.001)
continue;

//selecting only primary particle
// if(prisec != 1)continue;
if(Pid == 321) //K+
{c321++;
//
    cout<<setw(5)<<Pid<<setw(5)<<charge<<setw(15)<<Px<<setw(15)<<Py<<setw(15)<<Pz<<setw(15)<<En
PxKp[c321]=Px; PyKp[c321]=Py;PzKp[c321]=Pz;EKp[c321]=Energy;
//cout<<c321<<setw(5)<<Pid<<setw(5)<<charge<<setw(15)<<PxKp[c321]<<setw(15)<<PyKp[c321]<<setw(
}
else if(Pid ==-211) //Pi-
{c211m++;
//
    cout<<setw(5)<<Pid<<setw(5)<<charge<<setw(15)<<Px<<setw(15)<<Py<<setw(15)<<Pz<<setw(15)<<En
PxPim[c211m]=Px; PyPim[c211m]=Py;PzPim[c211m]=Pz;EPim[c211m]=Energy;
```

```
//cout<<c211m<<setw(5)<<Pid<<setw(5)<<charge<<setw(15)<<PxPim[c211m]<<setw(15)<<PyPim[c211m]<<s
}
else if(Pid==-321)
{c321m++;
PxKm[c321m]=Px; PyKm[c321m]=Py;PzKm[c321m]=Pz;EKm[c321m]=Energy;
}
else if(Pid==211)
{
c211++;
PxPip[c211]=Px; PyPip[c211]=Py;PzPip[c211]=Pz;EPip[c211]=Energy;
}









}//*****************************************************END OF PARTICLE
    LOOP*****************************************************************

//cout<<" c321 "<<c321<<" c211m  "<<c211m<<" c321m "<<c321<<" c211 "<<c211<<endl;



//**********************           K+    and Pi -
    ****************************

for(int i=1;i<=c321;i++) //Combinations of K+ and Pi-
```

```
{
vKp.SetPxPyPzE(PxKp[i],PyKp[i],PzKp[i],EKp[i]);
for(int j=1;j<=c211m;j++)
{

vPim.SetPxPyPzE(PxPim[j],PyPim[j],PzPim[j],EPim[j]);

vMother=vKp+vPim;

//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;

//cout<<" K+ "<<vKp.M2()<<" Pi- "<<vPim.M2()<<endl;

b.SetXYZ(-vMother.Px()/vMother.E(),-vMother.Py()/vMother.E(),-vMother.Pz()/vMother.E());
    //boost



z.SetXYZ(vMother.Py()/sqrt( pow(vMother.Px(),2)+pow(vMother.Py(),2)
    ),-vMother.Px()/sqrt( pow(vMother.Px(),2)+pow(vMother.Py(),2) ),0); //z axis
    set

//vMother.Boost(b);
//vKp.Boost(b);
vPim.Boost(b);
//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
```

```
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;


dau.SetXYZ(vPim.Px(),vPim.Py(),vPim.Pz());
//Calculating cos in Vector Meson Rest frame cos= V(daughter). z

coss=dau*z/dau.Mag();

M=vMother.M();
//cout<<"cos "<<coss<<" Inv M " << vMother.M()<<endl;


KpPim -> Fill(M,coss);


}
}
//*******************************END of K+ and Pi-
    ***************************************

//***************************K- and
    Pi+***************************************************
//

for(int i=1;i<=c321m;i++) //comb of K- and Pi+
{

vKm.SetPxPyPzE(PxKm[i],PyKm[i],PzKm[i],EKm[i]);
```

```
for(int j=1;j<=c211;j++)
{



vPip.SetPxPyPzE(PxPip[j],PyPip[j],PzPip[j],EPip[j]);



vMother2=vKm+vPip;

//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;

//cout<<" K+ "<<vKp.M2()<<" Pi- "<<vPim.M2()<<endl;

b2.SetXYZ(-vMother2.Px()/vMother2.E(),-vMother2.Py()/vMother2.E(),-vMother2.Pz()/vMother2.E())
    //Boost Vector



z.SetXYZ(vMother2.Py()/sqrt( pow(vMother2.Px(),2)+pow(vMother2.Py(),2)
    ),-vMother2.Px()/sqrt( pow(vMother2.Px(),2)+pow(vMother2.Py(),2) ),0);

