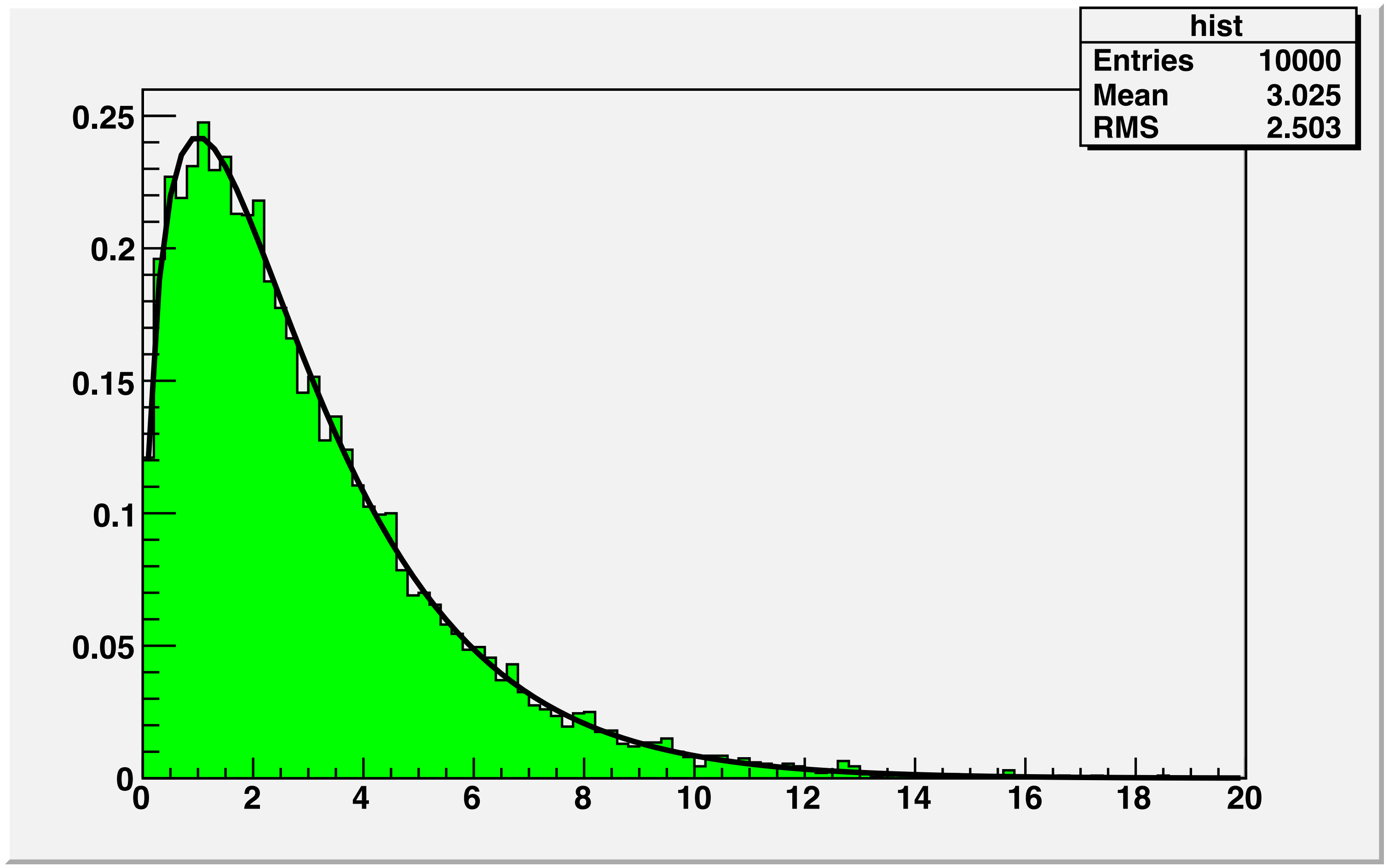
Chi-Square Distribution

How to make - A short explanation

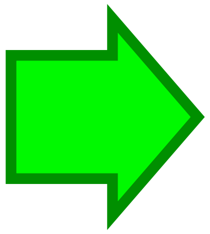
Goodness-of-fit with chi-square

Demonstration of the chi-square distribution:



line: function drawing of chi-square when n=3

histogram: distribution of chi-squares for 10000 independent fits



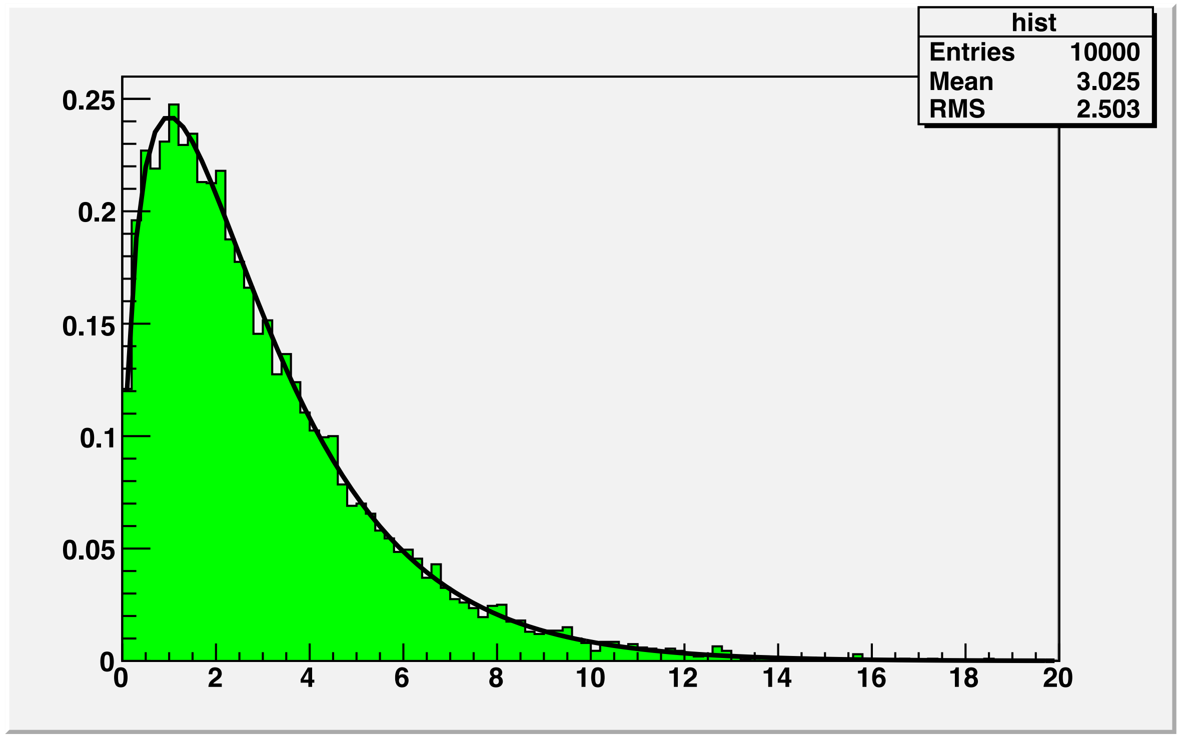
They agree well each other!

Can you write a ROOT program that produces the above plot?
I'll give you only the algorithm this time (see the next slide)

Goodness-of-fit with chi-square

Demonstration of the chi-square distribution: my algorithm (Of course this is not the only way of doing this)

1) Generate *N* random numbers r_i in [0,1]
2) Divide [0,1] into *n* bins
3) Count the number r_i belongs to each bin and save to y_i
4) Error on each bin is e_i = sqrt(y_i)
5) Perform a 0th-order polynomial fit on {y_i,e_i}
6) Get the chi-square of the fit in 5) and save it into your histogram
7) Repeat 1) - 6) M times and draw the histogram
8) Draw the chi-square function of n.d.f=*n-1* over the histogram



N = 1000
n = 4
M = 10000 in my case of the figure left

Q: How can I get the value of χ^2 *after the fit? (amin in the below codes is the one)*

```
Double_t amin,edm,errdef;
Int_t nvpar,nparx,icstat;
gMinuit->mnstat(amin,edm,errdef,nvpar,nparx,icstat);
```

