

MEASUREMENT OF A DOUBLE-SPIN ASYMMETRY  
FROM INCLUSIVE ELECTRON-PROTON SCATTERING IN  
THE CLAS EG4 EXPERIMENT

By: Christopher Kallian

Advisor: Dr. Fersch

# **Table of Contents**

## ***(0.0) Abstract:***

## ***(1.0) Introduction:***

- (1.1): Background
- (1.2): Motivation
- (1.3): Kinematics

## ***(2.0) Methods:***

- (2.1): Software Requirements
- (2.2): Collaboration
- (2.3): Processes
- (2.4): Flow Chart
- (2.5): Code Appendix

## ***(3.0) Results and Conclusions:***

- (3.1): Data Found

# **Abstract**

This experiment is a data analysis of the results from the CLAS EG4 experiment, performed at Jefferson Labs over ten years ago. These results were recorded in a C++/ROOT file on a work disk TTree, where a TTree is a data structure that simulates a hierarchical tree structure, where an ordered structure of nodes is created, where a node can either be a “root” node, which is a node on the top of the hierarchy, a “branch” node, which is a node that is beneath the root node, but still has “child” nodes, which are known as the “leaves” in the tree. By using C++/ROOT, this experiment iterated through the TTree and most of its events, taking in the data required to plot a calculated Asymmetry vs the Lorentz Invariant Missing mass in a given binning scheme as well as a 2-Dimensional histogram comparing the 4-momentum of the virtual photon squared to the Lorentz Invariant Missing Mass.

To do this, two scripts must be created so that the code may analyze the data so that it may be used to create the graphs. The first script is named the “.h” script, and only serves as a mean of initialization for the TTree itself, taking the data and placing them in “Branches” within the tree. The second script is called the “.C” script, and takes the data from the TTree and runs the processes required to find the values necessary to create the graphs. Once the data has been taken, it is then filtered through many if-statements, so that the only data values that are taken in are from electrons and not from the extra particles that were created during the collision. Additionally, some of the filters also allow us to determine the difference between electrons polarized with or against the ammonia target, allowing us to add to our  $n^+$  (polarized with) and  $n^-$  (polarized against) counts, enabling the construction of an asymmetry.

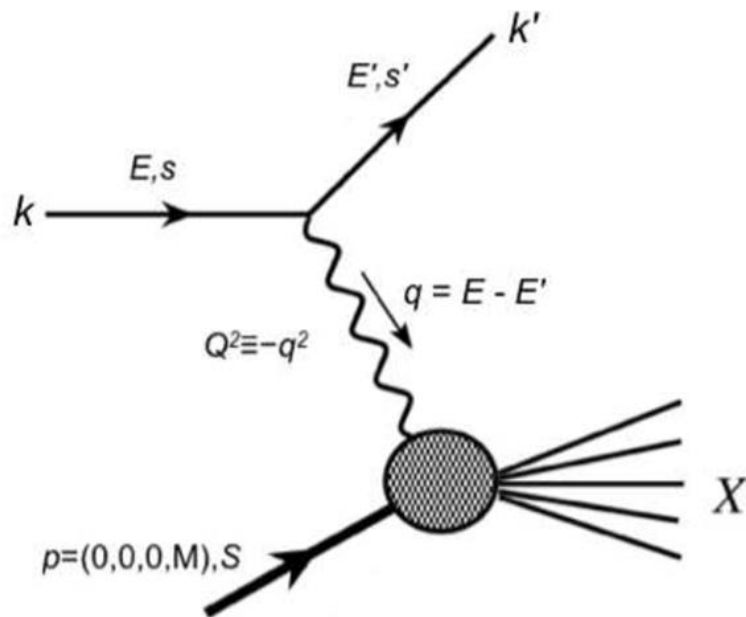
# **Introduction**

## **(1.1)**

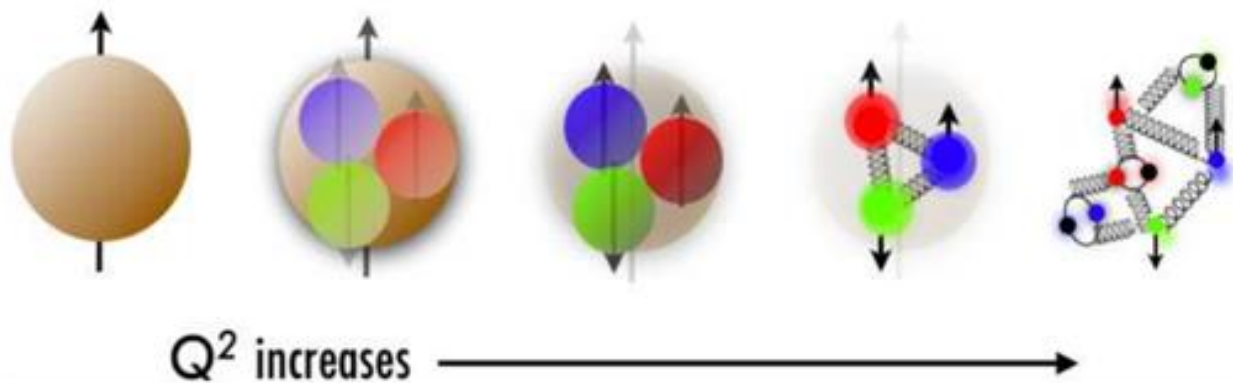
### **Background**

Several years ago, Jefferson Lab performed the EG4 Experiment in Hall-B and following the experiment, most of the data was collected and saved into a Tree. A good majority of the EG4 data has been published, but not all of it, for that the inclusive proton data is still in need of some work. This is where my analysis comes in, where I chose to create a 2-Dimensional histogram that will be comparing the calculated values of  $Q^2$  vs  $W$ , as well as creating a histogram that plots an Asymmetry to a calculated  $W$  value in a given binning scheme.

The variable  $Q^2$  is the squared value of the 4-Dimensional vector momentum of gamma\*, the virtual photon that is exchanged between the electron and proton during the scattering (Figure 1.10). As  $Q^2$  increases, the model of the proton begins to turn from a regular proton, to the 3 valence quarks, to an assembly of quarks, gluons, and quark anti-quark pairs (see Figure 1.11). The variable  $W$  represents the Lorentz-invariant missing mass (Lorentz-Invariant meaning same in all reference frames) from the collision. Both  $Q^2$  and  $W$  are both functions of  $p$ ,  $\theta$ , and  $E$ , where  $p$  is the momentum of the polarized electron beam and  $\theta$  is the scattering angle of the electron, and  $E$  is the incident energy. One value for the electron-beam energy will be used in the creation of these histograms.



**Figure 1.10:** Here we can see an example of the scattering, where an electron with some 4-momentum  $k$ , energy  $E$  and some spin  $S$  exchanges a virtual photon of 4-momentum  $q$ . Following the exchange, the electron scatters off in some direction  $\theta$  with a 4-momentum  $k'$ , scattering energy  $E'$  and a spin  $S'$ .



**Figure 1.11:** Here we see what the model of the proton looks like, in terms of the 4-momentum transferred by the electron squared. This is caused by Heisenberg uncertainty principle, because as our momentum (and uncertainty in momentum) increases, our uncertainty in position decreases.

## (1.2)

### Motivation:

During my time at Jefferson Lab (Summer 2016), I spent a good portion of my time creating 1 dimensional histograms with other variables (such as momentum, theta, etc.). Eventually, I began to work on 2-Dimensional histograms, plotting out other basic variables against each other (such as “X” vs “W”). Over the course of my time working on my capstone, I was able to create several different histograms, such as a 1-Dimensional histogram of  $Q^2$ , a 1-Dimensional histogram of W, a 2-Dimensional histogram comparing  $Q^2$  and W, two 1-dimensional histograms, one with the number of electrons polarized with the target, and other with the number of electrons polarized against the target. Finally, I finished by being able to create an asymmetry comparison to W in a given binning scheme for one run.

## (1.3)

### Kinematics:

In the EG4 experiment, polarized electrons are fired at a polarized ammonia ( $^{15}\text{NH}_3$ ) target. Scattering electrons off these protons allows us to get a better understanding of the proton and spin structure.

Taking the electron scattering with incident energy  $E$  on a target proton with mass  $M$  we see a scattering, with the electron being scattered off in some direction  $\theta$  and a scattering energy of  $E'$ .