//vMother.Boost(b);
//vKm.Boost(b2);
vPip.Boost(b2);
//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
```

```
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;


dau2.SetXYZ(vPip.Px(),vPip.Py(),vPip.Pz()); //daughter
//Calculating cos in Vector Meson Rest frame cos= V(daughter). z

coss2=dau2*z/dau2.Mag();

M2=vMother2.M();
//cout<<"cos "<<coss2<<" Inv M " << M2<<endl;


KmPip -> Fill(M2,coss2);
}
}
//*********************************END of K- and Pi
    +*******************************************


//**********************              K+    and Pi +
    ****************************
for(int i=1;i<=c321;i++) //Combinations of K+ and Pi+
{
vKp.SetPxPyPzE(PxKp[i],PyKp[i],PzKp[i],EKp[i]);
for(int j=1;j<=c211;j++)
{

vPip.SetPxPyPzE(PxPip[j],PyPip[j],PzPip[j],EPip[j]);
```

```
vMother3=vKp+vPip;

//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;

//cout<<" K+ "<<vKp.M2()<<" Pi- "<<vPim.M2()<<endl;

b3.SetXYZ(-vMother3.Px()/vMother3.E(),-vMother3.Py()/vMother3.E(),-vMother3.Pz()/vMother3.E())
    //boost



z.SetXYZ(vMother3.Py()/sqrt( pow(vMother3.Px(),2)+pow(vMother3.Py(),2)
    ),-vMother3.Px()/sqrt( pow(vMother3.Px(),2)+pow(vMother3.Py(),2) ),0); //z
    axis set

//vMother.Boost(b);
//vKp.Boost(b);
vPip.Boost(b3);
//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;


dau3.SetXYZ(vPip.Px(),vPip.Py(),vPip.Pz());
//Calculating cos in Vector Meson Rest frame cos= V(daughter). z
```

```
coss3=dau3*z/dau3.Mag();

M3=vMother3.M();
//cout<<"cos "<<coss3<<" Inv M " << vMother3.M()<<endl;


KpPip -> Fill(M3,coss3);


}
}
//*********************************END of K+ and Pi+
    ****************************************


//**********************          K-   and Pi -
    ******************************

for(int i=1;i<=c321m;i++) //Combinations of K- and Pi-
{
vKm.SetPxPyPzE(PxKm[i],PyKm[i],PzKm[i],EKm[i]);
for(int j=1;j<=c211m;j++)
{

vPim.SetPxPyPzE(PxPim[j],PyPim[j],PzPim[j],EPim[j]);

vMother4=vKm+vPim;

//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
```

```
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;

//cout<<" K+ "<<vKp.M2()<<" Pi- "<<vPim.M2()<<endl;

b4.SetXYZ(-vMother4.Px()/vMother4.E(),-vMother4.Py()/vMother4.E(),-vMother4.Pz()/vMother4.E())
    //boost



z.SetXYZ(vMother4.Py()/sqrt( pow(vMother4.Px(),2)+pow(vMother4.Py(),2)
    ),-vMother4.Px()/sqrt( pow(vMother4.Px(),2)+pow(vMother4.Py(),2) ),0); //z
    axis set

//vMother.Boost(b);
//vKp.Boost(b);
vPim.Boost(b4);
//cout<<vKp[0]<<" "<<vKp[1]<<" "<<vKp[2]<<" "<<vKp[3]<<" "<<vKp.M()<<"
    "<<vPim[0]<<" "<<vPim[1]<<" "<<vPim[2]<<" "<<vPim[3]<<" "<<vPim.M()<<"
    "<<vMother[0]<<" "<<vMother[1]<<" "<<vMother[2]<<" "<<vMother[3]<<"
    "<<vMother.M()<<endl;


dau4.SetXYZ(vPim.Px(),vPim.Py(),vPim.Pz());
//Calculating cos in Vector Meson Rest frame cos= V(daughter). z

coss4=dau4*z/dau4.Mag();

M4=vMother4.M();
//cout<<"cos "<<coss4<<" Inv M " << vMother4.M()<<endl;
```

```
KmPim -> Fill(M4,coss4);



