Analysis of Di-Hadron Azimuthal Spin Asymmetries in Semi-Inclusive Deep-Inelastic Scattering Using CLAS at JLab

Wyatt Meldman-Floch
Advised by Keith Griffioen

Abstract

The goal of my research is to explore the contribution of quark orbital angular momentum to the spin of the proton. This is accomplished by scattering electrons from quarks in a proton. I am studying the case where the struck quark hadronizes into a pion-pair by analyzing the trajectories of the electron and pion-pair. Since the pion's azimuthal distribution about the axis of momentum transfer is sensitive to the orbital motion of the original quark, my project looks for asymmetries in the azimuthal angular distribution of pion-pair production.

I. Mystery of Proton Spin

The origin of the proton's spin is still a mystery. Consider the spin sum rule

$$\frac{1}{2} = \frac{\Delta\Sigma}{2} + \Delta G + L_z \tag{1}$$

where $\frac{1}{2}$ is the proton spin in $\hbar$ units, $\Delta\Sigma$ is the spin contribution from quarks, ΔG is the spin contribution of quarks and gluons and L_z is the contribution of quark and gluon orbital angular momentum. The spin contribution of orbital angular momentum is L_z. Experiments [1] testing the attribution of the proton's spin to its quarks concluded that quark spin accounts for about 25 percent of the proton's total spin. Other Experiments [2] indicate that gluon spin also contributes little to the proton's spin. Therefore orbital angular momentum is most likely the dominant component of the proton spin. However, orbital angular momentum is difficult to measure directly. One probe that is sensitive to the orbital angular momentum of the proton is high energy electron quark scattering. The struck quark can hadronize into various particles, the lightest of which are pions. The pion's azimuthal distribution with respect to the electron momentum transfer direction is sensitive to the orbital motion of the original quark [2]. Some data have already been taken in the EG1 experiment at JLab's Hall B using the CLAS spectrometer. A schematic diagram of this type of experiment is given in Figure 1. Figure 1 shows the kinematics of the semi-inclusive deep-inelastic scattering reaction. An incident electron ℓ scatters from a quark in a proton and the scattered electron continues on the trajectory ℓ'. The momentum of the virtual photon is q and ν is the energy of the virtual photon, which is transferred as the electron scatters. The

squiggle represents a virtual photon exchanged from the electron to the quark. The transverse spin of the target with respect to q, the momentum transfer, is S_T. The difference in momenta between the the two pions is given by R. A plane can be constructed by the incident and scattered electrons (denoted as scattering plane in Figure 1). The angle between the scattering plane and the transverse spin of the target with respect to q is given by ϕ_S. The 4-momentum of the scattered electron is ℓ' . The 4-momentum of the scattered quark is q. The transverse component of the two-pion momentum difference is R_T. The angle between the momentum of the pion-pair, P_h is the sum of the individual pions, P_1 and P_2 and the scattering plane is ϕ_h. The blue plane in Figure 1 is made by the the virtual photons and P_h. The pink plane, denoted as the two-hadron plane in Figure 1, is made by the trajectories of P_1 and P_2 in the two hadron rest frame. The azimuthal scattering angle, ϕ_R, is the angle between the two-hadron plane and the scattering plane and is calculated using Eq (2). My project looks for asymmetries in the azimuthal distribution of pion-pair production, which are sensitive to spin-orbit correlations of quarks in the proton.

$$\phi_R = \frac{\mathbf{q} \times \ell \cdot \mathbf{R_T}}{|q \times \ell \cdot R_T|} \cos^{-1} \frac{\mathbf{q} \times \ell \cdot \mathbf{q} \times \mathbf{R_T}}{|q \times \ell||q \times R_T|}. \tag{2}$$

Also of interest is the fraction of the virtual photon energy carried by the pion-pair,

$$z = \frac{E_h}{\nu}, 0 \leq z \leq 1, \tag{3}$$

the squared 4-momentum transfer is

$$Q^2 = 4EE' \sin^2 \theta/2 \tag{4}$$

and the fraction of the proton's momentum carried by the struck quark is

$$x = \frac{Q^2}{2M\nu} \tag{5}$$

where ν is the energy transfer of the electron to the quark. The fraction of the virtual photon energy carried by the pion-pair, $z = \frac{E_h}{\nu}$. The energy of hadron pair is E_h. The energy of the incident electron is E. The energy of the scattered electron is E'. The angle between the incident electron and scattered electron is θ.

II. Methodology

An asymmetry can be defined as:

$$A = \frac{N^+ - N^-}{N^+ + N^-}. \tag{6}$$

If we denote the two possible spin polarizations of the target as $\leftarrow$ and $\rightarrow$, N^+ corresponds to electron and target spins anti parallel $\rightarrow\leftarrow$ and N^- corresponds to electron and target spins parallel $\rightarrow\rightarrow$. The asymmetry in the azimuthal direction of pion-pair production depends on (x, Q^2, P_T^2, ϕ, z).

Of most interest to my project is the azimuthal scattering angle ϕ_R, which is calculated using Eq (2). The angle between the scattering plane, ϕ_R, and the pion center-of-mass plane (labeled "two-hadron plane" in Figure (2) is used to calculate the extracted structure function $g_1(x, P_T)$

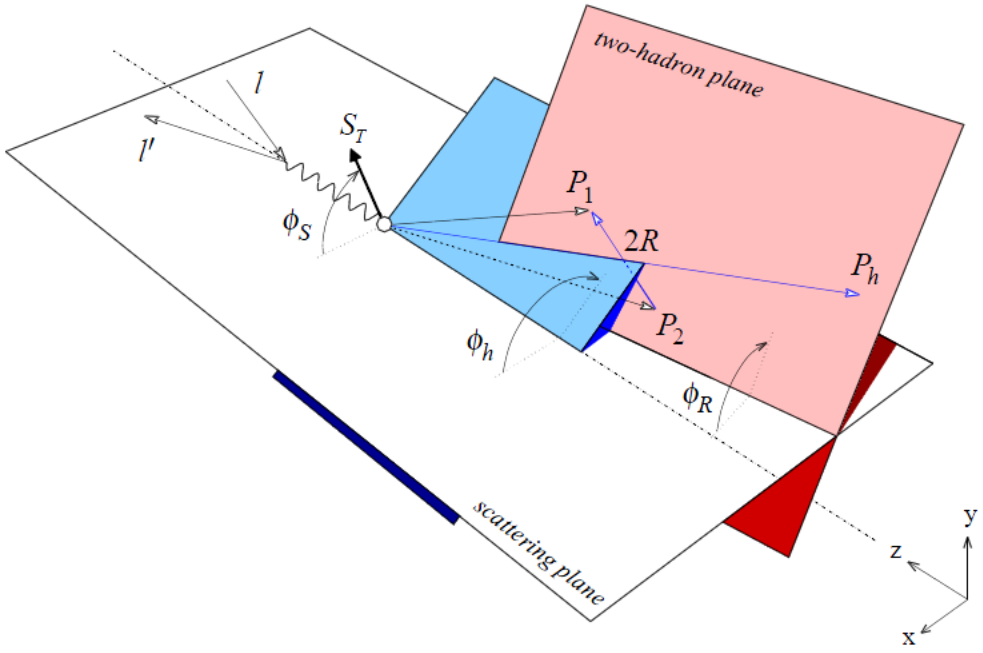


Figure 1: Schematic diagram of the experimental kinematics [1]

Eq (7) which is the sum over quark flavors, weighted by quark charge squared, of the difference in probability of finding a quark aligned and anti-aligned with the proton's spin. The structure function g_1 is a probability distribution of finding a quark with momentum fraction x. The extracted structure function can be expanded in terms of quark distribution functions as

$$g_1(x, P_T) = \sum_i e_i^2 [q_i^{\uparrow}(x, P_T) - q_i^{\downarrow}(x, P_T)]. \tag{7}$$