The two most important Kinematic Variables are:

- $Q^2 = 4EE' \sin^2(\frac{\theta}{2})$
- $W = \sqrt{M^2 + 2M(E' - E) - Q^2}$

These two variables listed above are the formulas that are used to find our Lorentz Invariant missing mass and 4-momentum per event. In order to find the Asymmetry values, we use the following equations:

- $A_{raw} = \frac{n^+ - n^-}{n^+ + n^-}$
- $A_{||} = \frac{1}{f} \cdot \frac{A_{raw}}{P_b P_t}$
- $\Delta A = \frac{1}{\sqrt{n^+ + n^-}}$

Where,  $n^+$  is the number of electrons polarized in the same direction of target, and  $n^-$  is the number of electrons polarized in the opposite direction of the target, and  $\Delta A$  is the calculated error of the Asymmetry. The experiment used five different electron beam energies, the maximum being 3.0GeV. I used only one of those five beam energies (3.0GeV). Additionally, all of the values for  $\theta$  will be drawn from the tree as the code runs (see Methods).

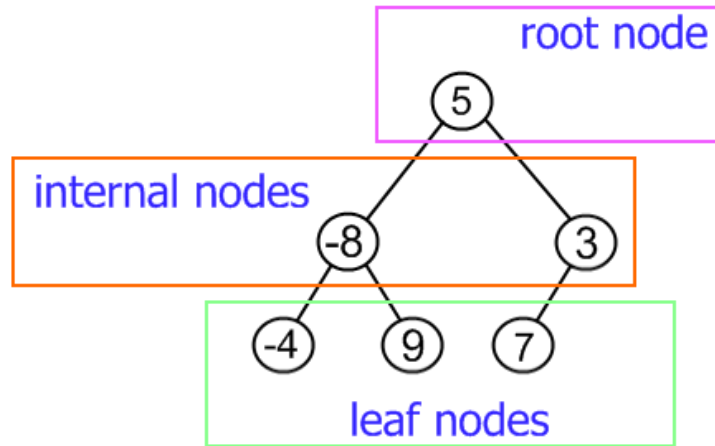
# **Methods:**

## **(2.1)**

### **Software Requirements**

Before the creation of the program, there are a few requirements that must be met. The first requirement is that the TTree must be downloaded and saved in Ubuntu, which was sent to me by my adviser, where, as stated from above, a TTree is a data structure that simulates a ordered tree structure, where an ordered structure of nodes is created, where a node can either be a “root” node, which is a node on the top of the hierarchy, a “branch” node (internal node), which is a node that is beneath the root node, but still has “child” nodes, which are known as the leave nodes (See Figure 2.10 for a visual representation). Once the TTree was acquired, I then had to open up the TTree and I extract the “Analyze” files from the H10 branch (NOTE: both of the Analyze files were contained in the tree, and are used as a foundation for the measurement). Once saved, I created a copy of them and named it “Vegeta”. The originals serve no purpose except as a cross check for the new code, if something is wrong with the foundation of the program, I can compare my code to the “Analyze” code.





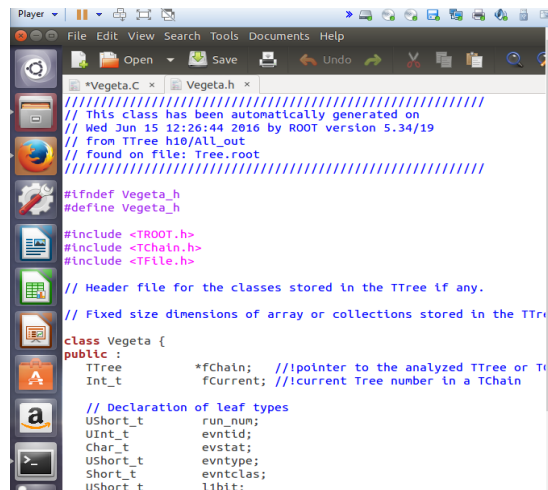
**Figure 2.10:** Here we see a visual representation of what a Tree looks like, where the Tree itself is the root node, the internal nodes are our branches (H10 is located here) and our leaf nodes contain our valuable data (such as momentum, scattering angle, etc.).

## (2.2)

## Collaboration

Before running the code, I first had to make sure that the Vegeta.C includes the Vegeta.h file, where the .C file contains all the code that acts upon the tree, and the .h file is a file that declares the leaf and branch types. Vegeta.h requires very few edits, where the edits that are made are to tell the file to allow Vegeta.C to include it. The reason why this needs to happen is because that Vegeta.h is the file that declares all of the leaf and branch types, naming the variables accordingly, and therefore, requires no edits on my part.

In order to get these files to be included with each other, they must first be saved in the same folder. Following this, I then renamed them so that they have the same name (Vegeta) and then edit through the analyze.h and change all instances of “analyze” to “Vegeta” (as shown below in Figure 2.21). Vegeta.C will not be able to include Vegeta.h properly if the class names, definitions, and constructors aren’t called Vegeta as well. Other than the name changes to Vegeta.h, there were no other alterations to that file.



```
Player
File Edit View Search Tools Documents Help
*Vegeta.C x Vegeta.h x
// This class has been automatically generated on
// Wed Jun 15 12:26:44 2016 by ROOT version 5.34/19
// from TTree h10/All_out
// found on file: Tree.root
// =====
#ifndef Vegeta_h
#define Vegeta_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
class Vegeta {
public:
    TTree      *fChain;    ///!pointer to the analyzed TTree or TChain
    Int_t      fCurrent;   ///!current Tree number in a TChain

    // Declaration of Leaf types
    UShort_t   run_num;
    UInt_t     evntId;
    Char_t     evstat;
    UShort_t   evntype;
    Short_t    evntclas;
    UShort_t   lbit;
};
```

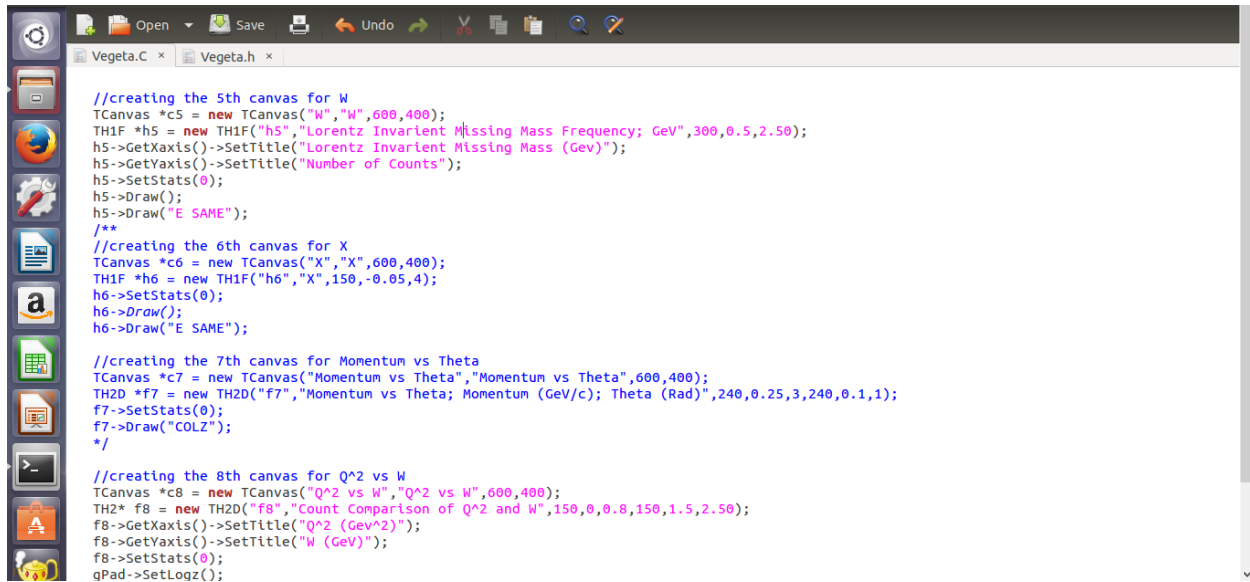
**Figure 2.21:** Here we define our class name and definition. These, along with a few other statements, are all required to be named “Vegeta” such that it can be included when Vegeta.C runs.

## (2.3)

### Processes

Before the script runs anything, there are several values that are required to be initialized. Before the arrays, we first start with the initialization of the doubles E, M, "NumeratorX", "NX", bin, and W, where E is the incident energy, M is the mass of the proton, NumeratorX is a placeholder for the numerator for finding the asymmetry, NX is the placeholder for the count for finding the asymmetry, bin is used to find which bin gets a count added to it (is always the integer value of  $100 \cdot W$ ), and W is set at 0 (W will be changed later). Following those constants, I then initialize the integer arrays npl, nmi, and the double array Asymmetry, where the npl and nmi arrays have 300 "bins" and are used for the purpose of keeping track of the number of counts in each bin (npl is for the n+ bins, and nmi is for the n- mins), and where the Asymmetry array has 30 "bins" and is used to hold onto the calculated asymmetry value. The npl, nmi, and Asymmetry arrays are all initialized to hold 0's in each of their bins, because (for reasons unknown) not doing this can cause weird things to occur during calculations, such as integers arrays spewing out doubles for no reason.