}
}
//*******************************END of K- and Pi-
    ***************************************
```

```
}//*****************************end of event loop*********************************
```

```
//Fill Histograms to File
TFile *File = new TFile("Histograms.root", "recreate");
File->cd();
KpPim->Write();
KmPip->Write();
KpPip->Write();
KmPim->Write();


//double Nc=KpPim->GetBinContent(1,1);
//cout<<Nc;
```

```
return 0;
}
```

Final Program for analyzing resonance signals

```
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include<math.h>

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>

void read()
{
gStyle->SetLineWidth(2);
gStyle->SetOptFit(1);

TFile *f = TFile::Open("Histograms.root"); //extract 2D scatter plots from the
    root file
f->ls{};
TH2D *hKpPim=(TH2D*)f->Get("K+Pi-");
TH2D *hKmPip=(TH2D*)f->Get("K-Pi+");
```

```
TH2D *hKpPip=(TH2D*)f->Get("K+Pi+");
TH2D *hKmPim=(TH2D*)f->Get("K-Pi-");

TF1 *fun[10];
TF1 *funbkg[10];
TF1 *funSig[10];
//hKpPim->Draw();
TH1D *hsig1[10]; //Histogram for signal 1
TH1D *hsig2[10]; //Histogram for signal 2
TH1D *hbkg1[10]; //Histogram for background 1
TH1D *hbkg2[10]; //Histogram for background 2

TH1D *hsigt[10]; //Histogram for total signal
TH1D *hbkgt[10]; //Histogram for total background
TH1D *hsigonly[10]; //Histogram of total signal minus background
TCanvas *can[10];
TCanvas *canbkg[10]; //Canvas for background

Double_t Mass[10],Width[10],Yield[10];
Double_t ErMass[10],ErWidth[10],ErYield[10];
for(int i=0;i<10;i++)
{
can[i] = new TCanvas(Form("can%d",i),"",10,10,600,600);
can[i]->SetLeftMargin(0.2);
can[i]->SetRightMargin(0.05);
can[i]->SetTopMargin(0.05);
can[i]->SetBottomMargin(0.13);

canbkg[i] = new TCanvas(Form("canbkg%d",i),"",10,10,600,600);
canbkg[i]->SetLeftMargin(0.2);
```

```
canbkg[i]->SetRightMargin(0.05);
canbkg[i]->SetTopMargin(0.05);
canbkg[i]->SetBottomMargin(0.13);


hsigt[i]=new TH1D(Form("Signal unlike pair%d",i),"",90,0.6,1.5);
hbkgt[i]=new TH1D(Form("bkgt%d",i),"",90,0.6,1.5);
hsigonly[i]=new TH1D(Form("Final Signal %d",i),"",90,0.6,1.5);
hsig1[i] = hKpPim->ProjectionX(Form("KpPim%d",i),i+1,i+1,"e");//Projection for
    each cos theta bin for K+Pi-
hsig2[i] = hKmPip->ProjectionX(Form("KmPip%d",i),i+1,i+1,"e");//Projection for
    each cos theta bin for K-Pi+
hbkg1[i] = hKpPip->ProjectionX(Form("KpPip%d",i),i+1,i+1,"e");//Projection for
    each cos theta bin for K+Pi+
hbkg2[i] = hKmPim->ProjectionX(Form("KmPim%d",i),i+1,i+1,"e");//Projection for
    each cos theta bin for K-Pi-
hsigt[i]->Add(hsig1[i],hsig2[i],1,1); //Total signal=signal1+signal2

for(int bin=1;bin<hbkgt[i]->GetNbinsX();bin++)
{ //Background combined together by like sign technique
double
    totbkg=2*TMath::Sqrt(hbkg1[i]->GetBinContent(bin)*hbkg2[i]->GetBinContent(bin));
double bkgerr;
bkgerr=0.01;
hbkgt[i]->SetBinContent(bin,totbkg);
hbkgt[i]->SetBinError(bin,bkgerr);
}
canbkg[i]->cd();
hsigt[i]->GetXaxis()->SetTitle("M_{K#pi} (GeV/c^{2})");
hsigt[i]->GetXaxis()->SetTitleOffset(1.2);
```

