FORM C
APPROVED FOR USE IN
PURDUE UNIVERSITY

Assignment #4.

1(a). The probability density for x is

$$\frac{1}{1}\frac{dF}{dx} = 1 + P_\tau(2x - 1)$$

which can be generated by inversion:

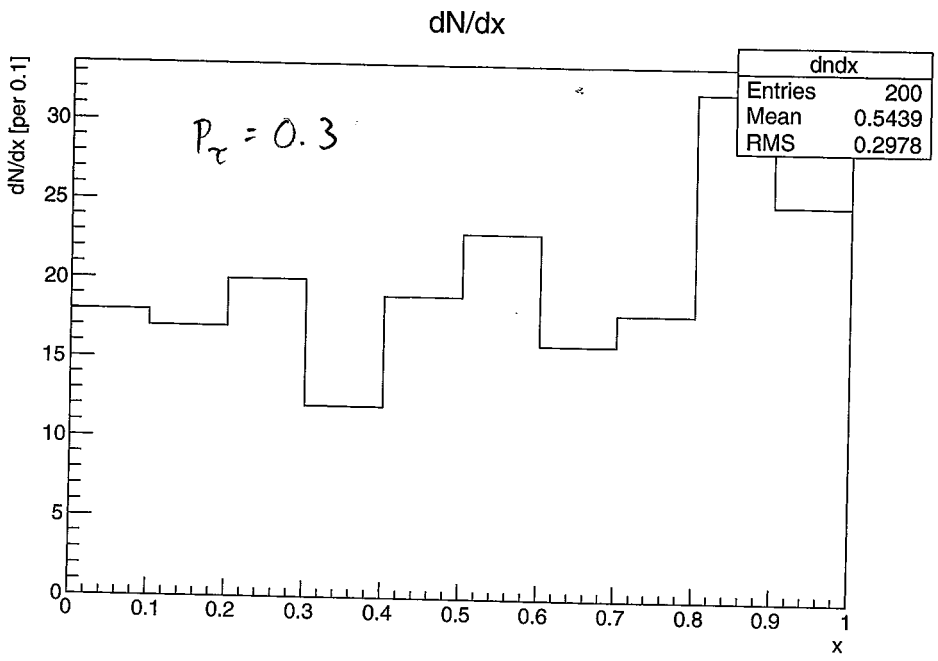
Let u be randomly distributed between 0 and 1. Then let

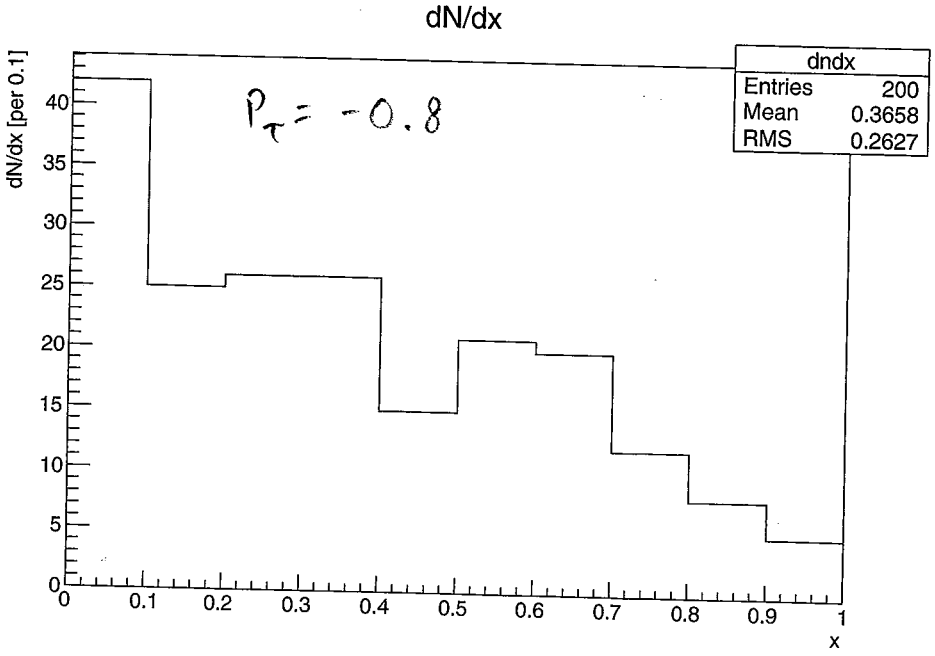
$$F(x) = \int_0^x \left(1 + P_\tau(2y - 1)\right) dy$$

$$= (1 - P_\tau)x + P_\tau x^2 = u.$$

So $$x = \frac{(P_\tau - 1) + \sqrt{(P_\tau - 1)^2 + 4P_\tau u}}{2P_\tau}$$

(See attached plots and source code.)