Following the variable initiation, the next section is devoted to the construction of the canvases and the histograms. Each individual canvas has its own constructor, with the title of the canvas, the x-axis size and the y-axis size fitted accordingly (see Figure 2.31 for details).



**Figure 2.31:** Here we see the basic constructors for the 1-Dimensional Histogram “W”, and the Two Dimensional Histogram  $Q^2$  vs W. In the first line of the constructor (Following TCanvas), the first two values in the parenthesis are the titles of the histograms and window respectively. The following 2 values show the size of the window which holds the histogram.

Once the canvas has been initialized, I then created the histogram using a method called “THXF”. This creates the histogram itself (the canvas creates the background and size of the histogram, while “THXF” creates the histogram itself with “X” dimensions. Within the THXF constructor, the names of the axes, number of bins, and boundaries are open to input from me (300 bins for 1-Dimensional Histograms, 150x150 bins for 2 Dimensional Histograms, and x-axis ranges from 0.5 to 2.5 for W, and 0 to 0.8 for  $Q^2$  vs W). After that, I then select the histogram and remove the legend from it, in order to remove some clutter. After that, I then “Draw” the histogram (NOTE: The histogram can still be altered after it has been drawn, and is best drawn this early so that it is easier to alter when data comes in). Finally, errors will be accounted for in this section, enabling the creation of error bars based on the data taken in. Once the histograms for W and  $Q^2$  were completed, I then created the 2-Dimensional histogram comparing the two values, following syntax similar to the creation of the 1 Dimensional Histograms.

After the creation of the  $Q^2$  vs W histogram, I then got to work on creating the graph that compares the calculated asymmetry in a given binning scheme to the Lorentz Invariant missing mass. Instead of using a histogram constructor, I instead used a “TGraphError” constructor, due to the fact that this is not a histogram, but is instead a graphed out comparison of our Asymmetry and W.

After the construction of the canvases and the graphs, comes the “for” loop. It is within this loop where the filling of the histograms are done, as well as keeping track of the total counts in npl and nmi. Before the filling of histogram, we must first determine if the particle that we are working with is an electron or not (particle ID 11). Any information about a particle that is not an electron does not contribute to the filling of the histograms. For most of the histograms, we only have to worry about dealing with electrons, but for the Asymmetry comparison and the n+ and n- histograms, we have to take into account the Helicity of the electron. In order to do this, we must go through two big filters, the first barrier determining if we have a scalar value of the helicity (Hel\_Flag), and the second barrier determining the spin of the electron (Hel\_ofl). Depending on if the electron was found to be polarized with or against the target, we fill either the n+ or n- histogram and add a count into npl or nmi (based on our helicity) for the array value of the calculated integer value of W (referred to our bin), and this value is calculated by determining our integer value of  $W*100$ . When we find the bin, we then add a count into npl (if polarized with) or nmi (if polarized against) into the “bin” that was determined by the integer value of  $W*100$ . Figure 4.44 shows what this particular process looks like.

```

*Vegeta.C (~/.EG4) - gedit
//
//if(id[0] == 11){
//h4->Fill((4*M*p[0]*sin(acos(cz[0])/2))*(sin(acos(cz[0])/2)));
//}

if(id[0] == 11){
h5->Fill(sqrt(E+E+2*M*p[0]-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2))););
if(hel_flag[0] != 0){
//plots "W" on n+ histogram
if(hel_ofl == 1){
h12->Fill(sqrt(E+E+2*M*p[0]-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2))););
W=sqrt(E+E+2*M*p[0]-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2))););
bin = int(100*W);
npl[bin] = npl[bin] + 1;
//cout << npl[bin] << " n+ bin " << bin << " value" << endl;
//cout << bin << " bin value" << W << " W Value" << endl;
}
//plots "W" on n- histogram
if(hel_ofl == 0){
h11->Fill(sqrt(E+E+2*M*p[0]-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2))););
W = sqrt(E+E+2*M*p[0]-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2))););
bin = int(100*W);
nmi[bin] = nmi[bin] + 1;
//cout << nmi[bin] << " n- bin " << bin << " value" << endl;
//cout << bin << " bin value" << W << " W Value" << endl;
}
}
}

```

**Figure 2.32:** *It is worth noting that the first line following the helicity offline checks are the fill commands for n+ and n-. Whenever an event is found to be an electron, and is determined to have a helicity of 0 or 1, a value is added into the n+ or n- histogram, that value being based on the Lorentz Invariant missing mass value associated with it.*

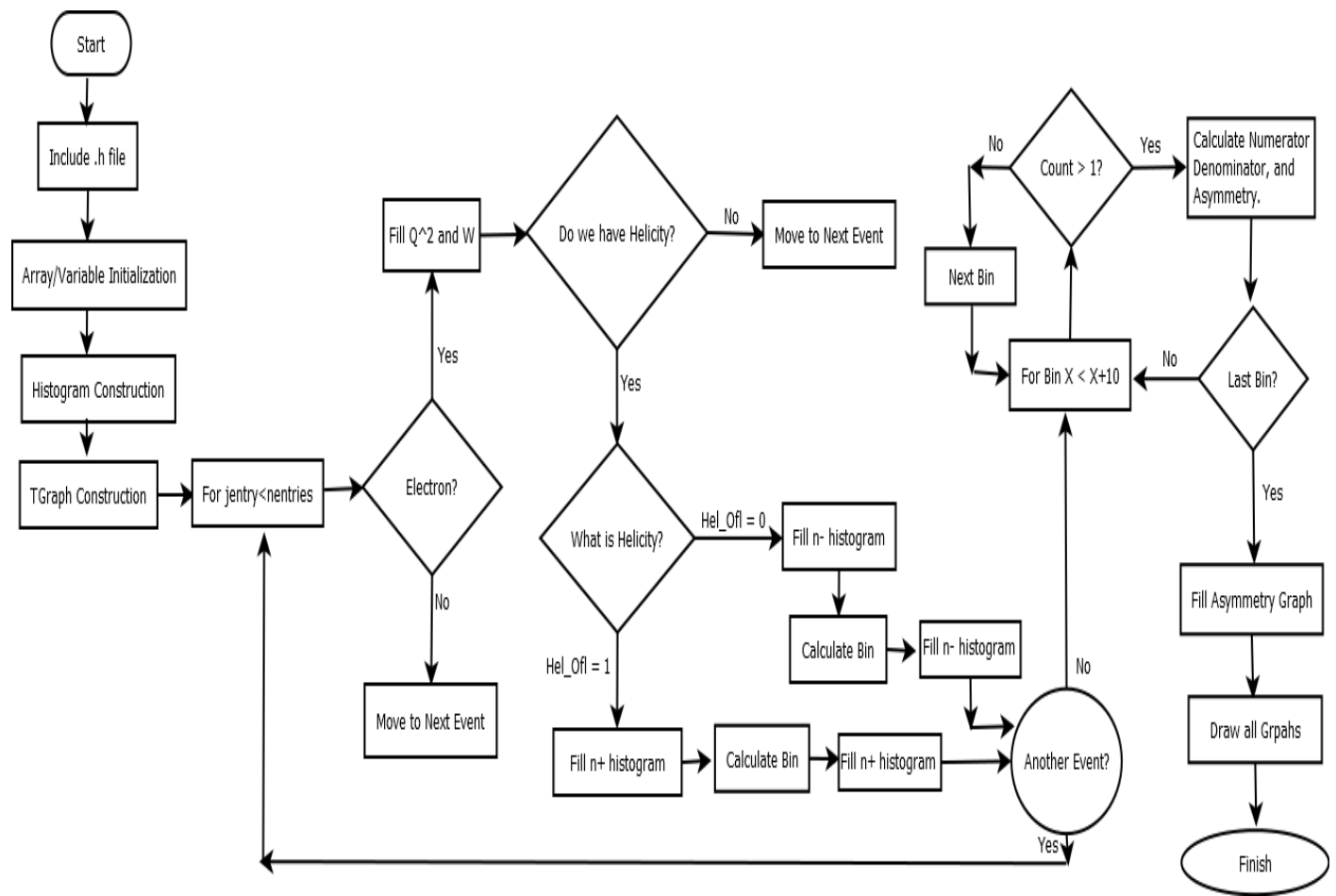
Once the main for loop has finished going through all the events within the first branch, we then move onto the next section of the script, which governs the creation of the binning scheme. In this section, the script loops over 10 times, starting at the number 50 (bin 50) and ending at the 59<sup>th</sup> bin. During this section, the NumeratorX values are found for each bin by taking the  $npl[k] - nmi[k] + \text{NumeratorX}$ , where “k” is the value of the current bin we are getting our counts from and NumeratorX is the current value of our numerator (summed up from the first to the last bin). Finding our numerator can also be achieved by summing all our npl and nmi values and then solving in the end, but this is a method that I felt was less suitable for how I structured my code.