(where ($\uparrow$) denotes quark spin opposite (along) the proton spin.) If we write

$$q_i^{\uparrow} - q_i^{\downarrow} \equiv g_1(x)^q \tag{8}$$

where the superscript refers to quark flavor, i.e. u, d, s, then the observed structure functions are

$$F_{LU}^{sin\phi_R} = -x\frac{|R|sin\theta}{Q}[\frac{M}{M_h}xe^q(x, P_T)H^{\sphericalangle q}(z, cos\theta, M_h) + \frac{1}{z}f_1^q(x, P_T)G^{\sphericalangle q}(z, cos\theta, M_h)] \tag{9}$$

$$F_{UL}^{sin\phi_R} = -x\frac{|R|sin\theta}{Q}[\frac{M}{M_h}xe^q(x, P_T)H^{\sphericalangle q}(z, cos\theta, M_h) + \frac{1}{z}f_1^q(x, P_T)G^{\sphericalangle q}(z, cos\theta, M_h)] \tag{10}$$

$$F_{LL}^{cos\phi_R} = -x\frac{|R|sin\theta}{Q}\frac{1}{z}D^{\sphericalangle q}(z, cos\theta, M_h) \tag{11}$$

$$F_{LL} = xg_1^q(x, P_T)D_1^q(z, cos\theta, M_h) \tag{12}$$

$$F_{UU} = xf_1^q(x, P_T)D_1^q(z, cos\theta, M_h) \tag{13}$$

where the fragmentation functions, $H^{\sphericalangle q}(z)$ and $G^{\sphericalangle q}(z)$, are the probability of a struck quark hadronizing into a pion pair at particular z-values, $f_1^q(x, P_T)$ is the structure function of the unpolarized target and $e^q(x, P_T)$ is a structure function that measures quark gluon correlations. The masses of the proton and pion pair are M, M_h respectively. The structure functions $F_{LU}^{sin\phi_R}$, $F_{UL}^{sin\phi_R}$ and $F_{LL}^{cos\phi_R}$ are the electron scattering probabilities. We've limited ourselves to longitudinally polarized targets, and in order to extract $g_1(x, P_T)$ we're only interested in F_{LL} and $F_{LL}^{cos\phi_R}$. We don't know $g_1(x, P_T)^q$ and $D^{\sphericalangle q}$ explicitly, but data measured over a wide kinematic range can callow us to extract these from data. In order to calculate $g_1(x)$ and compare it to the theory, we need to check for a $cos\phi_R$ dependence in the asymmetry of ϕ_R. If the graph of the ϕ_R asymmetries shows a $cos\phi_R + constant$ dependance, then this supports the $g_1(x, P_T)$ from the di-hadron cross section formalism in Equations 8-12.

My project scans a large CLAS data set of processed experimental scattering events and histograms events and asymmetries as a function of the various kinematic variables. The CLAS data were taken in 2000-2001. The experiment's purpose was to extract the spin structure function, g_1. My project goes a step further to get the two dimensional function $g_1(x, P_T)$, which is the spin structure as a function of transverse momentum . While more data of this nature will be generated when the 12 GeV experiments start in 2 years, this older data set allows us to get preview of what's going on in this reaction.

My project analyzed the transverse momentum dependance (P_T^2) of the quark distributions which the transverse momentum dependent extracted structure function $g_1(x, p_T)$. Fitting asymmetries as a function of ϕ_R allows us to extract the product $g_1(x, p_T)H^{\angle}(z)$. If we know $H^{\angle}(z)$ from other experiments, we can extract $g_1(x, p_T)$.

III. Procedure

My project uses the ROOT analysis framework extensively. ROOT is an open source analysis software package developed at CERN that interprets C++ in order to visualize data. One of C++'s functionalities is its ability to plot graphs from user specified n-tuples. An n-tuple is a data structure that stores data in n-sized tuples. Tuples are a structure that contain data separated by commas and are advantageous since specific data can be accessed by index. This is executed using a C code called a root macro. My project began by creating ROOT macro routines to study the kinematic variables and asymmetries from one CLAS run. Once this was completed I applied these routines to the entire set of runs. The data set consisted of files with the same format which were concatenated using the shell script in Appendix 7.

In order to sort the raw CLAS data by polarity I used the C++ code in Appendix 4. The code in Appendix 4 searches each line of the data file for the helicity component and prints the line to an out file if the helicity is correct. Sorted data was then used to calculate kinematic variables using Appendix 1. To sort the kinematic variables into appropriate root n-tuples, I used the C++ code in Appendix 5 and read in the out file generated from running the code in Appendix 1. The code in Appendix 5 checks each line of the data file containing kinematic variables and prints lined containing the kinematic variable of interest into an output file.

The root macro in Appendix 2 was used to generate histograms of kinematic variables (see Figure 2,3). The C++ codes from Appendix 4 and 5 were used again to separate ϕ_R in both polarities. With this data and the root macro in Appendix 3, histograms of ϕ_R for both polarities are plotted as well as a plot of their asymmetries. With the same root macro a cosine fit is superimposed on the ϕ_R asymmetry plot.

IV. Analysis

Consideration of spin orientations and pion charge was not necessary in the calculation of the kinematic variables in Figure 2,3. Figures 2 and 3 are distributions of kinematic variables independent of spin orientation and pion-pair charges. A histogram of the ϕ_R distribution is given in h1. The bumps in the ϕ_R histogram only represent events accepted by the detector, thus efficiency varies for different angles. In h2 is the mass of the two-pions distribution. The bump around 0.8 GeV is characteristic of ρ meson production which occurs within that range. Graph h3 contains the θ center of mass distribution. The sudden drop after 1.5 is very curious. One would expect a drop in accepted events around 1.5 due to the geometry of the detector. This is rather low, but not necessarily indicative of bad data. The ν distribution in h4 is linear with a cut off of around 5 GeV as expected given the energy range of CLAS. Plot h5 shows the Q^2 distribution ranging from just below 1 to about 5 GeV, which is within the range of CLAS. The plot of x in h6 is within expected range of 0 to 1. The typical value is between .2 and .3, implying that the struck quark will have roughly a third of the proton momentum. The center of mass energy of the proton system, W in h7 drops around 3GeV as expected. In plot h8, the distribution of z is contained

between .1 and .9, which is to be expected given that it is a fraction of the energy transfer that comes from ν. The plot of missing mass in h9 falls around 3 GeV which is within expected range. The plot of R_T, the transverse component of the two-pion momentum difference in h10 is within expected range of 0 to 1. The lower cut off at 2, for W (h7), eliminates resonances that occur below 2 GeV. In order to limit my experiment to the case of deep inelastic where the scattering of just quarks and not the whole proton is considered, events beneath 2GeV must be neglected.