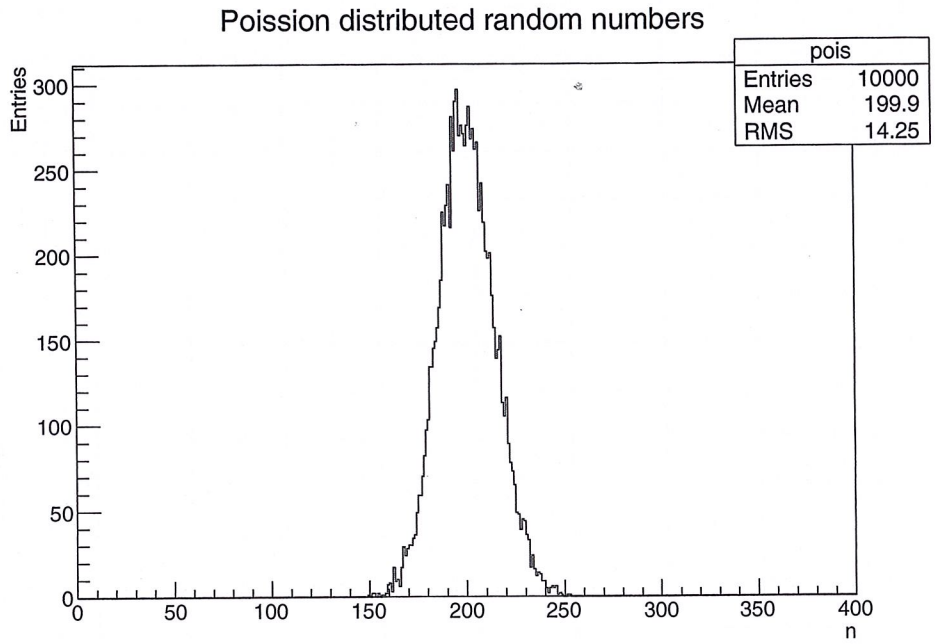

dN/dx
dndx
Entries 200
Mean 0.5439
RMS 0.2978
$P_\tau = 0.3$
dN/dx [per 0.1]
x


dN/dx
dndx
Entries 200
Mean 0.3658
RMS 0.2627
$P_\tau = -0.8$
dN/dx [per 0.1]
x

```
void Question1a(double q) {

  int nexp = 200;

  int nbin = 10;
  TH1F *h_x = new TH1F("dndx","dN/dx",nbin,0,1);
  h_x->SetMinimum(0);

  TRandom *rnd = new TRandom();

//
//  Generate template distributions
//
  for ( int ievt=0; ievt<nexp; ievt++ ) {
    double u = rnd->Uniform();
    double x = u;
    if ( q != 0 ) x = ((q-1)+sqrt((q-1)*(q-1)+4*q*u))/(2*q);
    h_x->Fill(x,1);
  }

  TCanvas *c1 = new TCanvas("c1",NULL,0,0,700,500);
  h_x->Draw();
  h_x->GetXaxis()->SetTitle("x");
  h_x->GetYaxis()->SetTitle("dN/dx [per 0.1]");
}
```

1(b)

```
void Question1b(double y) {

  TH1F *h_pois = new TH1F("pois","Poission distributed random numbers",400,0,400);
  TRandom *rnd = new TRandom();

  for ( int ievt=0; ievt<10000; ievt++ ) {
    int n = rnd->Poisson(y);
    h_pois->Fill(n,1);
  }
  TCanvas *c1 = new TCanvas("c1",NULL,0,0,700,500);
  h_pois->Draw();
  h_pois->GetXaxis()->SetTitle("n");
  h_pois->GetYaxis()->SetTitle("Entries");
}
```