This process is identical for finding the denominator, which is done by calculating  $npl[k] + nmi[k] + N2$  (summed up the same way as the Numerator). Once the numerator and the denominator have been found, we then calculate the asymmetry. This process repeats itself many more times, ending at bin 249 (this is due to the fact that there are very few W counts before 50 or after 250).

Following this section, comes the second half of the constructor for the W vs Asymmetry graph. Using the Root TGraph constructor requires the values to be hard coded into each of the initialization sections, these initialization sections being the x, y, ex, and ey values, where the y values are filled with our saved asymmetries from the binning system, the x values start at 0.55 and increase by 0.1 as we move within the data, ex is set to 0, due to there being no error in the x direction, and ey is the asymmetry error, which is found by  $\frac{1}{\sqrt{n^+ + n^-}}$ . Once these values have been placed into the graph, the script then goes through some other basic initialization commands, such as the axis lengths, titles, filling styles, etc. After this section, each of the other canvases for the histograms are updated as well, so that they may display the information necessary.

## (2.4)

## Flow Chart





## (2.5)

## Code Appendix

```
#define Vegeta_cxx
#include "Vegeta.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <stdlib.h>
void Vegeta::Loop()
{
// In a ROOT session, you can do:
//   Root > .L Vegeta.C
//   Root > Vegeta t
//   Root > t.GetEntry(12); // Fill t data members with entry number 12
//   Root > t.Show();      // Show values of entry 12
//   Root > t.Show(16);    // Read and show values of entry 16
//   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

/** To do List:
-Factor in the Multiple Scattering Cuts
-Momentum Correction
-EC (Electro-magnetic calridimers cuts)
-Ospenko Cuts (track and timing for cherkov counter)
-Vz Cuts (cut on vertex positions)
-PMIn Cuts (cut out low momentum particles)
-Sector Cuts
-nphe cuts (# of photo-electrons, cut out weak ones)
-Fiducial cuts (remove inefficient data)
-Elastic Peak should be positive (Look at the directory for more details of the cuts)
-Plot n+ and n- on the same Canvas/Plot (Check any differences).
-create n+ and n- vs W
-make the code more efficient (CRITICAL)
*/

//Initializing Most of the Constants
double E = 0.936;
double M = 3.1;
double W=0;
int npl[300];
int nmi[300];
double Count[300];
double Asymmetry[30];
for(int i =0; i<300;i++){
    npl[i] = 0;
    nmi[i] = 0;
}
double Q = 0;
double Numerator[30];
double N[30];
for(int z=0;z<30;z++){
    Asymmetry[z] = 0;
```

```

    Numerator[z] = 0;
    N[z] = 0;
}
double hol[300];
int j=0;
int bin=0;
float bins[300];

//creating the first canvas for momentum
//TCanvas *c1 = new TCanvas("Momentum","Momentum",600,400);
//TH1F *h1 = new TH1F("h1","Momentum; GeV/C",150,0,4);
//h1->SetStats(0);
//h1->Draw();
//h1->Draw("E SAME");

//creating the second canvas for Theta
TCanvas *c2 = new TCanvas("Theta","Theta",600,400);
TH1F *h2 = new TH1F("h2","Theta; Radians",150,0.7,1);
h2->SetStats(0);
h2->Draw();
h2->Draw("E SAME");

//creating the third canvas for Phi
//TCanvas *c3 = new TCanvas("Phi","Phi",600,400);
//TH1F *h3 = new TH1F("h3","Phi; Radians",150,-3,3);
//h3->SetStats(0);
//h3->Draw();
//h3->Draw("E SAME");

//Creating the 4th canvas for Q^2
TCanvas *c4 = new TCanvas("Q^2","Q^2",600,400);
TH1F *h4 = new TH1F("h4","4D Angular Momentum Squared Frequency; GeV^2",150,0,log(5));
gPad->SetLogx();
h4->GetXaxis()->SetTitle("4th-Dimensional Angular Momentum Vector Squared (GeV^2)");
h4->GetYaxis()->SetTitle("Number of Counts");
h4->SetStats(0);
h4->Draw();
h4->Draw("E SAME");

//creating the 5th canvas for W
TCanvas *c5 = new TCanvas("W","W",600,400);
TH1F *h5 = new TH1F("h5","Lorentz Invariant Missing Mass Frequency; GeV",300,0.5,2.50);
h5->GetXaxis()->SetTitle("Lorentz Invariant Missing Mass (GeV)");
h5->GetYaxis()->SetTitle("Number of Counts");
h5->SetStats(0);
h5->Draw();
h5->Draw("E SAME");
/**
//creating the 6th canvas for X
TCanvas *c6 = new TCanvas("X","X",600,400);
TH1F *h6 = new TH1F("h6","X",150,-0.05,4);
h6->SetStats(0);
h6->Draw();
h6->Draw("E SAME");

//creating the 7th canvas for Momentum vs Theta
TCanvas *c7 = new TCanvas("Momentum vs Theta","Momentum vs Theta",600,400);
TH2D *f7 = new TH2D("f7","Momentum vs Theta; Momentum (GeV/c); Theta (Rad)",240,0.25,3,240,0.1,1);
f7->SetStats(0);
f7->Draw("COLZ");
*/

//creating the 8th canvas for Q^2 vs W
TCanvas *c8 = new TCanvas("Q^2 vs W","Q^2 vs W",600,400);
TH2* f8 = new TH2D("f8","Count Comparison of Q^2 and W",150,0,0.8,150,1.5,2.50);
f8->GetXaxis()->SetTitle("Q^2 (GeV^2)");
f8->GetYaxis()->SetTitle("W (GeV)");
f8->SetStats(0);
gPad->SetLogz();
f8->Draw("COLZ");

```

```

/** creating the 9th canvas for Q^2 vs X
TCanvas *c9 = new TCanvas("(Q^2) vs X", "Q^2 vs X", 600, 400);
TH2D *f9 = new TH2D("f9", "Q^2 vs X; GeV^2", 200, 0, 1, 200, 0, 1);
f9->SetStats(0);
f9->Draw("COLZ");
*/

//creating the 11th canvas for n- (uses other histograms for data)
TCanvas *c11 = new TCanvas("n-", "n- count", 600, 400);
TH1D *h11 = new TH1D("h11", "n- count", 300, 0, 2.60);
h11->GetXaxis()->SetTitle("Lorentz Invariant Missing Mass");
h11->GetYaxis()->SetTitle("Number of Electrons polarized against the Target");
h11->Draw();
h11->Draw("E SAME");

//creating the 12th canvas for n+, this histogram will not be drawn
TCanvas *c12 = new TCanvas("n+", "n+ count", 600, 400);
TH1D *h12 = new TH1D("h12", "n+ count", 300, 0, 2.60);
h12->GetXaxis()->SetTitle("Lorentz Invariant Missing Mass");
h12->GetYaxis()->SetTitle("Number of Electrons Polarized with the Target");
h12->Draw();
h12->Draw("E SAME");

c15 = new TCanvas("c15", "Calculated Asymmetry per binned W count", 200, 300, 700, 500);
c15->SetFillColor(42);
c15->GetFrame()->SetFillColor(21);
c15->GetFrame()->SetBorderSize(12);

//ofstream Wfile;
//Wfile.open ("Lorentz.txt");
//runs through the code and adds the values into my histogram, as well as doing the histogram addition
if (fChain == 0) return;
Long64_t nentries = fChain->GetEntriesFast();

Long64_t nbytes = 0, nb = 0;
for (Long64_t jentry=0; jentry<nentries;jentry++) {
    //if(id[0] == 11){
        //h1->Fill(p[0]);
    //}
    if(id[0] == 11){
        h2->Fill(cz[0]);
    }

    //if(id[0] == 11){
        //if(cx[0] < 0 && cy[0] < 0) {
            //h3->Fill(atan(cy[0]/cx[0]) - 3.14);
        //}
        //if(cx[0] < 0 && cy[0] > 0) {
            //h3->Fill(atan(cy[0]/cx[0]) + 3.14);
        //}
        //if(cx[0] > 0) {
            //h3->Fill(atan(cy[0]/cx[0]));
        //}
    //}

    if(id[0] == 11){
        h4->Fill((4*M*p[0]*sin(acos(cz[0])/2))*sin(acos(cz[0])/2));
        Q=(4*M*p[0]*sin(acos(cz[0])/2))*sin(acos(cz[0])/2);
    }

    if(id[0] == 11){
        h5->Fill(sqrt(E*E+2*E*(M-p[0])-4*M*p[0]*sin(acos(cz[0])/2))*sin(acos(cz[0])/2));
        if(hel_flag[0] != 0){
            //plots "W" on n+ histogram
            if(hel_ofl == 1){
                h12->Fill(sqrt(E*E+2*E*(M-p[0])-4*M*p[0]*sin(acos(cz[0])/2))*sin(acos(cz[0])/2));
                W=sqrt(E*E+2*E*(M-p[0])-4*M*p[0]*sin(acos(cz[0])/2))*sin(acos(cz[0])/2));
                //Wfile << W << endl;
                bin = int(100*W);
                npl[bin] = npl[bin] + 1;
                f8->Fill(Q,W);
                //cout << npl[bin] << " n+ bin " << bin << " value" << endl;
                //cout << bin << " bin value" << W << " W Value" << endl;
            }
        }
    }
}