```
hsigt[i]->GetYaxis()->SetTitleOffset(1.7);
hsigt[i]->GetYaxis()->SetTitle("Counts");
hsigt[i]->SetMarkerColor(1);
hsigt[i]->SetMarkerStyle(20);
hsigt[i]->SetMarkerSize(1.0);
hsigt[i]->Draw();      //Signal Total=Signal1(K+Pi-) + Signal2(K-Pi+)

hbkgt[i]->SetMarkerColor(2);
hbkgt[i]->SetMarkerStyle(24);
hbkgt[i]->SetMarkerSize(1.0);
hbkgt[i]->Draw("same");



canbkg[i]->SaveAs(Form("ScaledBkgInvariantMassDistInCosThetaStarBin%d.gif",i));
canbkg[i]->SaveAs(Form("ScaledBkgInvariantMassDistInCosThetaStarBin%d.png",i));
canbkg[i]->Print(Form("ScaledBkgInvariantMassDistInCosThetaStarBin%d.eps",i));
canbkg[i]->cd();

fun[i]= new TF1(Form("BWfunc%d",i),fitfunction,0.65,1.2,6); //Fit function to
    signalonly:BW+poly2

fun[i]->SetParameters(1000,0.048,0.896,150,150,60);
fun[i]->SetParName(0,"A");
fun[i]->SetParName(1,"#Gamma");
fun[i]->SetParName(2,"Mass");
fun[i]->SetParName(3,"a_{0}"); //coefficient of degree2 in poly2
fun[i]->SetParName(4,"a_{1}"); //coefficient of degree1 in poly2
fun[i]->SetParName(5,"a_{2}"); //coefficient of degree0 in poly2
//fun[i]->SetParLimits(0,0,10000000000);
//fun[i]->SetParLimits(2,0.8,0.95);
```

```
//Final Signal=Total Signal(due to K+Pi- & K-Pi+) - Total background (like sign
    K+Pi+ and K-Pi-)
hsigonly[i]->Add(hsigt[i],hbkgt[i],1,-1);
can[i]->cd();
hsigonly[i]->GetXaxis()->SetTitle("M_{K#pi} (GeV/c^{2})");
hsigonly[i]->GetXaxis()->SetTitleOffset(1.2);
hsigonly[i]->GetYaxis()->SetTitleOffset(1.7);
hsigonly[i]->GetYaxis()->SetTitle("Counts");
hsigonly[i]->SetMarkerColor(1);
hsigonly[i]->SetMarkerStyle(20);
hsigonly[i]->SetMarkerSize(1.0);
hsigonly[i]->Draw();
// gPad->BuildLegend();
hsigonly[i]->Fit(fun[i],"REM");
funbkg[i]= new TF1(Form("Bkgfunc%d",i),"([0]*x*x+[1]*x+[2])",0.65,1.2);
funbkg[i]->SetParameters(fun[i]->GetParameter(3),fun[i]->GetParameter(4),fun[i]->GetParameter(5
funbkg[i]->SetLineColor(3);
funbkg[i]->Draw("same");
funSig[i]= new TF1(Form("Sigfunc%d",i),"(0.01*([0]/2.*TMath::Pi())*([1])*(1./(
    pow( x-[2],2) + [1]*[1]*0.25)))",0.65,1.2);
funSig[i]->SetParameters(fun[i]->GetParameter(0),fun[i]->GetParameter(1),fun[i]->GetParameter(2
funSig[i]->SetLineColor(4); //4 Blue
funSig[i]->Draw("same"); //BreitWigner function+Poly2
//gPad->BuildLegend();
can[i]->SaveAs(Form("InvariantMassDistInCosThetaStarBin%d.gif",i));
can[i]->SaveAs(Form("InvariantMassDistInCosThetaStarBin%d.png",i));
can[i]->Print(Form("InvariantMassDistInCosThetaStarBin%d.eps",i));
can[i]->cd();
Mass[i] = fun[i]->GetParameter(2); //Taking Values of Parameters from the fitted
    function
```