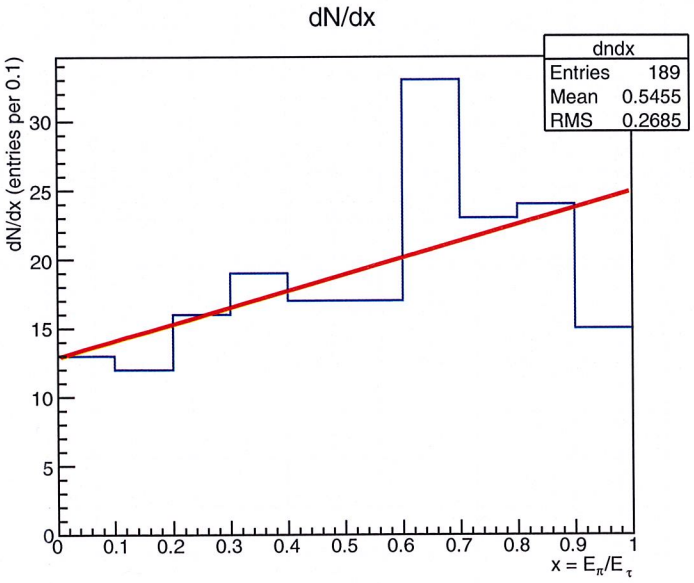
The rms is 14.25, to be compared with $\sqrt{200} = 14.14$ when $\mu = 200$ for large N.

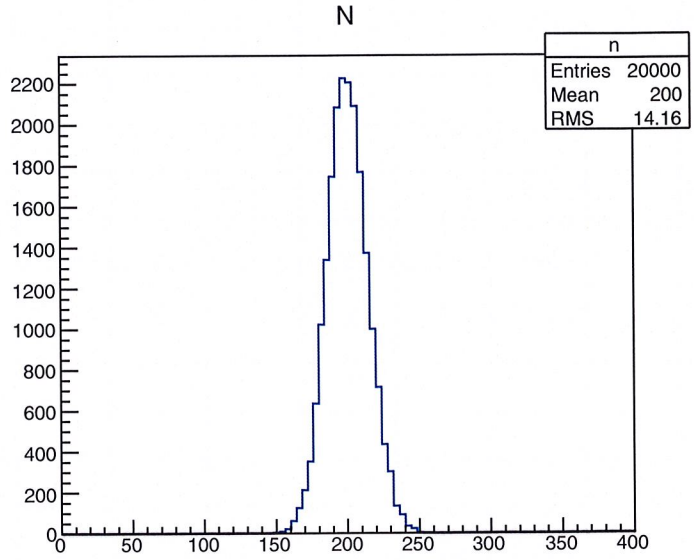
FORM C
APPROVED FOR USE IN
PURDUE UNIVERSITY

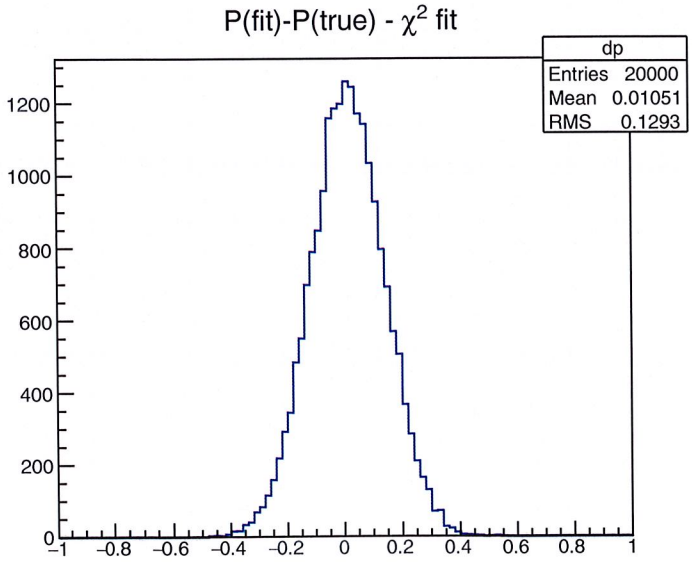
FORM C
APPROVED FOR USE IN
PURDUE UNIVERSITY

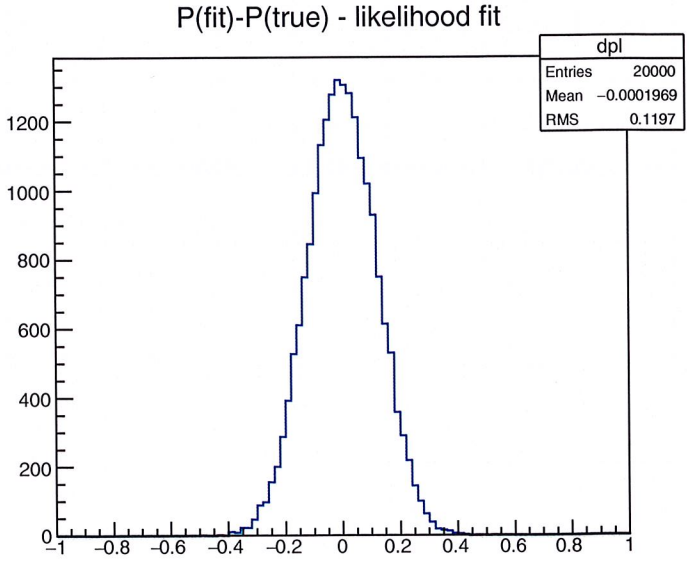
1(c) Using a χ^2 fit with 20,000 events, the mean of $(x - P_\pi)$ was 0.0105 ± 0.0010. Hence, the fitted parameter has a small bias.