```

```

//plots "W" on n- histogram
if(hel_ofl == 0){
    h11->Fill(sqrt(E*E+2*E*(M-p[0])-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2))));
    W = sqrt(E*E+2*E*(M-p[0])-4*M*p[0]*sin(acos(cz[0])/2)*(sin(acos(cz[0])/2)));
    //Wfile << W << endl;
    bin = int(100*W);
    nmi[bin] = nmi[bin] + 1;

    //cout << nmi[bin] << " n- bin " << bin << " value" << endl;
    //cout << bin << " bin value " << W << " W Value" << endl;
}
}
}
/**
if(id[0] == 11){
h6->Fill(4*M*p[0]*sin(acos(cz[0])/2)/(2*E*(M-p[0])));
}

if(id[0] == 11){
f7->Fill(p[0],acos(cz[0]));
}

*/
// if(id[0] == 11){
//f8->Fill(Q,W);
//}

/** if(id[0] == 11){
    f9->Fill((4*M*p[0]*sin(acos(cz[0])/2)*sin(acos(cz[0])),4*M*p[0]*sin(acos(cz[0])/2)/(2*E*(M-p[0])));
}
*/
//determines if an electron is considered n+ or n-, and then plots W
Long64_t ientry = LoadTree(jentry);

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

//Asymmetry vs W fill section
for(int k=50;k<60;k++){
    Numerator[0] = npl[k] - nmi[k] + Numerator[0];
    N[0] = npl[k] + nmi[k] + N[0];
    Asymmetry[0] = (Numerator[0]/N[0]);
}
for(k=60;k<70;k++){
    Numerator[1] = npl[k] - nmi[k] + Numerator[1];
    N[1] = npl[k] + nmi[k] + N[1];
    Asymmetry[1] = (Numerator[1]/N[1]);
}
for(k=70;k<80;k++){
    Numerator[2] = npl[k] - nmi[k] + Numerator[2];
    N[2] = npl[k] + nmi[k] + N[2];
    Asymmetry[2] = (Numerator[2]/N[2]);
}
for(k=80;k<90;k++){
    Numerator[3] = npl[k] - nmi[k] + Numerator[3];
    N[3] = npl[k] + nmi[k] + N[3];
    Asymmetry[3] = (Numerator[3]/N[3]);
}
for(k=90;k<100;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[4] = npl[k] - nmi[k] + Numerator[4];
        N[4] = npl[k] + nmi[k] + N[4];
        Asymmetry[4] = (Numerator[4]/N[4]);
    }
}
for(k=100;k<110;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[5] = npl[k] - nmi[k] + Numerator[5];
        N[5] = npl[k] + nmi[k] + N[5];
        Asymmetry[5] = (Numerator[5]/N[5]);
    }
}
for(k=110;k<120;k++){
    if((npl[k]>0) || (nmi[k]>0)){

```

```

        Numerator[6] = npl[k] - nmi[k] + Numerator[6];
        N[6] = npl[k] + nmi[k] + N[6];
        Asymmetry[6] = (Numerator[6]/N[6]);
    }
}
for(k=120;k<130;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[7] = npl[k] - nmi[k] + Numerator[7];
        N[7] = npl[k] + nmi[k] + N[7];
        Asymmetry[7] = (Numerator[7]/N[7]);
    }
}
for(k=130;k<140;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[8] = npl[k] - nmi[k] + Numerator[8];
        N[8] = npl[k] + nmi[k] + N[8];
        Asymmetry[8] = (Numerator[8]/N[8]);
    }
}
for(k=140;k<150;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[9] = npl[k] - nmi[k] + Numerator[9];
        N[9] = npl[k] + nmi[k] + N[9];
        Asymmetry[9] = (Numerator[9]/N[9]);
    }
}
for(k=150;k<160;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[10] = npl[k] - nmi[k] + Numerator[10];
        N[10] = npl[k] + nmi[k] + N[10];
        Asymmetry[10] = (Numerator[10]/N[10]);
    }
}
for(k=160;k<170;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[11] = npl[k] - nmi[k] + Numerator[11];
        N[11] = npl[k] + nmi[k] + N[11];
        Asymmetry[11] = (Numerator[11]/N[11]);
    }
}
for(k=170;k<180;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[12] = npl[k] - nmi[k] + Numerator[12];
        N[12] = npl[k] + nmi[k] + N[12];
        Asymmetry[12] = (Numerator[12]/N[12]);
    }
}
for(k=180;k<190;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[13] = npl[k] - nmi[k] + Numerator[13];
        N[13] = npl[k] + nmi[k] + N[13];
        Asymmetry[13] = (Numerator[13]/N[13]);
    }
}
for(k=190;k<200;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[14] = npl[k] - nmi[k] + Numerator[14];
        N[14] = npl[k] + nmi[k] + N[14];
        Asymmetry[14] = (Numerator[14]/N[14]);
    }
}
for(k=200;k<210;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[15] = npl[k] - nmi[k] + Numerator[15];
        N[15] = npl[k] + nmi[k] + N[15];
        Asymmetry[15] = (Numerator[15]/N[15]);
    }
}
for(k=210;k<220;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[16] = npl[k] - nmi[k] + Numerator[16];
        N[16] = npl[k] + nmi[k] + N[16];
        Asymmetry[16] = (Numerator[16]/N[16]);
    }
}
}