Figure 2: Part 1 of kinematic variable histograms

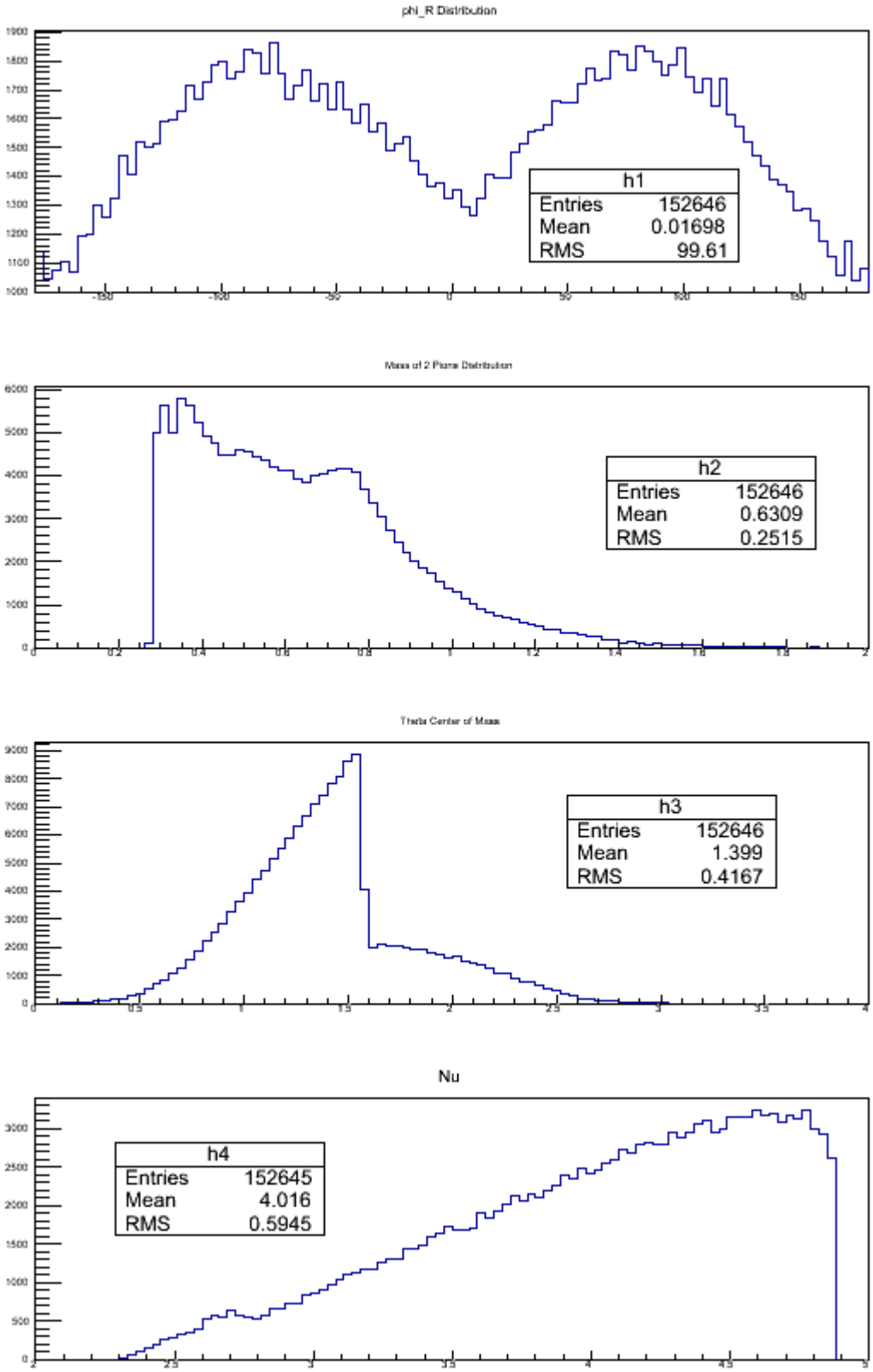
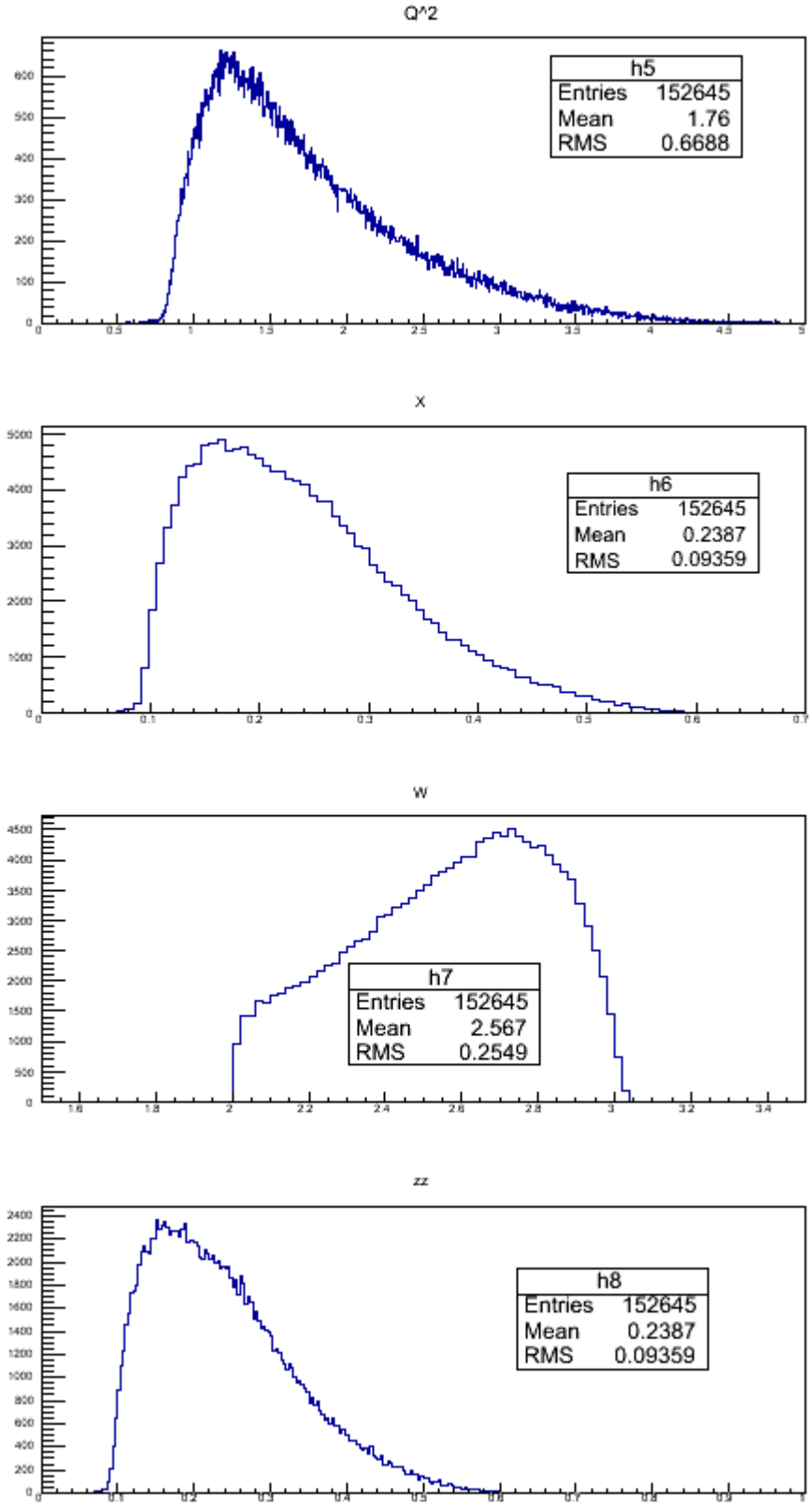


Figure 3: Part 2 of kinematic variable histograms



After checking the validity of the data by plotting the kinematic variables in Figures 2 and 3, the ROOT macro routine in Appendix 3 was used to plot the asymmetries of ϕ_R. Histograms of ϕ_R for each polarization are given along with a plot of the ϕ_R asymmetry.

The ϕ_R asymmetry in terms of the di-hadron cross section is given by

$$A = \frac{F_{LL} + F_{LL}^{cos\phi_R} cos\theta}{F_{UU}} = \frac{F_{LL}}{F_{UU}} + \frac{F_{LL}^{cos\phi_R} cos\theta}{F_{UU}} \tag{14}$$

where $\frac{F_{LL}}{F_{UU}}$ is the constant term in the fit of ϕ_R and $\frac{F_{LL}^{cos\phi_R} cos\theta}{F_{UU}}$ is the cosine-dependent term. If the plot of ϕ_R asymmetries (Figure 3) showed a clear Cosine distribution, this would support the model being tested. To calculate the ϕ_R asymmetry, the GetAsymmetry root method was used as shown in Appendix 3.

Unfortunately, the Cosine fit parameters are much too small to indicate a proper fit. A possible source of error that might affect a Cosine dependence in the ϕ_R asymmetry distribution are the charges of the pions. In order to account for pion charges, the C++ code in Appendix 6 was used to separate data by pion charge. For simplicity, we chose only cases where the pions had the same charge. The macro in Appendix 3 was used to plot the ϕ_R histograms and asymmetry for the charge-separated pions. The asymmetries for the charge-separated pions are shown in Figures 4 and 5.

As we can see, the ϕ_R asymmetries still do not show a clear cosine distribution in any case. According to Equation 13, we would have at least expected a noticeable constant term. Fortunately, the run info file showed that the half wave plate had not been accounted for. The half wave plate was used to check for when runtime data is taken in opposite spin orientation. Thus not accounting for the half wave plate flipped helicity in the data. The Asymmetry with the half wave plate taken into account is calculated using Apendix 8 and shown in Figure 7. The Asymmetry with the half wave plate in Figure 7 is .