(d) Using a likelihood fit, the mean is -0.00020 ± 0.0010 and there is no evidence of bias.









```
void TauPolarization(double p) {

  int nexp = 20000;
  double y = 200;
  TH1F *h_x = new TH1F("dndx","dN/dx",10,0,1);
  h_x->SetMinimum(0);
  TH1F *h_n = new TH1F("n","N",100,0,2*y);
  h_n->SetMinimum(0);

  TRandom *rnd = new TRandom();
  TCanvas *c1 = new TCanvas("c1",NULL,0,0,700,600);
  c1->Divide(2,2);
//
//  Part (a) - Generate a histogram with exactly 200 events.
//
  c1->cd(1);
  for ( int ievt=0; ievt<y; ievt++ ) {
    double u = rnd->Uniform();
    double x = ((p-1)+sqrt((p-1)*(p-1)+4*p*u))/(2*p);
    h_x->Fill(x,1);
  }
  h_x->Draw();
  h_x->GetXaxis()->SetTitle("x = E_{#pi}/E_{#tau}");
  h_x->GetYaxis()->SetTitle("dN/dx (entries per 0.1)");

//
//  Part (b) - Generate Poisson distributed random numbers with mean y.
//
  c1->cd(2);
  for ( int iexp=0; iexp<nexp; iexp++ ) {
    int n = rnd->Poisson(y);
    h_n->Fill(n,1);
  }
  h_n->Draw();
  cout << "RMS of yield distribution = " << h_n->GetRMS() << endl;
  cout << "Expected RMS = sqrt(" << y << ") = " << sqrt(y) << endl;

//
//  Part (c) - Fit the generated polarization distributions
//
  c1->cd(3);
  TH1F *h_dp = new TH1F("dp","P(fit)-P(true) - #chi^{2} fit",100,-1,1);
  TH1F *h_prob = new TH1F("prob","P(#chi^{2},N_{dof})",100,0,1);
  h_prob->SetMinimum(0);
  for ( int iexp=0; iexp<nexp; iexp++ ) {
    int n = rnd->Poisson(y);
    h_x->Reset();
    for ( int ievt=0; ievt<n; ievt++ ) {
      double u = rnd->Uniform();
      double x = ((p-1)+sqrt((p-1)*(p-1)+4*p*u))/(2*p);
      h_x->Fill(x,1);
    }
    TF1 *func = new TF1("fdndx","[0]*(1+[1]*(2*x-1))",0,1);
    func->FixParameter(0,n*h_x->GetBinWidth(1));
    h_x->Fit("fdndx");
    h_dp->Fill(func->GetParameter(1)-p,1);
    h_prob->Fill(TMath::Prob(func->GetChisquare(),h_x->GetNbinsX()-1),1);
  }
  h_dp->Draw();
//
//  Part (d) - Fit the generated polarization distributions
//
  c1->cd(4);
  TH1F *h_dpl = new TH1F("dpl","P(fit)-P(true) - likelihood fit",100,-1,1);
```

```
  TH1F *h_errpl = new TH1F("errpl","Uncertainty in P(fit) - likelihood fit",100,0,0.5);
  TH1F *h_probl = new TH1F("probl","P(#chi^{2},N_{dof})",100,0,1);
  for ( int iexp=0; iexp<nexp; iexp++ ) {
    int n = rnd->Poisson(y);
    h_x->Reset();
    for ( int ievt=0; ievt<n; ievt++ ) {
      double u = rnd->Uniform();
      double x = ((p-1)+sqrt((p-1)*(p-1)+4*p*u))/(2*p);
      h_x->Fill(x,1);
    }
    TF1 *func = new TF1("fdndx","[0]*(1+[1]*(2*x-1))",0,1);
    func->FixParameter(0,n*h_x->GetBinWidth(1));
    h_x->Fit("fdndx","L");
    h_dpl->Fill(func->GetParameter(1)-p,1);
    h_errpl->Fill(func->GetParError(1),1);
    h_probl->Fill(TMath::Prob(func->GetChisquare(),h_x->GetNbinsX()-1),1);
  }
  h_dpl->Draw();
}
```

FORM C
APPROVED FOR USE IN
PURDUE UNIVERSITY

2 (a) See attached.