```

```

for(k=230;k<240;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[17] = npl[k] - nmi[k] + Numerator[17];
        N[17] = npl[k] + nmi[k] + N[17];
        Asymmetry[17] = (Numerator[17]/N[17]);
    }
}
for(k=240;k<250;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator[18] = npl[k] - nmi[k] + Numerator[18];
        N[18] = npl[k] + nmi[k] + N[18];
        Asymmetry[18] = (Numerator[18]/N[18]);
    }
}
/**for(k=250;k<260;k++){
    if((npl[k]>0) || (nmi[k]>0)){
        Numerator20 = npl[k] - nmi[k] + Numerator20;
        N20 = npl[k] + nmi[k] + N20;
        cout << N20 << endl;
        Asymmetry[19] = (Numerator20/N20);
    }
}
ofstream nmfile;
nmfile.open ("n-counts.txt");
for(int z = 50; z < 250 ; z++){
    nmfile << nmi[z] << endl;
    //myfile << "Asymmetry" << " " << Asymmetry[z] << endl;
}
ifstream infile("n-counts.txt");
string qWord;
double value = 0;
double sum = 0;

ofstream npfile;
npfile.open ("n+counts.txt");
for(z = 50; z<250 ; z++){
    npfile << npl[z] << endl;
}
ifstream infile3("n+counts.txt");
while(!infile3.eof()){
    infile3 >> qWord;
    cout << qWord << endl;
}
while (!infile.eof()){
    infile >> qWord;
    cout << qWord << endl;
}
ifstream infile2("Lorentz.txt");
while (!infile2.eof()){
    infile2 >> qWord;
    //cout << qWord << endl;
    value = atof(qWord.c_str());
    sum = sum + value;
    cout << sum << endl;
}
cout << sum << endl;
Wfile.close();
nmfile.close();
npfile.close(); */

Double_t y[30] =
{Asymmetry[0],Asymmetry[1],Asymmetry[2],Asymmetry[3],Asymmetry[4],Asymmetry[5],Asymmetry[6],Asymmetry[7],Asymmetry[8],Asymmetr
y[9],Asymmetry[10],Asymmetry[11],Asymmetry[12],Asymmetry[13],Asymmetry[14],Asymmetry[15],Asymmetry[16],Asymmetry[17],Asymmetr
y[18]};
Double_t x[30]={0.65,0.75,0.85,0.95,1.05,1.15,1.25,1.35,1.45,1.55,1.65,1.75,1.85,1.95,2.05,2.15,2.25,2.35,2.45};
const Int_t n = 30;
Double_t ex[n] = {0};
Double_t ey[n] =
{(1/sqrt(N[0])),(1/sqrt(N[1])),(1/sqrt(N[2])),(1/sqrt(N[3])),(1/sqrt(N[4])),(1/sqrt(N[5])),(1/sqrt(N[6])),(1/sqrt(N[7])),(1/sqrt(N[8])),(1/sqr
t(N[9])),(1/sqrt(N[10])),(1/sqrt(N[11])),(1/sqrt(N[12])),(1/sqrt(N[13])),(1/sqrt(N[14])),(1/sqrt(N[15])),(1/sqrt(N[16])),(1/sqrt(N[17])),(1/sqrt(N[18]))};

gr = new TGraphErrors(n,x,y,ex,ey);
gr->GetXaxis()->SetLimits(0.5,2.6);
gr->GetYaxis()->SetLimits(0,0.040);
gr->SetTitle("Calculated Asymmetry per Bin;Lorentz Invariant Missing Mass (GeV); Asymmetry Value");

```

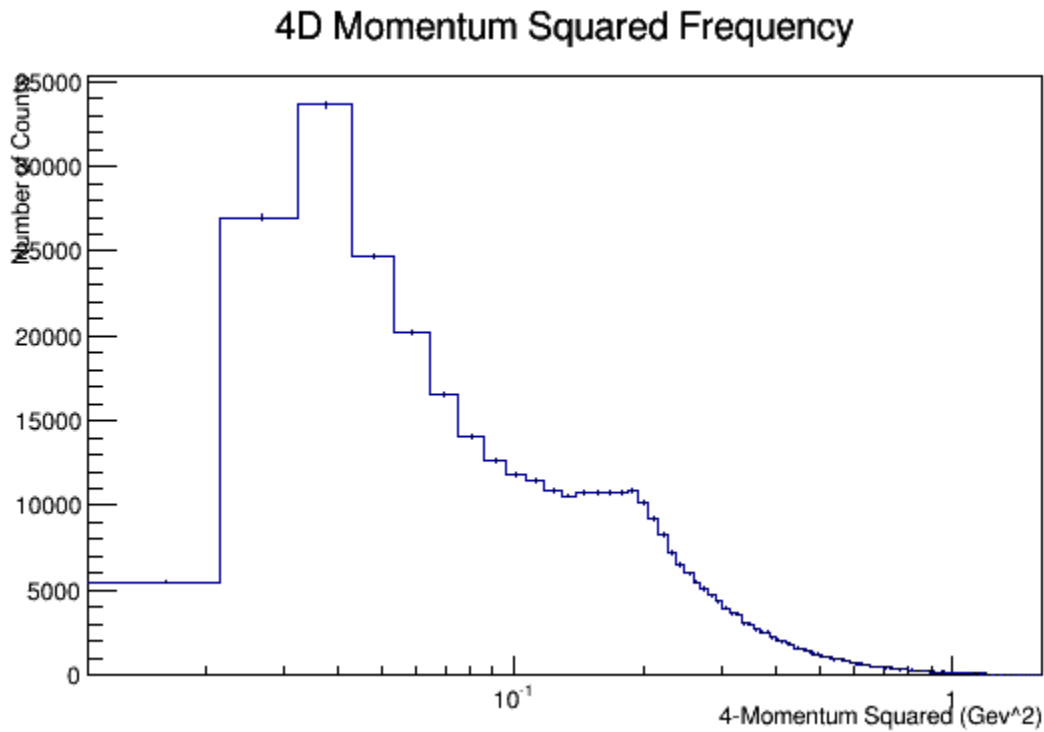
```
gr->SetMarkerColor(1);
gr->SetMarkerStyle(21);
gr->Draw("A*");
// "prints" out the histograms
// c1->Update();
c2->Update();
// c3->Update();
c4->Update();
c5->Update();
// c6->Update();
// f7->Update();
f8->Update();
// f9->Update();
c11->Update();
c12->Update();
}
```

# **Results and Conclusions:**

## **(3.1)**

### **Data Found**

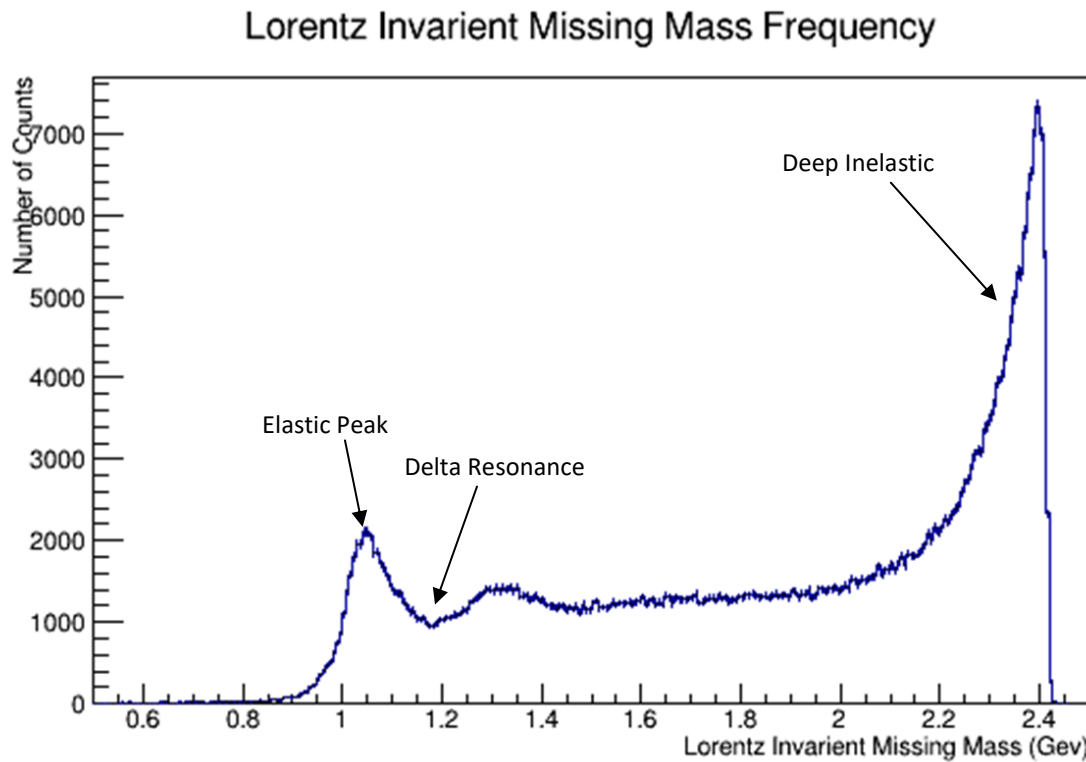
Following all of the methods previously stated, the results yielded by one run through the script has given me results that I would expect to see. Starting with the  $Q^2$  histogram, we can see a count comparison to the 4-Momentum of the virtual photon squared.