Figure 4: ϕ_R separated by helicity and their Asymmetry

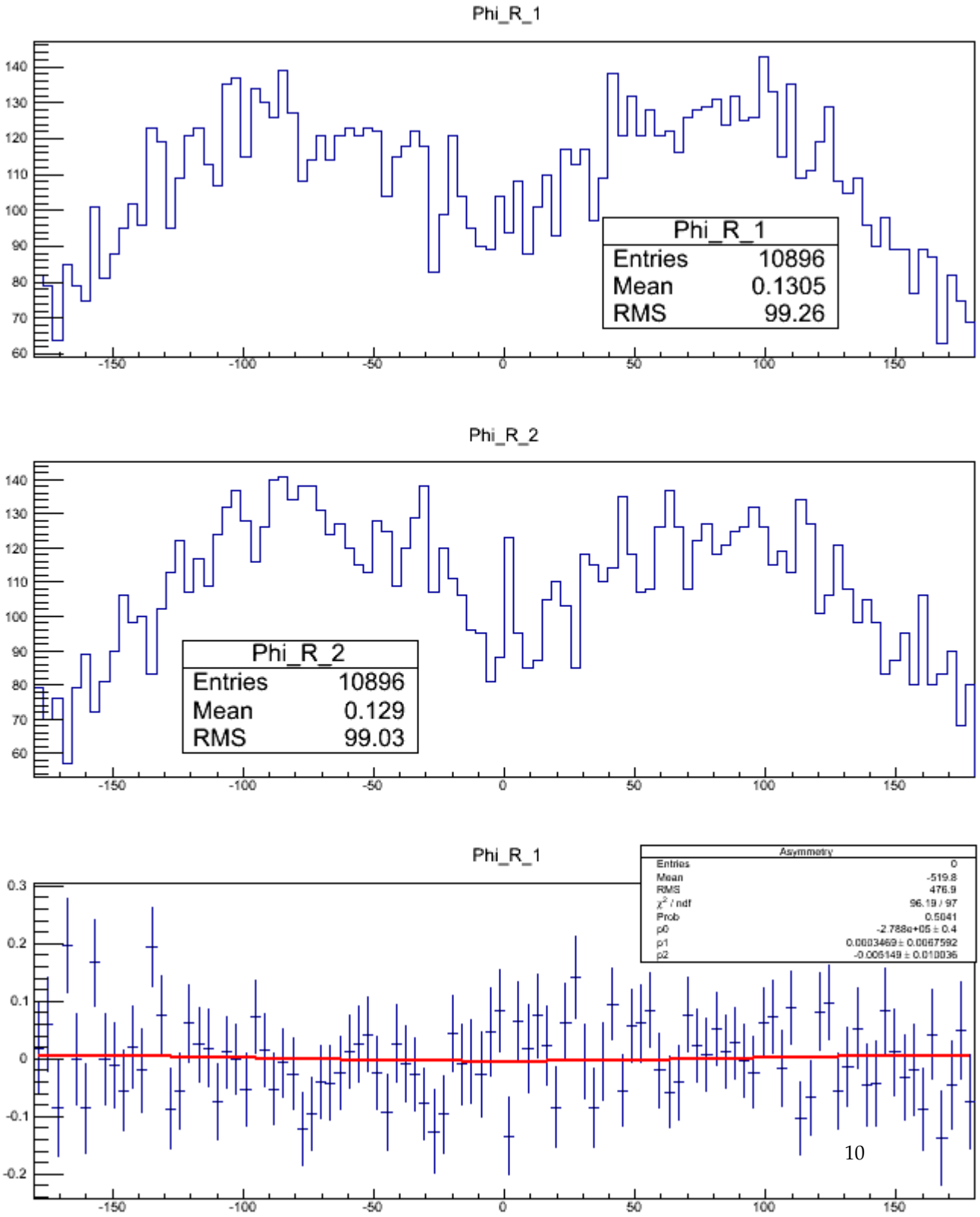


Figure 5: ϕ_R for positively charged pions separated by helicity and Asymmetry

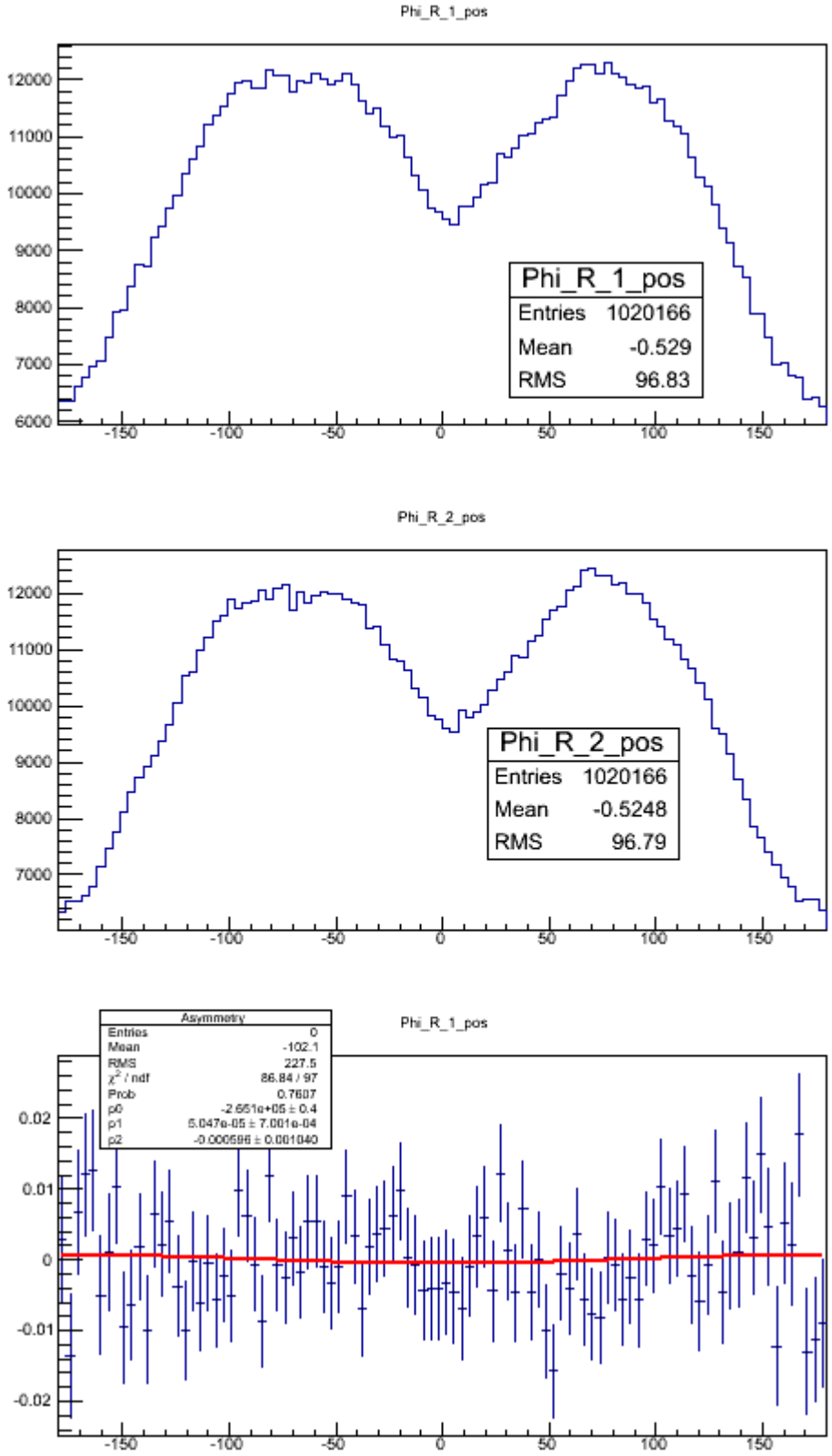


Figure 6: ϕ_R for negatively charged pions separated by helicity and Asymmetry

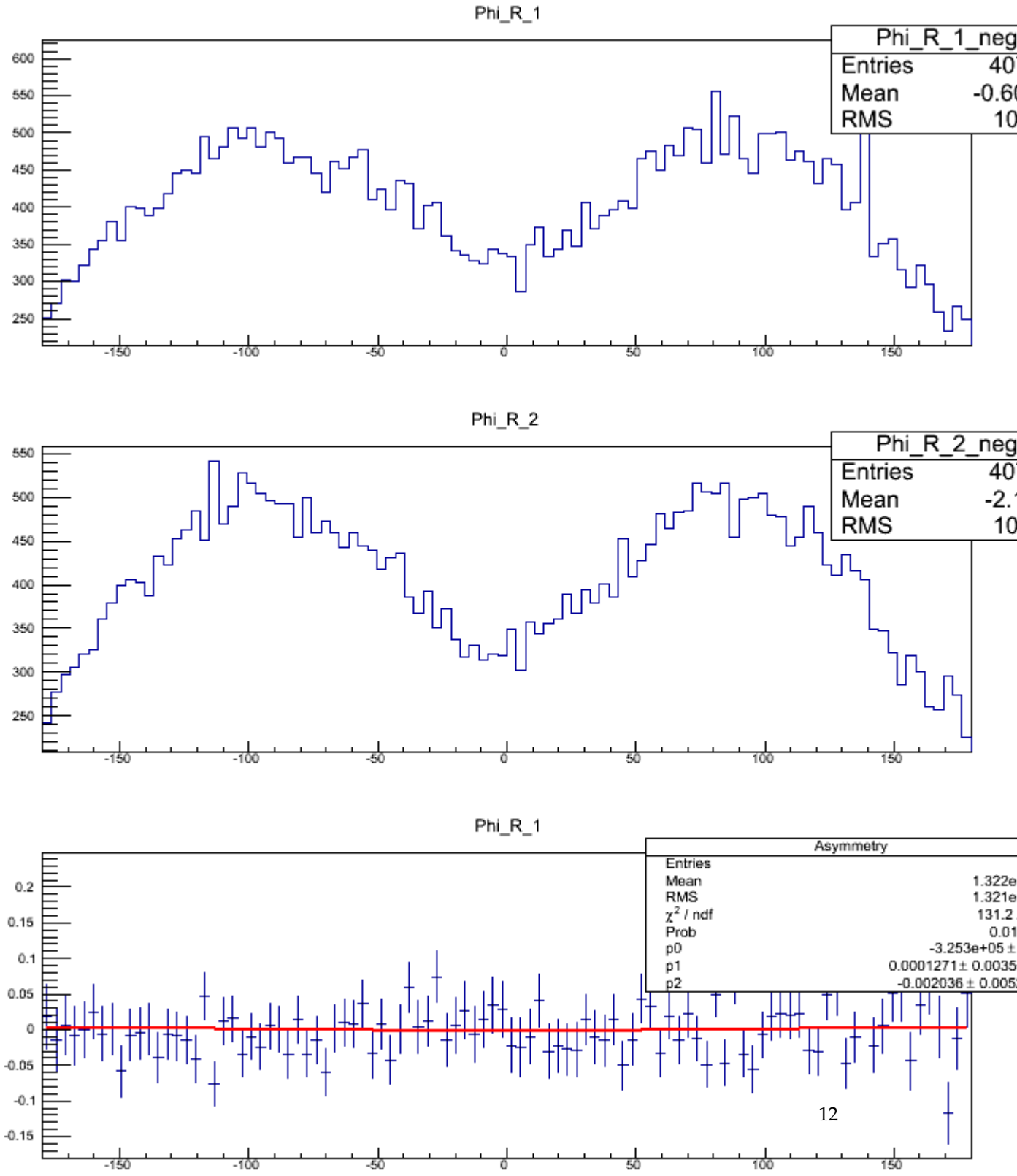
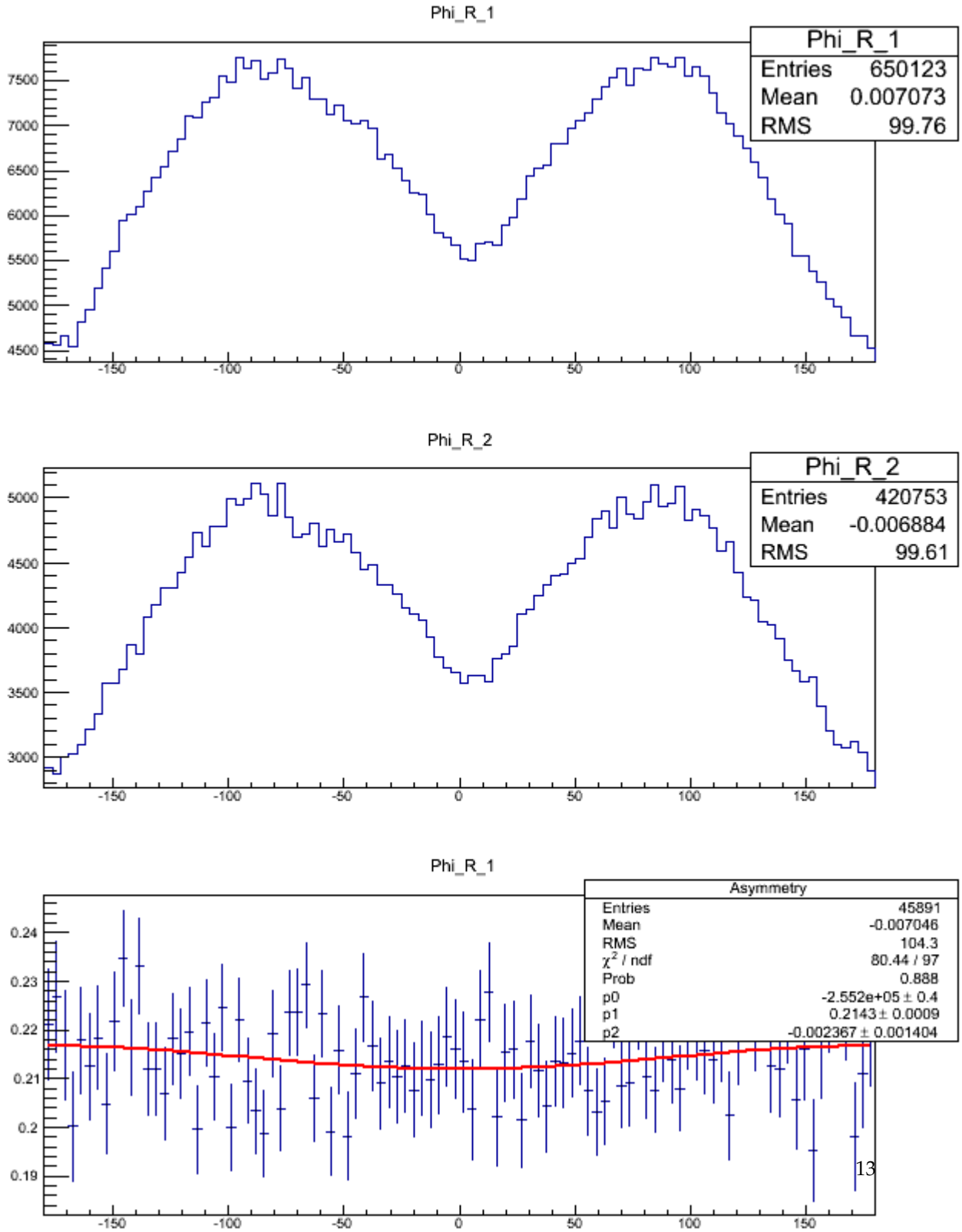


Figure 7: ϕ_R Asymmetry Half Wave Plate Accounted



V. Conclusion

The goal of my research was to check the validity of the data set and look for noticeable asymmetries in the ϕ_R distribution. It was shown that there is a ϕ_R asymmetry and it has a cosine dependance, which is in accordance with theory. The next step is to check the ϕ_R asymmetry's dependance on other kinematic variables. This could be accomplished by plotting cumulative asymmetry values that occur within specified ranges of the kinematic variable in question and plotting the asymmetries.

VI. Appendix 1: C++ code to calculate kinematic variables

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char **argv) {
   int helicity; // 1=positive helicity 2=negative helicity
   int picharge[2]; // charge of the two detected pions
   double beam_energy = 5.736; // beam energy in GeV
   double four_mom_beam[4]; // 4-momentum of the electron beam
   double four_mom_electron[4]; // 4-momentum of the electron in the lab frame
   double four_mom_pion_lab[2][4]; // 4-momentum of the 2 pions in the lab frame
   double four_mom_pion_cm[2][4]; // 4-momentum of the 2 pions in the cm frame