(b) If the templates are y_i^+ and y_i^- for bin i, then the expected number of events in bin i is

$$\mu_i = n_+ y_i^+ + n_- y_i^-$$

Then, the probability of observing n_i events in bin i is Poisson distributed:

$$P(n_i | \mu_i) = \frac{\mu_i^{n_i} e^{-\mu_i}}{n_i!}$$

The likelihood function is

$$L = \prod_i P(n_i | \mu_i)$$

and $$\log L = \sum_i \left(n_i \log \mu_i - \mu_i \right) + \text{const.}$$

(c) See attached code

(d) The observed distribution can be written

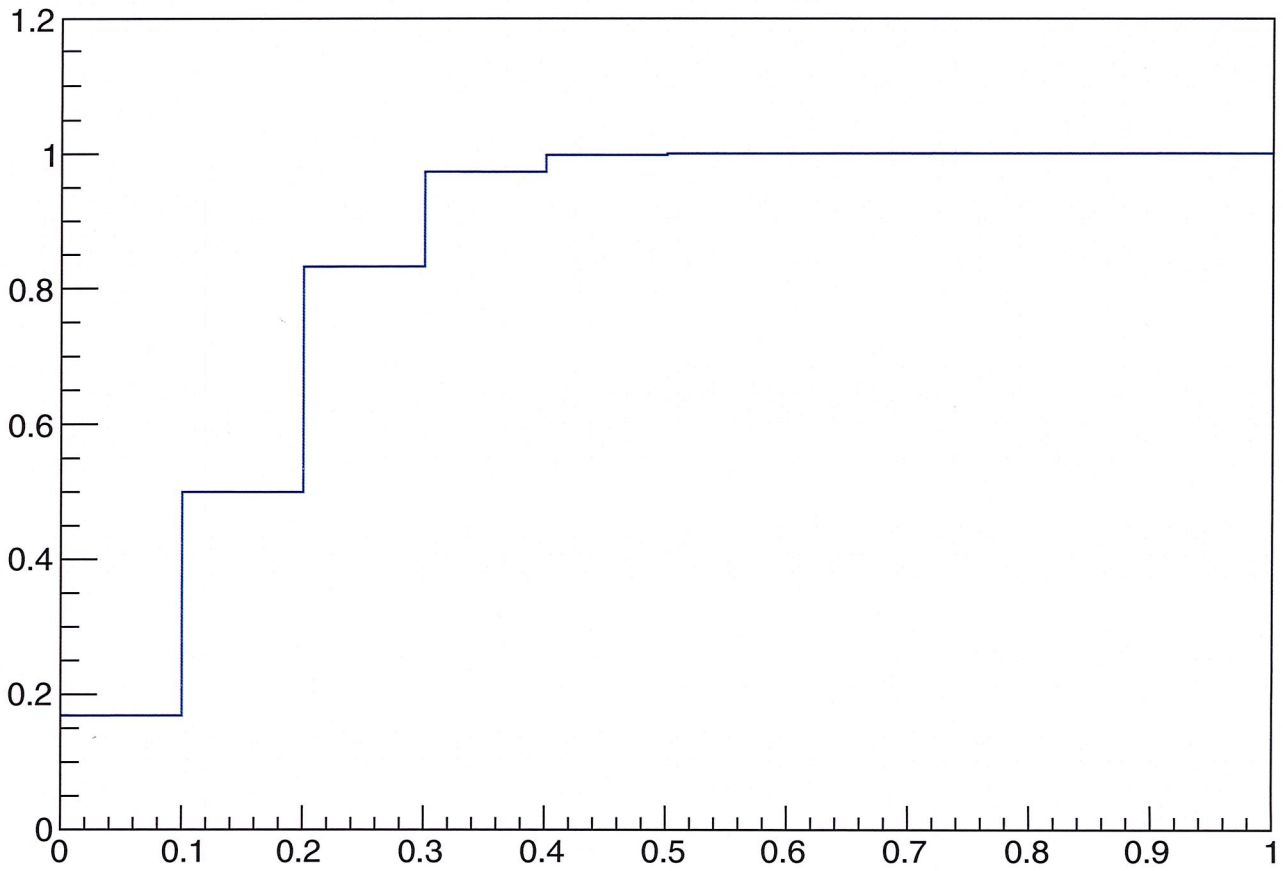
$$\frac{d\Gamma}{dx} = n_+ \left(1 + (2x - 1) \right) + n_- \left(1 - (2x - 1) \right)$$