**Figure 3.11:** *Plotted out as stated in the “Kinematics” section, we can see that the most amount of counts is within the 0.05 to 0.06 GeV<sup>2</sup> range.*

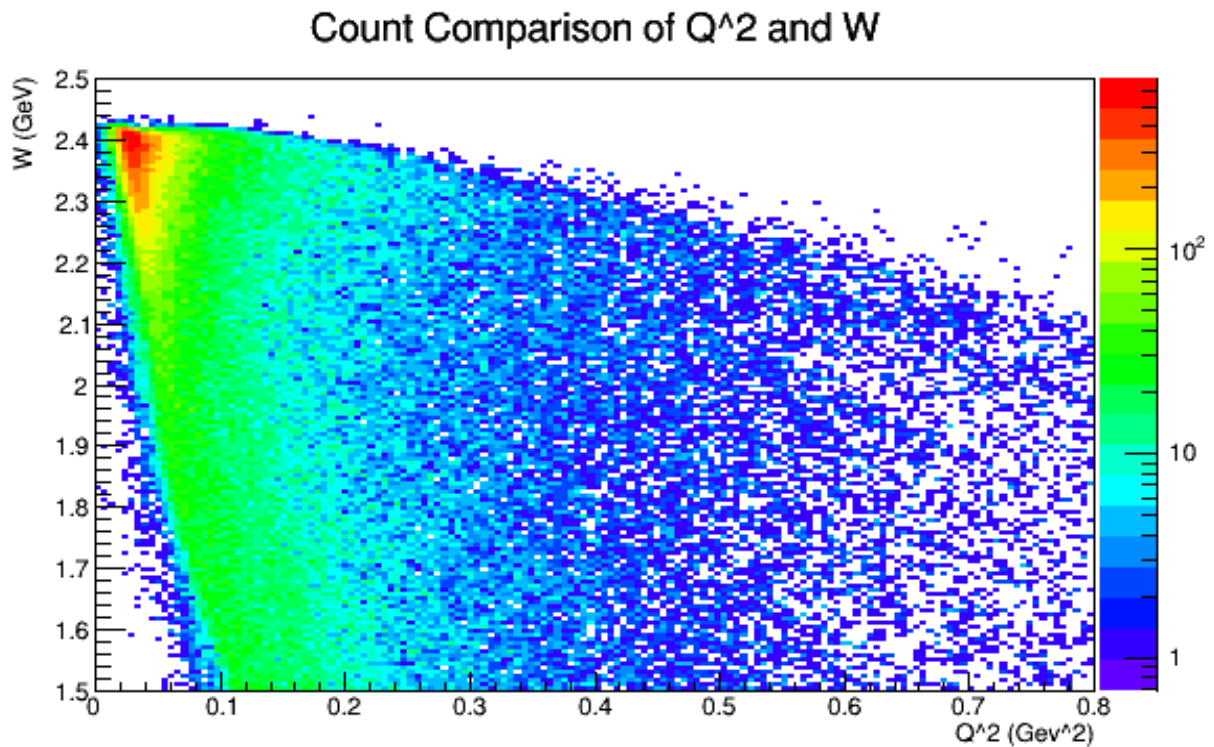


In addition to the 4-Momentum Vector, the Lorentz Invariant missing mass is also calculated by the method stated in the “Kinematics” section. In the histogram of the Lorentz Invariant Missing Mass, we have a few key sections that are worth noting. The first important section is known as the “elastic peak”, and it is at this point where collisions started to become inelastic, where extra particles were created during the collision. The second labeled area is known as the “Delta Resonance” and is where we see the creation of delta particles from the collision. The final section is dubbed “Deep Inelastic”, and it is within this section we a large amount of mass to go missing in all reference frames. Additionally, it is during this section we see resonances from other particles flatten out. Observe figure 3.12 for more details.



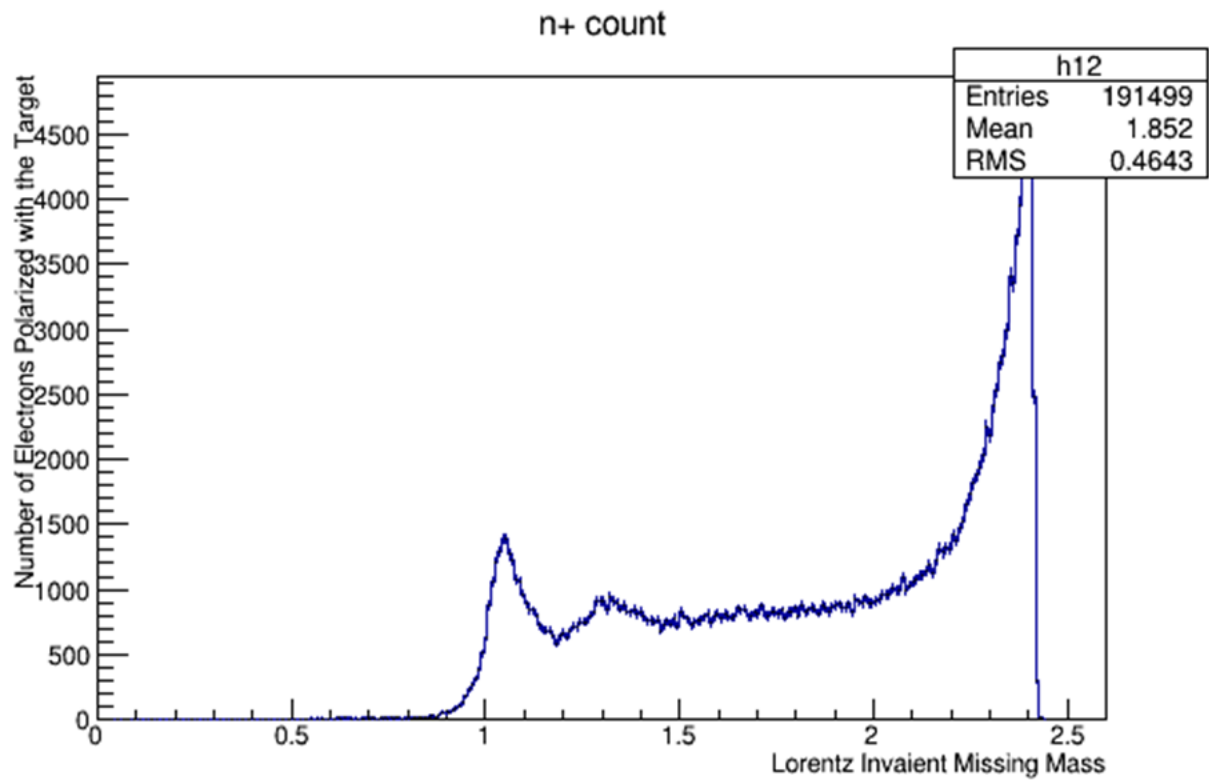
**Figure 3.12:** Due to the lack of detector cuts (such as the momentum corrections) our Elastic peak is located at 1.1 GeV. If the detector cuts were implemented, we should see the peak shift down to around 0.9 GeV, which uncoincidentally, is roughly the mass of a proton.

Following the creation of these two histograms, I then took a look at the 2-Dimensional histogram comparing the 4-Momentum and the Lorentz Invariant Missing Mass, giving us a count comparison between the two. As seen in Figure 3.13, we can see that we have a “hotspot” at 2.4 GeV (Lorentz Invariant Missing Mass) and 0.06 GeV<sup>2</sup> (for our 4-Momentum).