```
Width[i] = fun[i]->GetParameter(1)*1000;
Double_t y = fun[i]->GetParameter(0); //Get Yield
ErMass[i] = fun[i]->GetParError(2);
ErWidth[i] = fun[i]->GetParError(1)*1000;
Double_t ery = fun[i]->GetParError(0); //error in yield
Yield[i] = y / 0.2; //normalized by bin width
ErYield[i] = ery /0.2;
}

TLine *lm = new TLine(-1,0.896,1,0.896);
lm->SetLineColor(4);
lm->SetLineWidth(2);
lm->SetLineStyle(2);
TLine *lw = new TLine(-1,52,1,52);
lw->SetLineColor(4);
lw->SetLineWidth(2);
lw->SetLineStyle(2);

Double_t CosThetaStar[10] = {-0.9,-0.7,-0.5,-0.3,-0.1,0.1,0.3,0.5,0.7,0.9};
Double_t ErCosThetaStar[10] = {0,0,0,0,0,0,0,0,0,0};
TGraphErrors *grM = new TGraphErrors(10,CosThetaStar,Mass,ErCosThetaStar,ErMass);
TGraphErrors *grW = new
    TGraphErrors(10,CosThetaStar,Width,ErCosThetaStar,ErWidth); //in MeV
TGraphErrors *grdNdCostTheta = new
    TGraphErrors(10,CosThetaStar,Yield,ErCosThetaStar,ErYield);

TCanvas *c = new TCanvas("c","",10,10,600,600);
c->SetLeftMargin(0.2);
c->SetRightMargin(0.05);
c->SetTopMargin(0.05);
```

```
c->SetBottomMargin(0.13);
grM->SetTitle("Mass vs. cos#theta");
grM->SetMaximum(0.905);
grM->SetMinimum(0.89);
grM->SetMarkerColor(1);
grM->SetMarkerStyle(20);
grM->SetMarkerSize(1.0);
grM->GetXaxis()->SetTitle("cos#theta");
grM->GetXaxis()->SetTitleSize(0.05);
grM->GetXaxis()->SetTitleOffset(1.25);
grM->GetXaxis()->SetTitleFont(42);
grM->GetXaxis()->CenterTitle(true);
grM->GetXaxis()->SetLabelSize(0.05);
grM->GetXaxis()->SetLabelFont(42);
grM->GetXaxis()->SetNdivisions(505);
grM->GetYaxis()->SetTitle("Mass (GeV/c^{2})");
grM->GetYaxis()->SetTitleFont(42);
grM->GetYaxis()->CenterTitle(true);
grM->GetYaxis()->SetTitleSize(0.05);
grM->GetYaxis()->SetTitleOffset(2.);
grM->GetYaxis()->SetLabelSize(0.05);
grM->GetYaxis()->SetLabelFont(42);
grM->GetYaxis()->SetNdivisions(505);
grM->Draw("AP");
lm->Draw();
c->SaveAs("MassVsCosThetaStar.gif");
c->SaveAs("MassVsCosThetaStar.png");
c->Print("MassVsCosThetaStar.eps");
c->cd();
TCanvas *c1 = new TCanvas("c1","",10,10,600,600);
```

```
c1->SetLeftMargin(0.2);
c1->SetRightMargin(0.05);
c1->SetTopMargin(0.05);
c1->SetBottomMargin(0.13);
grW->SetTitle("Width vs. cos#theta");
grW->SetMaximum(60);
grW->SetMinimum(0);
grW->SetMarkerColor(1);
grW->SetMarkerStyle(20);
grW->SetMarkerSize(1.0);
grW->GetXaxis()->SetTitle("cos#theta");
grW->GetXaxis()->SetTitleSize(0.05);
grW->GetXaxis()->SetTitleOffset(1.25);
grW->GetXaxis()->SetTitleFont(42);
grW->GetXaxis()->CenterTitle(true);
grW->GetXaxis()->SetLabelSize(0.05);
grW->GetXaxis()->SetLabelFont(42);
grW->GetXaxis()->SetNdivisions(505);
grW->GetYaxis()->SetTitle("#Gamma (MeV/c^{2})");
grW->GetYaxis()->SetTitleFont(42);
grW->GetYaxis()->CenterTitle(true);
grW->GetYaxis()->SetTitleSize(0.05);
grW->GetYaxis()->SetTitleOffset(1.7);
grW->GetYaxis()->SetLabelSize(0.05);
grW->GetYaxis()->SetLabelFont(42);
grW->GetYaxis()->SetNdivisions(505);
grW->Draw("AP");
lw->Draw();
c1->SaveAs("WidthVsCosThetaStar.gif");
c1->SaveAs("WidthVsCosThetaStar.png");
```