   double four_mom_pion_cm_polar[2][4]; // 4-momentum of the 2 pions in the cm frame in polar coord.
   double qq[4]; // 4-momentum transfer in lab frame
   double qq_hat[4]; // unit vector
   double qq_polar[4]; // qq in polar coordinates
   double four_mom_pair[4]; // 4-momentum of the pion pair in the lab frame
   double four_mom_pair_cm[4]; // 4-momentum of the pion pair in the cm frame
   double four_mom_diff[4]; // 4-momentum difference of the pion pair
   double four_mom_pair_hat[4]; // 4-momentum unit vector for pion pair
   double four_mom_diffT[4]; // 4-momentum difference with momentum perp to pair
   double tmp[4]; // temporary 4-momentum
   double tmp1[4]; // temporary 4-momentum
   double tmp2[4]; // temporary 4-momentum
   double tmp3[4]; // temporary 4-momentum
   int flag; // flag variable
   int counts[45][45]; // events with (theta,phi)
   int ii, jj;

   double nu, ww, xx, q2, zz; // electron Lorentz invariants
   double m2pi; // invariant mass of pion pair
   double phiR; // azimuthal angle
   double thetaCM1, thetaCM2; // pion theta in CM of pair

   double mass_electron = 0.000511; // mass of electron in GeV
   double mass_pi= 0.13957; // mass of charged pion in GeV
   double mass_pi0 = 0.13500; // mass of neutral pion in GeV
   double mass_proton = 0.93827; // mass of the proton in GeV
   double mass_missing; // missing mass for the reaction

   void zeroth(double*, double); // calculates 0th component of a 4-momentum from other components
   void diff4v(double*, double*, double*);  // difference of 2 4-vectors
```