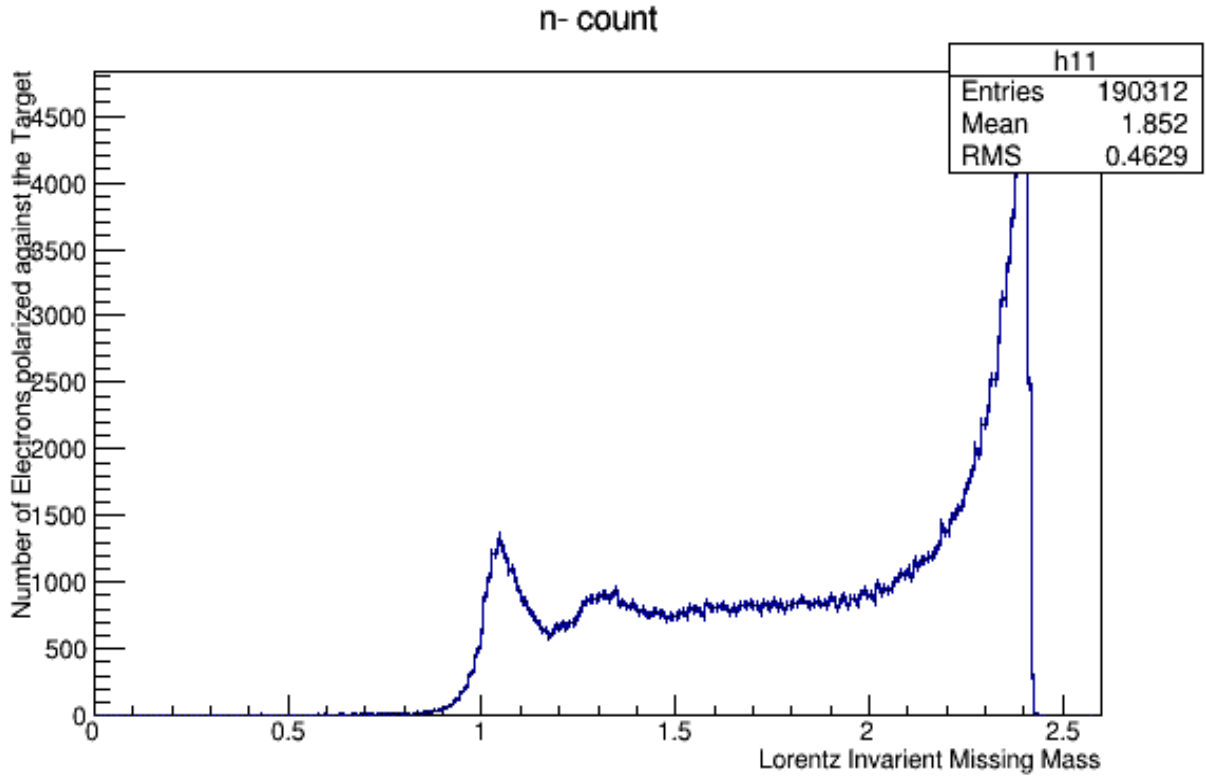


**Figure 3.13:** Taking a look at this comparison, we can see that we have a considerable amount of counts in the 2.4/0.06 GeV/GeV<sup>2</sup> area. This allows us to conclude that if the 4-Momentum of our Virtual photon squared is around 0.06, we are most likely to see a highly inelastic collision.

With the 1-dimensional histogram for the Lorentz Invariant Missing Mass created, it became much more feasible to create a histogram drawing out W, but with the check of the electron being polarized with or against the electron. This led to the creation of the n+ and n- histograms, which are shown as figures 3.14 and 3.15:



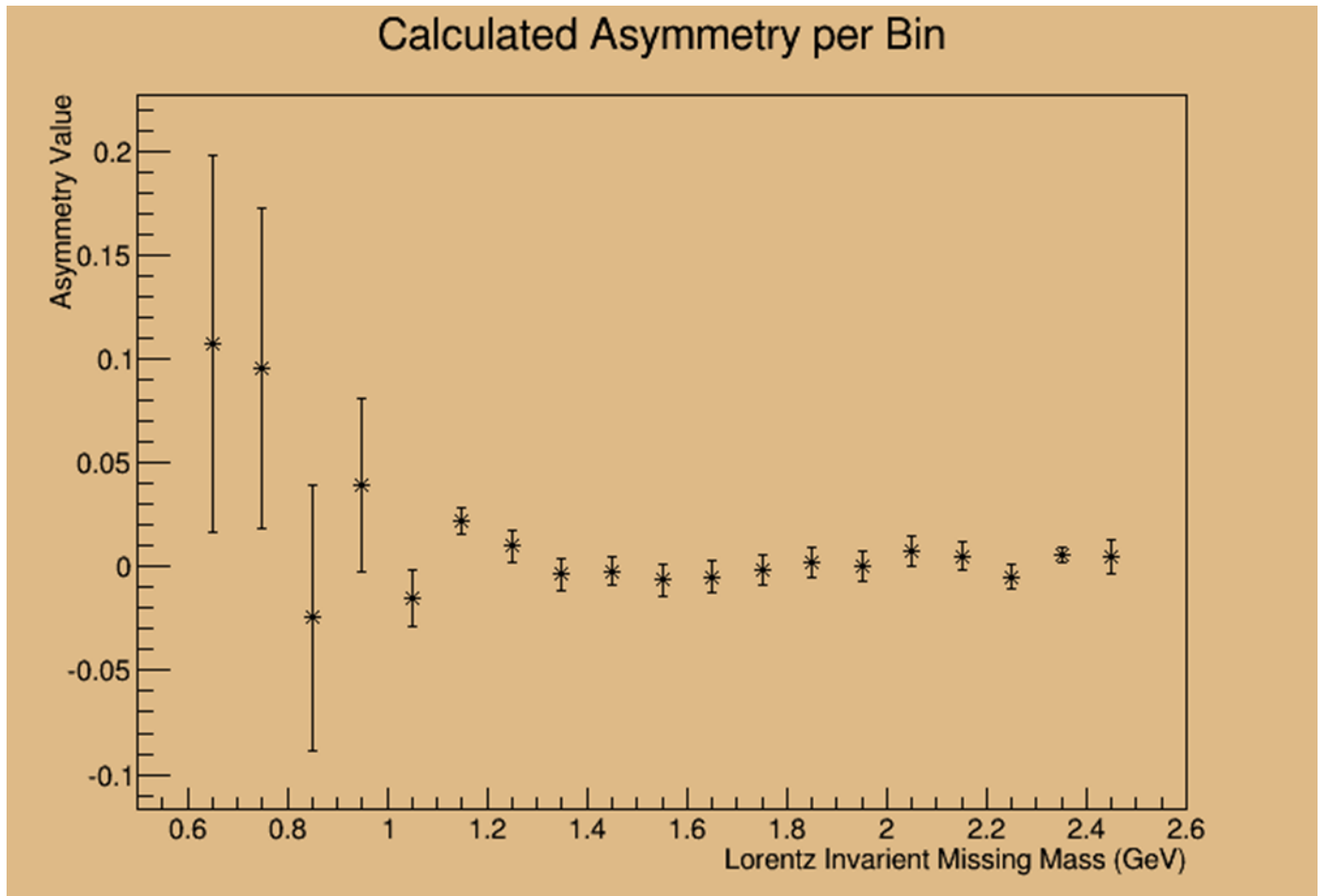
**Figure 3.14:** Similar to  $W$ , this histogram displays the number of counts for our values of  $W$ , but where only electrons polarized with the target are displayed.



**Figure 3.15:** *This histogram is nearly identical to Figure 3.14, but in this histogram, only electrons that are polarized against the target are counted. Take note of the small dip in counts at the elastic peak.*

Taking a look at both of the n+ and n- histograms, it is clearly noticeable that the histograms look almost exactly like the histogram for the Lorentz Invariant Missing Mass, which is a good sign, considering that both of these histograms are being filled in as a function of W, only with a check to determine the helicity. Additionally, we are also able to tell we are on the right track by being able to tell the minor differences between both histograms, being that they are slightly different from each other, and due to the fact that the n+ histogram contains more counts than the n- histogram. This is also a sign that we should be seeing mostly positive values for our asymmetry, and due to the number of counts being very similar, we should also see an asymmetry that is close to 0 in most places.

With the  $n^+$  and  $n^-$  histograms finished, it was then time to move onto the creation of the calculated Asymmetry comparison to W histogram. This histogram (as the title may suggest) compares the average asymmetry values of all of the bins that span over 0.1 GeV, giving a binning system. Figure 3.16 shows this comparison.



**Figure 3.16:** *Each bin spans 0.1 GeV*

For our Asymmetry vs W comparison, we can clearly see that (generally) the size of the Asymmetry decreases as the number of W counts increases. The reason why we see this decrease in Asymmetry size is due to the nearly equal distribution of electron polarization. At the same time however, we can also notice that the bins that received more counts, also contain much smaller error.

When observing the histogram for  $W$ , it is easy to notice that the amount of counts around 0.6 to 0.8 GeV are extremely low, and due to this, we end up getting an asymmetry that is larger than most (along with larger error bars). However, when taking a look at the higher values of the Lorentz Invariant Missing Mass (1.4-2.5 GeV) it is found that the asymmetry is a lot smaller (along with the error bars), showing that the amount of counts that have been received have been a lot higher, and due to the asymmetry being nearly 0, we can also quantify that the number of  $n^+$  and  $n^-$  counts are nearly the same, which is confirmed to be correct by our  $n^+$  and  $n^-$  histograms.

In addition to these results, there is one other value that gives us key information about the comparison, and that would be the bin located at around 1.10 GeV. As one may notice here, we have a positive asymmetry that is noticeably larger than the higher count asymmetries that follow. This value is our calculated asymmetry for our elastic peak, and due to this noticeable asymmetry as well as the smaller size of the error bars, we can conclude that we are much more likely to see an electron polarized with the target instead of against. Additionally, if we were to lose the ability to detect the polarization of the electron, we can still conclude that over several events, we can still expect the asymmetry at the elastic peak to still be positive.

Overall, by using only 1 run and no detector cuts, we can conclude that at a 4-Momentum of  $0.06 \text{ GeV}^2$ , we are more likely to see an inelastic collision than an elastic. Additionally, after observing our asymmetry and  $W$  comparison, we can conclude that most of the electrons were polarized with the target at the elastic peak, and due to this, we can expect this polarization for the elastic peak for any unobserved events, due to the low error bars and comparison.