```
c1->Print("WidthVsCosThetaStar.eps");
c1->cd();

TF1 *funRho = new TF1("funRho",RhoZeroZero,-1,1.,2);
funRho->SetParameters(1000,0.3);
funRho->SetParNames("Const.","#rho_{00}");
TCanvas *c2 = new TCanvas("c2","",10,10,600,600);
c2->SetLeftMargin(0.2);
c2->SetRightMargin(0.05);
c2->SetTopMargin(0.05);
c2->SetBottomMargin(0.13);
grdNdCostTheta->SetMaximum(600000);
grdNdCostTheta->SetMinimum(540000);
grdNdCostTheta->SetTitle("dN/dcos#theta vs. cos#theta");
grdNdCostTheta->SetMarkerColor(1);
grdNdCostTheta->SetMarkerStyle(20);
grdNdCostTheta->SetMarkerSize(1.0);
grdNdCostTheta->GetXaxis()->SetTitle("cos#theta");
grdNdCostTheta->GetXaxis()->SetTitleSize(0.05);
grdNdCostTheta->GetXaxis()->SetTitleOffset(1.25);
grdNdCostTheta->GetXaxis()->SetTitleFont(42);
grdNdCostTheta->GetXaxis()->CenterTitle(true);
grdNdCostTheta->GetXaxis()->SetLabelSize(0.05);
grW->GetXaxis()->SetLabelFont(42);
grdNdCostTheta->GetXaxis()->SetNdivisions(505);
grdNdCostTheta->GetYaxis()->SetTitle("dN/dcos#theta");
grdNdCostTheta->GetYaxis()->SetTitleFont(42);
grdNdCostTheta->GetYaxis()->CenterTitle(true);
grdNdCostTheta->GetYaxis()->SetTitleSize(0.05);
grdNdCostTheta->GetYaxis()->SetTitleOffset(1.7);
```

```
grdNdCostTheta->GetYaxis()->SetLabelSize(0.05);
grdNdCostTheta->GetYaxis()->SetLabelFont(42);
grdNdCostTheta->GetYaxis()->SetNdivisions(505);
grdNdCostTheta->Draw("AP");
grdNdCostTheta->Fit(funRho,"REM");
c2->SaveAs("dNdcosthetaVsCosThetaStar.gif");
c2->SaveAs("dNdcosthetaVsCosThetaStar.png");
c2->Print("dNdcosthetaVsCosThetaStar.eps");
c2->cd();
}

double fitfunction(double *x,double *par)
{
return 0.01*(par[0]/2.*TMath::Pi())*(par[1])*(1./( pow( x[0]-par[2],2) +
    par[1]*par[1]*0.25)) + par[3]*x[0]*x[0] +par[4]*x[0] +par[5];
}
double RhoZeroZero(double *x,double *par)
{
return par[0]*(3./4.)*((1.-par[1])+(3*par[1]-1)*TMath::Cos(x[0]));
}
```

---

* viraj.thakkar@niser.ac.in

[1] Schilling, K., P. Seyboth, and G. Wolf. "On the analysis of vector-meson production by polarized photons." Nuclear Physics B 15.2 (1970): 397-412.

[2] Liang, Zuo-Tang, and Xin-Nian Wang. "Spin alignment of vector mesons in non-central A+ A collisions." Physics Letters B 629.1 (2005): 20-26.

[3] Becattini, Francesco, Fulvio Piccinini, and J. Rizzo. "Angular momentum conservation in heavy ion collisions at very high energy." Physical Review C 77.2 (2008): 024906.

[4] Adams, John, et al. "K (892)* resonance production in Au+ Au and p+ p collisions at s NN= 200 GeV." Physical Review C 71.6 (2005): 064902.