Thus, $$\frac{1}{\Gamma} \frac{d\Gamma}{dx} = f_+ \left(1 + (2x - 1) \right) + f_- \left(1 - (2x - 1) \right)$$

where $f_+ = n_+ / (n_+ + n_-)$ and $f_- = n_- / (n_+ + n_-) = 1 - f_+$

Then, $$\frac{1}{\Gamma} \frac{d\Gamma}{dx} = 1 + (2f_+ - 1)(2x - 1)$$

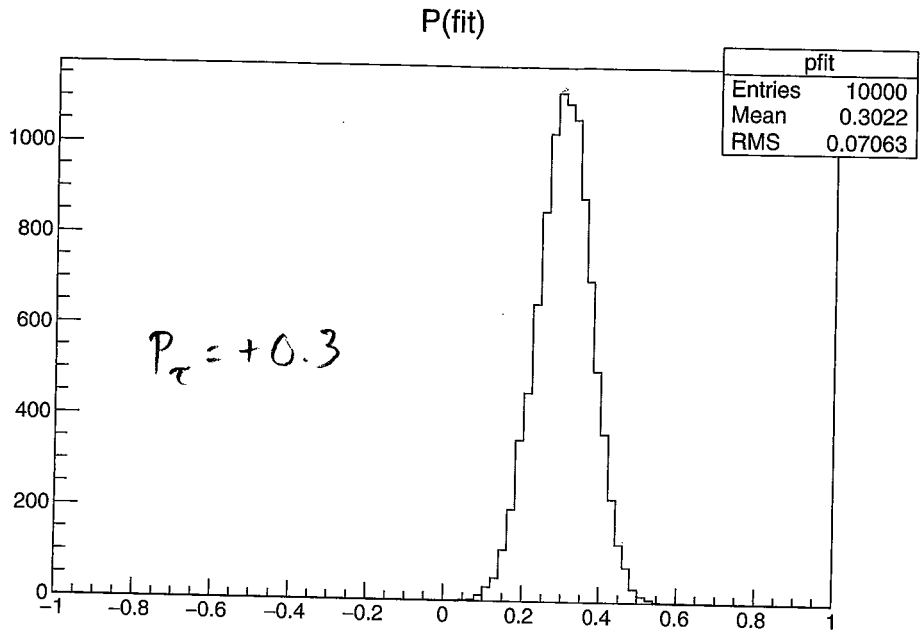
So $$\langle P_\tau \rangle = 2f_+ - 1 = \frac{n_+ - n_-}{n_+ + n_-}.$$