```
   void sum4v(double*, double*, double*);  // sum of 2 4-vectors
   void scale4v(double*, double*, double); // scale a 4-vector by a constant
   void print4v(double*, char*); // print a 4-vector and a string
   double dot4v(double*, double*); // 4-vector dot product
   double dot3v(double*, double*); // 3-vector dot product
   void hat4v(double*, double*); // 4-vector normalized to the magnitude of the spacial part
   void polar4v(double*, double*); // convert a Cartesian 4-vector to polar coordinates
   void boost4v(double*, double*, double*); // Lorentz transform fm1 into frame where fm2 is at rest
   void cross4v(double*, double*, double*); // cross product of vector, product of 0th component
   void exchange4v(double*, double*); // exchance two four vectors

   four_mom_beam[0] = sqrt(mass_electron * mass_electron + beam_energy * beam_energy);
   four_mom_beam[1] = 0.0;
   four_mom_beam[2] = 0.0;
   four_mom_beam[3] = beam_energy;

   for(ii=0;ii<45;ii++) {
      for(jj=0;jj<45;jj++) {
         counts[ii][jj] = 0;
      }
   }

   while( scanf("%d %d %lf %lf %lf %lf %lf %lf %lf %lf %lf", &helicity, picharge, picharge+1,
      four_mom_electron+1, four_mom_electron+2, four_mom_electron+3,
      four_mom_pion_lab[0]+1, four_mom_pion_lab[0]+2, four_mom_pion_lab[0]+3,
      four_mom_pion_lab[1]+1, four_mom_pion_lab[1]+2, four_mom_pion_lab[1]+3)  != EOF ) {
//      printf("%g %g %g\n", four_mom_pion_lab[1][1], four_mom_pion_lab[1][2], four_mom_pion_lab[1][
      printf("\n");
      printf("pion charges = %d %d\n", picharge[0], picharge[1] );
      flag = 0;
      if(picharge[0] == 1 && picharge[1] == -1) {
         flag = 1;
      }
      if(picharge[1] == 1 && picharge[0] == -1) {
         exchange4v(four_mom_pion_lab[0],four_mom_pion_lab[1]);
         flag = 1;
      }
//      if(flag != 1) { continue; }

      zeroth(four_mom_electron, mass_electron);
      zeroth(four_mom_pion_lab[0], mass_pi);
      zeroth(four_mom_pion_lab[1], mass_pi);
//      printf("%g\n", four_mom_electron[0]);
      sum4v(four_mom_pion_lab[0], four_mom_pion_lab[1], four_mom_pair);
      diff4v(four_mom_pion_lab[0], four_mom_pion_lab[1], four_mom_diff);
```

```
      m2pi = sqrt(dot4v(four_mom_pair,four_mom_pair));
      diff4v(four_mom_beam, four_mom_electron, qq);
      hat4v(qq, qq_hat);
      polar4v(qq,qq_polar);
      nu = qq[0];
      q2 = -dot4v(qq,qq);
      xx = q2 / 2 / mass_proton / nu;
      ww = sqrt( mass_proton * mass_proton + 2 * mass_proton * nu - q2 );
      zz = four_mom_pair[0]/nu;
//        calculate missing mass
      diff4v(qq, four_mom_pair, tmp);
      tmp[0] += mass_proton;
      mass_missing = sqrt(dot4v(tmp,tmp));
      x_feynman1 = 2 * dot3v(four_mom_pion_cm1,qq_hat) / ww;
      x_feynman2 = 2 * dot3v(four_mom_pion_cm2,qq_hat) / ww;

//
      printf("zz, mm =  %g %g\n", zz, mass_missing);
      printf("\n");
      printf("nu, q2 =  %g %g\n", nu, q2);
      printf("\n");
      printf("xx, ww = %g %g\n",  xx, ww);
      printf( x_feynman1,"x_feynman1=");
      printf(x_feynman2, "x_feynman2=");
//      printf("qq= %g %g %g %g\n", qq[0], qq[1], qq[2], qq[3]);
      print4v(qq,"qq=");
      print4v(qq_hat,"qq_hat=");
      print4v(qq_polar,"qq_polar=");
      print4v(four_mom_electron,"kprime=");
      print4v(four_mom_pion_lab[0],"p_pi1=");
      print4v(four_mom_pion_lab[1],"p_pi2=");
      print4v(four_mom_pair,"p_pair=");
      printf("m2pi= %g\n", m2pi);
      boost4v(four_mom_pair, four_mom_pair, four_mom_pair_cm);
      print4v(four_mom_pair_cm, "four_mom_pair_cm");
      boost4v(four_mom_pion_lab[0], four_mom_pair, four_mom_pion_cm[0]);
      boost4v(four_mom_pion_lab[1], four_mom_pair, four_mom_pion_cm[1]);
      print4v(four_mom_pair,"p_pair=");
      print4v(four_mom_pion_cm[0],"4-mom-pion-cm1= ");
      print4v(four_mom_pion_cm[1],"4-mom-pion-cm2= ");
      polar4v(four_mom_pion_cm[0], four_mom_pion_cm_polar[0]);
      polar4v(four_mom_pion_cm[1], four_mom_pion_cm_polar[1]);
      print4v(four_mom_pion_cm_polar[0],"4-mom-pion-cm-polar1= ");
      print4v(four_mom_pion_cm_polar[1],"4-mom-pion-cm-polar2= ");
      print4v(four_mom_diff, "pion difference= ");
```

```
      hat4v(four_mom_pair,four_mom_pair_hat);
      print4v(four_mom_pair_hat,"4-mom-pair-hat= ");

      thetaCM1 = acos( dot3v(four_mom_pion_cm[0], four_mom_pair_hat) /
         sqrt(dot3v(four_mom_pion_cm[0], four_mom_pion_cm[0]) ) );
      printf("thetaCM1 = %g radians, %g degrees\n", thetaCM1, thetaCM1*180/3.14159);

      thetaCM2 = acos( dot3v(four_mom_pion_cm[1], four_mom_pair_hat) /
         sqrt(dot3v(four_mom_pion_cm[1], four_mom_pion_cm[1]) ) );
      printf("thetaCM2 = %g radians, %g degrees\n", thetaCM2, thetaCM2*180/3.14159);

      scale4v(four_mom_pair_hat, tmp, dot3v(four_mom_diff, four_mom_pair_hat) );
      diff4v(four_mom_diff, tmp, four_mom_diffT);
      print4v(four_mom_diffT, "R_T= ");
      scale4v(four_mom_diffT, four_mom_diffT, 0.5);
      print4v(four_mom_diffT, "R_T= ");

      cross4v(qq,four_mom_beam, tmp1);
      cross4v(qq,four_mom_diffT, tmp2);
//      tmp1[0] = tmp2[0] = 0.0;
      phiR = acos( dot3v(tmp1,tmp2) / sqrt(dot3v(tmp1,tmp1)) / sqrt(dot3v(tmp2,tmp2)) );
      if( dot3v(tmp1,four_mom_diffT) < 0 ) {
         phiR = - phiR;
      }
      printf("phiR = %g radians, %g degrees\n", phiR, phiR*180/3.14159);
      printf("R_T dot 4-mom-pair= %g\n", dot3v(four_mom_diffT,four_mom_pair) );

      ii = (int) (thetaCM1 * 45 / 3.14159 );
      jj = (int) ( phiR * 22.5 / 3.14159 + 22.5 );
      counts[ii][jj]++;
      printf("ii, jj= %d %d\n", ii, jj);
   }
   for(ii=0;ii<45;ii++) {
      fprintf(stderr,"\n");
      for(jj=0;jj<45;jj++) {
         fprintf(stderr, "%3d ", counts[ii][jj] );
      }
   }
}

// calculates the 0th component of a 4-momentum given the 3-momentum and a mass assuming the particl
// is on-shell

void zeroth(double *four_momentum, double mass) {
```

```
   four_momentum[0] = sqrt( mass * mass + four_momentum[1] * four_momentum[1]
      + four_momentum[2] * four_momentum[2] + four_momentum[3] * four_momentum[3]);
   return;
}