The source code: chi_minuit.C

```
#include "TMinuit.h"

#define NGEN1 10000
#define NGEN2 1000
#define NBIN 100
#define XMAX 20.0
#define XMIN  0.0

gROOT->SetStyle("Plain");
gROOT->ForceStyle();

Int_t n = 11; // ndf + 1 = n
Double_t *x1 = new Double_t[n];
Double_t *y1 = new Double_t[n];
Double_t *e1 = new Double_t[n];

float gammln(float xx)
{
        double x,y,tmp,ser;
        static double cof[6]={76.18009172947146,-86.50532032941677,
                24.01409824083091,-1.231739572450155,
                0.1208650973866179e-2,-0.5395239384953e-5};
        int j;

        y=x=xx;
        tmp=x+5.5;
        tmp -= (x+0.5)*log(tmp);
        ser=1.000000000190015;
        for (j=0;j<=5;j++) ser += cof[j]/++y;
        return -tmp+log(2.5066282746310005*ser/x);
}

Double_t f1(Double_t *x, Double_t *n)
{
  Double_t lnf = TMath::Log(1) - n[0]*0.5*TMath::Log(2.0) - gammln(n[0]*0.5)
               + (n[0]*0.5-1.0)*TMath::Log(x[0]) - x[0]*0.5;
  return TMath::Exp(lnf);
}

Double_t func(Double_t x, Double_t *par)
{
  return par[0];
}

void fcn(Int_t &npar, Double_t *gin, Double_t &f, Double_t *par, Int_t iflag)
{
  Int_t i;

  Double_t chisq = 0;
  Double_t delta;

  for (i=0;i<n;i++) {
   delta  = (y1[i]-func(x1[i],par))/e1[i];
   chisq += delta*delta;
  }
  f = chisq;
}
```

```
void chi_minuit()
{
   gRandom->SetSeed();

   TCanvas* my_canvas = new TCanvas("my_canvas","",200,10,600,400);

   hist = new TH1F("hist","",NBIN,XMIN,XMAX);
   hist->SetFillColor(3);

   Double_t vstart[1] = {0.0};
   Double_t step[1] = {0.001};

   //
   // random number generation
   //
   Int_t dummy;
   Float_t u1,u2,z1,z2;
   for (Int_t i=0;i<NGEN1;i++)
   {
     for (Int_t j=0; j<n;j++) y1[j] = 0.0;
     for (Int_t j=0; j<NGEN2; j++)
     {
       u1 = gRandom->Rndm(dummy);
       for (Int_t k=0;k<n;k++)
       {
         if ((u1 >= k/(1.0*n) && (u1 < (k+1)/(1.0*n))))
         y1[k]++;
       }
     }
     for (Int_t l=0;l<n;l++)
     {
       e1[l] = TMath::Sqrt(y1[l]);
     }
     //
     // calls MINUIT explicitly
     //
     TMinuit *gMinuit = new TMinuit(1);  // initialize TMinuit with a maximum of 3 params
     gMinuit->SetFCN(fcn);               // set the minimization function
     Double_t arglist[10];
     Int_t ierflg = 0;

     arglist[0] = 1;
     gMinuit->mnexcm("SET ERR", arglist ,1,ierflg);
     gMinuit->mnparm(0, "theta_0", vstart[0], step[0], 0,0,ierflg);

     arglist[0] = 5000;
     arglist[1] = 1.;
     gMinuit->mnexcm("MINUIT", arglist ,2,ierflg);

     Double_t amin,edm,errdef;
     Int_t nvpar,nparx,icstat;
     gMinuit->mnstat(amin,edm,errdef,nvpar,nparx,icstat);
     hist->Fill(amin);
   }
   hist->Scale(NBIN/(1.0*NGEN1*(XMAX-XMIN)));
   hist->Draw();

   TF1 *fun1 = new TF1("fun1",f1,XMIN,XMAX,1);
   fun1->SetParameter(0, n-1); fun1->Draw("LSAME");
}
```