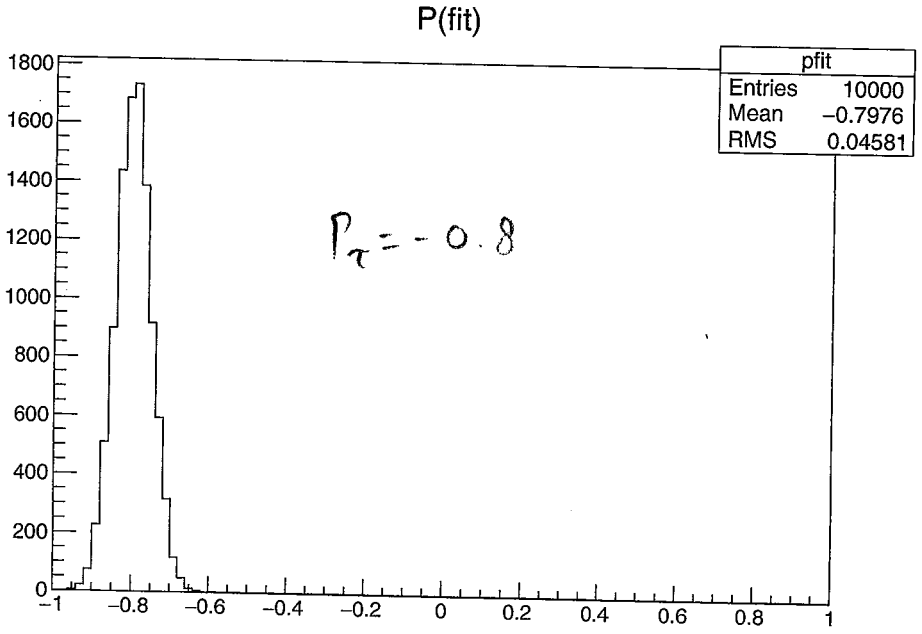

dN/dx - acceptance
1.2
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

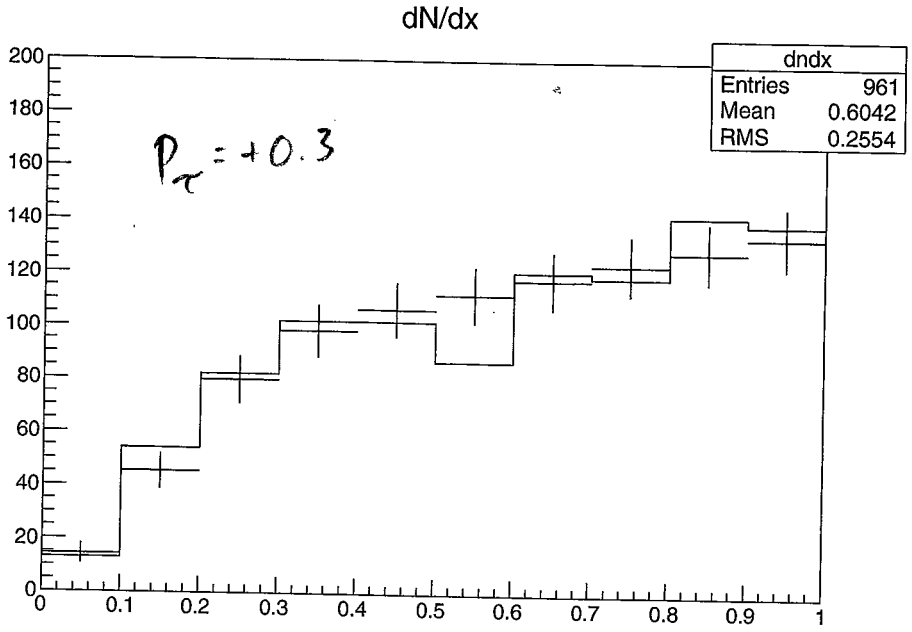
(e) See attached

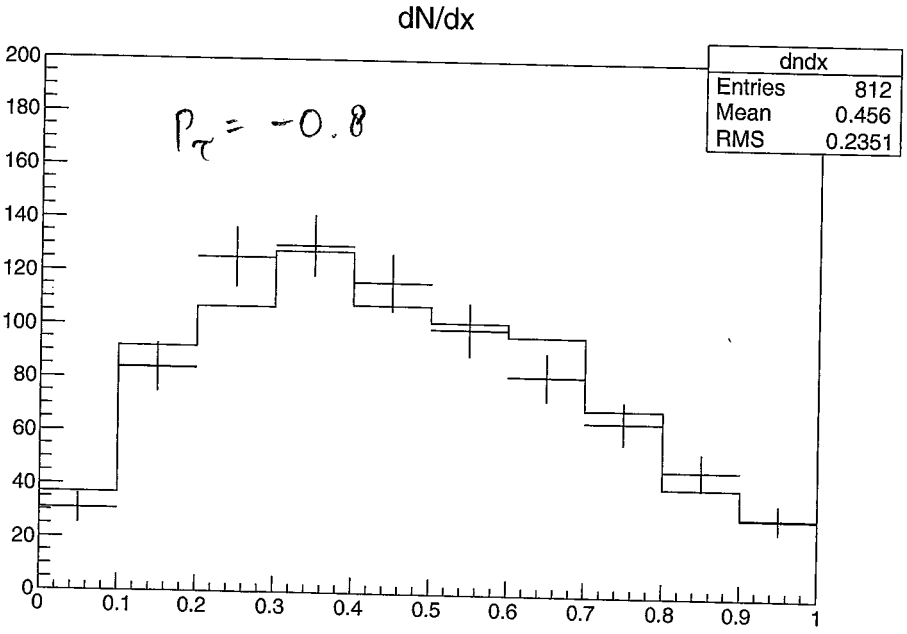
(f) See attached.

FORM C
APPROVED FOR USE IN
PURDUE UNIVERSITY


P(fit)
pfit
Entries 10000
Mean 0.3022
RMS 0.07063
$P_\tau = +0.3$
1000
800
600
400
200
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1


P(fit)
pfit
Entries 10000
Mean −0.7976
RMS 0.04581
$P_\tau = -0.8$
1800
1600
1400
1200
1000
800
600
400
200
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1


dN/dx
dndx
Entries 961
Mean 0.6042
RMS 0.2554
$P_\tau = +0.3$
200
180
160
140
120
100
80
60
40
20
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1