// calculates the sum of two 4-vectors: fv3 = fv1 - fv2

void sum4v(double *fv1, double *fv2, double *fv3) {
   int ii;
   for(ii=0;ii<4;ii++) {
      fv3[ii] = fv1[ii] + fv2[ii];
   }
   return;
}

// calculates the difference between two 4-vectors: fv3 = fv1 - fv2

void diff4v(double *fv1, double *fv2, double *fv3) {
   int ii;
   for(ii=0;ii<4;ii++) {
      fv3[ii] = fv1[ii] - fv2[ii];
   }
   return;
}

// print a 4-vector

void print4v(double *fv, char *string) {
   printf("%s %g %g %g %g\n", string, fv[0], fv[1], fv[2], fv[3]);
   return;
}

// 4-vector dot product

double dot4v(double *fv1, double *fv2) {
   return fv1[0]*fv2[0] - fv1[1]*fv2[1] -  fv1[2]*fv2[2] -  fv1[3]*fv2[3];
}


// normalize a 4-vector fv1 such that the spacial components are a unit vector

void hat4v(double *fv1, double *fv2) {
   double norm;
   int ii;
```

```
   norm = sqrt( fv1[1] * fv1[1] + fv1[2] * fv1[2] + fv1[3] * fv1[3] );
   for(ii=0;ii<4;ii++) {
      fv2[ii] = fv1[ii] / norm;
   }
   return;
}


// convert a Cartesian 4-vector into polar coordinates
// fv2[1] = r; fv2[2] = theta; fv2[3] = phi

void polar4v(double *fv1, double *fv2) {
   fv2[0] = fv1[0];
   fv2[1] = sqrt(  fv1[1] * fv1[1] + fv1[2] * fv1[2] + fv1[3] * fv1[3] );
   fv2[2] = acos( fv1[3] / fv2[1] );
   fv2[3] = atan2( fv1[2], fv1[1] );
}

// Lorentz boost Particle 1 with 4-momentum fm1 into the frame where Particle 2
// with 4-momentum fm2 is at rest; fm3 is the boosted 4-momentum

void boost4v(double *fm1, double *fm2, double *fm3) {
   double gamma, beta; // Lorentz boost parameters
   double direction[4]; // unit vector in the direction of the boost
   double magnitude; // magnitude of fm2 vector
   double fm_para; // parallel component
   double dot4v(double*, double*);
   int index; // loop index

   gamma = fm2[0] / sqrt( dot4v(fm2,fm2) );
   beta = sqrt( 1 - 1/gamma/gamma);
   printf("boost(): gamma, beta= %g %g\n", gamma, beta);
   magnitude = sqrt( fm2[1] * fm2[1] + fm2[2] * fm2[2] + fm2[3] * fm2[3] );
   printf("boost(): direction ");
   for(index=0;index<4;index++) {
      direction[index] = fm2[index] / magnitude;
      printf("%g ", direction[index]);
   }
   printf("\n");
   fm_para = fm1[1] * direction[1] + fm1[2] * direction[2] + fm1[3] * direction[3];
   fm3[0] = gamma * ( fm1[0] - beta * fm_para );
   for( index=1; index<4; index++) {
      fm3[index] = fm1[index] + ( gamma * ( fm_para - beta * fm1[0]) - fm_para ) * direction[index];
   }
   return;
```

```
}

// Perform a cross product fv1 and fv2 spatial components while multiplying
// together the time-like components and putting the result in fv3

void cross4v(double *fv1, double *fv2, double *fv3) {
   fv3[0] = fv1[0] * fv2[0];
   fv3[1] = fv1[2] * fv2[3] - fv2[2] * fv1[3];
   fv3[2] = fv1[3] * fv2[1] - fv2[3] * fv1[1];
   fv3[3] = fv1[1] * fv2[2] - fv2[1] * fv1[2];
   return;
}

// fv2 = fv1 * constant

void scale4v(double *fv1, double *fv2, double constant) {
   int index;

   for(index=0; index<4; index++) {
      fv2[index] = fv1[index] * constant;
   }
   return;
}


// 3-vector dot product of the spatial part of 4-vectors fv1 and fv2

double dot3v(double *fv1, double *fv2) {
   return fv1[1] * fv2[1] + fv1[2] * fv2[2] + fv1[3] * fv2[3];
}


// exchange fv1 and fv2

void exchange4v(double *fv1, double *fv2) {
   double tmp[4];
   int ii;

   for(ii=0;ii<4;ii++) {
      tmp[ii] = fv1[ii];
   }
   for(ii=0;ii<4;ii++) {
      fv1[ii] = fv2[ii];
   }
   for(ii=0;ii<4;ii++) {
```

```
      fv2[ii] = tmp[ii];
   }
   return;
}
```

VII. Appendix 2: Kinematic variable histogram macro

```
{

gROOT->Reset();

FILE *fp = fopen("Data.txt","r");
 FILE *fpp = fopen("Data1.txt","r");
 FILE *fp3 = fopen("DatathetaCM1.txt","r");
 FILE *fp4 = fopen("kinvarsnu.txt","r");
 FILE *fp5 = fopen("kinvarsxx.txt","r");
 FILE *fp55 = fopen("kinvarszz.txt","r");
 FILE *fp6 = fopen("R_t_sorted.txt","r");
 FILE *fp7 = fopen("x_feynman_1.txt","r");
FILE *fp8 = fopen("phiR_1.txt","r");
FILE *fp9 = fopen("phiR_2.txt","r");




 Float_t x,y,z,n,w,a,b,c,d,e,i,g,h,m;
Int_t ncols;
Int_t ncols2;
Int_t ncols3;
Int_t ncols4;
Int_t ncols44;
Int_t ncols_xx;
Int_t ncols_zz;
Int_t ncols_4mom;
Int_t ncols_xf;
Int_t ncols_xff;


char var1[132];
Int_t nlines = 0;
Int_t nlines2 = 0;
Int_t nlines3 = 0;
Int_t nlines4 = 0;
Int_t nlines44 = 0;
Int_t nlines_xx = 0;
Int_t nlines_zz = 0;
Int_t nlines_4mom = 0;
 Int_t nlines_xf = 0;
Int_t nlines_xff = 0;
```

```
TFile *f = new TFile("basic.root","RECREATE");

TH1F *h1 = new TH1F("h1","phi_R Distribution",100,-180,180);
TH1F *h2 = new TH1F("h2","Mass of 2 Pions Distribution",100,0,2);
TH1F *h3 = new TH1F("h3","Theta Center of Mass",100,0,4);
TH1F *h4 = new TH1F("h4","Nu",100,2,5);
TH1F *h5 = new TH1F("h5","Q^2",1000,0,5);
TH1F *h6 = new TH1F("h6","X",100,0,0.7);
TH1F *h7 = new TH1F("h7","W",100,1.5,3.5);
TH1F *h8 = new TH1F("h8","zz",300,0,1);
TH1F *h9 = new TH1F("h9","Missing_Mass",100,0,3);
TH1F *h10 = new TH1F("h10","R_T (3-Momentum Vector Magnitude)",1000,0,1.5
);
TH1F *h11 = new TH1F("h11","x_feynman_1",100,0,3);

TNtuple *ntuple = new TNtuple("ntuple","data from ascii file","x:y:z:n:w");
while (1) {
  ncols = fscanf(fp,"%s %s %s %s %f %s",var1,var1,var1,var1, &x, var1);
if (ncols < 0) break;

h1->Fill(x);

nlines++;

  ncols2 = fscanf(fpp,"%s %f",var1,&y);

if (ncols2 < 0) break;

h2->Fill(y);
 nlines2++;

 ncols3 = fscanf(fp3,"%s %s %f %s %s %s",var1,var1,&z,var1, var1, var1);

if (ncols3 < 0) break;

h3->Fill(z);
 nlines3++;

 ncols4 = fscanf(fp4,"%s %s %s %f %f",var1,var1,var1,&n,&w);

 if (ncols4 < 0) break;

 h4->Fill(n);
```

```
 h5->Fill(w);
 nlines4++;


 ncols_xx = fscanf(fp5,"%s %s %s %f %f",var1,var1,var1,&a,&b);
 h6->Fill(a);
 h7->Fill(b);
 ncols_xx++;
 ncols_zz = fscanf(fp55,"%s %s %s %f %f",var1,var1,var1,&c,&d);
 h8->Fill(c);
 h9->Fill(d);
 ncols_zz++;
 ncols_4mom = fscanf(fp6,"%s %f %f %f %f",var1,&e,&i,&g,&h);

 m = sqrt( i*i + g*g + h*h);
 if (nlines3 < 5) printf("R_T=%8s %g",var1,d);
 h10->Fill(m);
 ncols_4mom++;