dN/dx
dndx
Entries 812
Mean 0.456
RMS 0.2351
$P_\tau = -0.8$
200
180
160
140
120
100
80
60
40
20
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

```
TH1F *h_p[2], *h_x;

void fcn(int &npar,double *gin,double &f,double *par,int iflag) {
  double n = h_x->GetEntries();
  double logl = 0;
  double np = par[0];
  double nm = par[1];

  for ( int i=1; i<=h_x->GetNbinsX(); i++ ) {
    double x = h_x->GetBinCenter(i);
    double y = h_x->GetBinContent(i);
    double z = np*h_p[1]->GetBinContent(i)+nm*h_p[0]->GetBinContent(i);
    if ( z > 0 ) {
      logl += y*log(z)-z;
    }
  }
  f = -2*logl;
}

void TauPolarizationTemplates(double p) {

  int ntoy = 10000;
  int nexp = 1000000;
  double y = 1000;

  double ethr = 0.15;
  double eres = 0.10;
  int nbin = 10;

  TH1F *h_pl = new TH1F("pl","dN/dx - P=-1",nbin,0,1);
  TH1F *h_pr = new TH1F("pr","dN/dx - P=+1",nbin,0,1);
  h_x = new TH1F("dndx","dN/dx",nbin,0,1);
  h_p[0] = h_pl;
  h_p[1] = h_pr;

  TH1F *h_ngen = new TH1F("ngen","dN/dx - generated",nbin,0,1);
  TH1F *h_nacc = new TH1F("nacc","dN/dx - accepted",nbin,0,1);
  TH1F *h_acc = new TH1F("acc","dN/dx - acceptance",nbin,0,1);
  TRandom *rnd = new TRandom();

//
//  Generate template distributions
//
  double plr[2] = { -1, +1 };
  for ( int ievt=0; ievt<nexp; ievt++ ) {
    for ( int i=0; i<2; i++ ) {
      double u = rnd->Uniform();
      double q = plr[i];
      double x = u;
      if ( q != 0 ) x = ((q-1)+sqrt((q-1)*(q-1)+4*q*u))/(2*q);
      h_ngen->Fill(x,1);
      u = rnd->Uniform();
      if ( u < 0.5+0.5*TMath::Erf((x-ethr)/eres/sqrt(2)) ) {
        h_nacc->Fill(x,1);
        h_p[i]->Fill(x,1);
      }
      else {
        h_p[i]->Fill(-1,1);
      }
    }
  }
//
//  Normalize the template distributions
//
```

```
  int npl = h_pl->GetEntries();
  int npr = h_pr->GetEntries();
  for ( int i=0; i<=h_pl->GetNbinsX(); i++ ) {
    h_pl->SetBinContent(i,h_pl->GetBinContent(i)/npl);
    h_pr->SetBinContent(i,h_pr->GetBinContent(i)/npr);
    h_acc->SetBinContent(i,h_nacc->GetBinContent(i)/h_ngen->GetBinContent(i));
  }

  int ierr;
  TMinuit *min = new TMinuit(2);
  min->SetFCN(fcn);

  TH1F *h_pfit = new TH1F("pfit","P(fit)",100,-1,1);
  TH1F *h_pfl = new TH1F("pfl","Fit P=-1",nbin,0,1);
  TH1F *h_pfr = new TH1F("pfr","Fit P=+1",nbin,0,1);
  TH1F *h_pflr = new TH1F("pflr","Fit",nbin,0,1);

  for ( int iexp=0; iexp<ntoy; iexp++ ) {
    h_x->Reset();

//
//  Generate one toy experiment...
//
    int nevt = rnd->Poisson(y);
    for ( int ievt=0; ievt<nevt; ievt++ ) {
      double u = rnd->Uniform();
      double x = u;
      if ( p != 0 ) x = ((p-1)+sqrt((p-1)*(p-1)+4*p*u))/(2*p);
      u = rnd->Uniform();
      if ( u < 0.5+0.5*TMath::Erf((x-ethr)/eres/sqrt(2)) ) {
        h_x->Fill(x,1);
      }
    }

//
//  Fit the toy experiment...
//
    min->mncler();
    min->mnparm(0,"np",y/2,0.1,0,y,ierr);
    min->mnparm(1,"nm",y/2,0.1,0,y,ierr);
    if ( min->Migrad() == 0 ) {
      double np, enp;
      min->GetParameter(0,np,enp);
      double nm, enm;
      min->GetParameter(1,nm,enm);
      double f = np/(np+nm);
      double fp = 2*f-1;

      h_pfit->Fill(fp,1);

      for ( int i=1; i<=h_x->GetNbinsX(); i++ ) {
        double zp = np*h_p[1]->GetBinContent(i);
        h_pfr->SetBinContent(i,zp);
        double zm = nm*h_p[0]->GetBinContent(i);
        h_pfl->SetBinContent(i,zm);
        h_pflr->SetBinContent(i,zp+zm);
      }
    }
  }
  h_pfit->Draw();
}
```