}
printf(" found %d points
",nlines3);
fclose(fp);
fclose(fpp);
fclose(fp3);
fclose(fp4);
 fclose(fp5);
fclose(fp6);
f->Write();

TCanvas *c1 = new TCanvas("c1","c1",50,50,800,600);

 c1->Divide(2,5);
 c1->cd(1);

 h1->Draw();
 c1->cd(2);
 h2->Draw();
 c1->cd(3);
 h3->Draw();
 c1->cd(4);
 h4->Draw();
 c1->cd(5);
 h5->Draw();
```

```
 c1->cd(6);
 h6->Draw();
 c1->cd(7);
 h7->Draw();
 c1->cd(8);
 h8->Draw();
 c1->cd(9);
h9->Draw();
 c1->cd(10);
 h10->Draw();
 }
}
```

VIII. Appendix 3: ϕ_R separated by helicity and their Asymmetry

```
{
gROOT->Reset();

FILE *fp11 = fopen("phiR_1.txt","r");
FILE *fp12 = fopen("phiR_2.txt","r");

 Float_t x,pr1,pr2,asymmetry,err,bin;

Int_t ncols_phiR1;
Int_t ncols_phiR2;

char var1[132];

Int_t nlines_phiR1 = 0;
Int_t nlines_phiR2 = 0;

TFile *f = new TFile("basic.root","RECREATE");

TH1F *h1 = new TH1F("Phi_R_1"," Phi_R_1",100,-3.14,3.14);
TH1F *h2 = new TH1F("Phi_R_2","Phi_R_2",100,-3.14,3.14);
TH1F *Asymmetry = new TH1F("Phi_R Asymmetry"," Phi_R Asymmetry ",100,-3.14,3.14);

 TF1 *fit = new TF1("fit","[1] + [2]*cos(x*((3.14159)/180))");
fit->SetParLimits(1, -180, 180);

TNtuple *ntuple = new TNtuple("ntuple","data from ascii file","x:y:z:n:w");
while (nlines_phiR2 < 10896) {

ncols_phiR1 = fscanf(fp11,"%s %s %s %s %f %s",var1,var1,var1,var1, &pr1, var1);
ncols_phiR2 = fscanf(fp12,"%s %s %s %s %f %s",var1,var1,var1,var1, &pr2, var1);

 h1->Fill(pr1);
h2->Fill(pr2);


 Asymmetry = (TH1F*) h1->GetAsymmetry(h2);

nlines_phiR1++;
 nlines_phiR2++;

}

fclose(fp11);
```

```
 fclose(fp12);
f->Write();

TCanvas *c1 = new TCanvas("c1","c1",50,50,800,600);

 c1->Divide(1,3);

 c1->cd(1);
 h1->Draw();
c1->cd(2);
 h2->Draw();
c1->cd(3);
 Asymmetry->Fit("fit");
Asymmetry->SetName("Asymmetry");
 Asymmetry->Draw();
 }
```

IX. Appendix 4: C++ code to sort data by helicity

```
//
//  main.cpp
//  r27070.twopi split
//
//  Created by Wyatt Meldman-Floch on 3/26/13.
//  Copyright (c) 2013 Wyatt Meldman-Floch. All rights reserved.
//
//  Separates raw data by polarity. The second charcter in each raw data
//  line contains the polarity.
//
//

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

using namespace std;

int main() {

    fstream in;
    ofstream out;

    in.open("/Users/Wyatt/Desktop/seniorproject/twopi.big");
    out.open("/Users/Wyatt/Desktop/seniorproject/root/twopi.big_2");

    string search1 = "1";

    string line;

    while(in.good()) {
        getline (in,line,'\n');

        if (line[1] == '2')
        {
            out << line <<endl;
        }

        }


    in.close();
```

```
    out.close();

    return 0;

}
```

X. Appendix 5: C++ code to sort variables from data set

```
//
//  main.cpp
//  hist_reader
//
//  Created by Wyatt Meldman-Floch on 1/11/13.
//  Copyright (c) 2013 Wyatt Meldman-Floch. All rights reserved.
//
//  Extracts relevent data from the sorted data file.
//

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

using namespace std;

int main() {

    fstream in;
    ofstream out;

    in.open("/Users/Wyatt/Desktop/seniorproject/twopi.big_2.out");
    out.open("/Users/Wyatt/Desktop/seniorproject/root/phiR_2_big.txt");

    string search1 = "phiR";

    string line;

    while(in.good()) {
        getline (in,line,'\n');
        if (line.find(search1) != string::npos)
        {
            out << line <<endl;
        }
    }

    in.close();
    out.close();

    return 0;
}
```

XI. Appendix 6: C++ code to sort raw data set by charge

```
//
//  main.cpp
//  pion_charge_split
//
//  Created by Wyatt Meldman-Floch on 4/18/13.
//  Copyright (c) 2013 Wyatt Meldman-Floch. All rights reserved.
//


#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

using namespace std;

int main() {

    fstream in;
    ofstream out;

    in.open("/Users/Wyatt/Desktop/seniorproject/twopi.big");
    out.open("/Users/Wyatt/Desktop/seniorproject/root/twopi.big_neg");

    string search1 = "1";

    string line;

    while(in.good()) {
        getline (in,line,'\n');

        if (line[3] == '-')
        {
        if (line[4] == '1')
            if (line[6] == '-')
            {
        {if (line[7] == '1'){

            out << line <<endl;}
```

```
        }

            }}}


    in.close();
    out.close();

    return 0;

}
```

XII. Appendix 7: Shell script to concatenate data sets

```
#Shell script used to contatenate all datafiles in one file


rm ../twopi.big

cat r27* > ../twopi.big
```

XIII. Appendix 8: ϕ_R Asymmetry with half wave plate

```
{
gROOT->Reset();
FILE *fp11 = fopen("dplate_1.phir","r");
FILE *fp12 = fopen("dplate_2.phir","r");
 Float_t x,pr1,pr2,asymmetry,err,bin;
Int_t ncols_phiR1;
Int_t ncols_phiR2;
char var1[132];
Int_t nlines_phiR1 = 0;
Int_t nlines_phiR2 = 0;
TFile *f = new TFile("basic.root","RECREATE");
TH1F *h1 = new TH1F("Phi_R_1"," Phi_R_1",100,-180,180);
TH1F *h2 = new TH1F("Phi_R_2","Phi_R_2",100,-180,180);
 TH1F *Asymmetry = new TH1F("Phi_R Asymmetry"," Phi_R Asymmetry no dplate",100,-180,180);
 TF1 *fit = new TF1("fit","[1] + [2]*cos(x*((3.14159)/180))");
fit->SetParLimits(1, -180, 180);
TNtuple *ntuple = new TNtuple("ntuple","data from ascii file","x:y:z:n:w");




 while(fscanf(fp11,"%s %s %s %s %f %s",var1,var1,var1,var1, &pr1, var1) != EOF){
 h1->Fill(pr1);
nlines_phiR1++;}

while( fscanf(fp12,"%s %s %s %s %f %s",var1,var1,var1,var1, &pr2, var1) != EOF){
 h2->Fill(pr2);
 nlines_phiR2++;}

Asymmetry = (TH1F*) h1->GetAsymmetry(h2);
fclose(fp11);
 fclose(fp12);
f->Write();
TCanvas *c1 = new TCanvas("c1","c1",50,50,800,600);
 c1->Divide(1,3);
 c1->cd(1);
 h1->Draw();
c1->cd(2);
 h2->Draw();
c1->cd(3);
 Asymmetry->Fit("fit");
Asymmetry->SetName("Asymmetry");
```

```
Asymmetry->Draw(); }
```

References

[1] Sergio Anefalos Pereira (INFN - Frascati), DPWG meeting 12/10/2012

[2] A. Adare et al. (PHENIX Collaboration). Cross Section and Parity-Violating Spin Asymmetries of W Boson Production in Polarized p p Collisions at $sqrt[s] = 500$ GeV. Physical Review Letters 106, 6 2011

[3] M. M. Aggarwal et al. (STAR Collaboration). Measurement of the Parity-Violating Longitudinal Single-Spin Asymmetry for W Boson Production in Polarized Proton-Proton Collisions at $\sqrt[s]{=}$500 GeV. Physical Review Letters 106, 6 2011