ROOT'a GİRİŞ

İlkay TÜRK ÇAKIR

İSTANBUL ÜNİVERSİTESİ

3-7 Şubat, 2020

GENEL BİLGİLER

- ROOT, 1994 yılında René Brun ve Fons Rademakers tarafından CERN'de (Avrupa Nükleer Araştırma Merkezi) geliştirilmeye başlanan, bilimsel amaçlı, özellikle deneysel parçacık fiziği alanında kullanılan, açık kaynak kodlu bilimsel bir veri analiz paketidir.

- ROOT parçacık fiziği veri analizi için tasarlanmış olmasına rağmen aynı zamanda astronomi ve veri madenciliği gibi diğer uygulamalarda da kullanılmaktadır.

- ROOT'un en önemli özelliği, ROOT'un içerisinde Tree adında, içerisinde branchleri ve leaveleri olan bir veri taşıyıcı olmasıdır. Bu veri taşıyıcısı içerisine yüklenen verileri saklar ve gerektiğinde ROOT platformu üzerinde çeşitli kullanım imkanları sağlar. Bu özellik verilerin kaybolması, saklanması, bellek gereksinimleri gibi sorunların önüne geçer.

GENEL BİLGİLER

- Yılda birkaç petabaytlık veri alan LHC (Büyük Hadron Çarpıştırıcısı - CERN) deneylerinin verilerini işleme esnasında yüksek işlem verimliliği için tasarlanan ROOT, 2009 yılından itibaren yüksek enerji fiziğindeki veri analizi ve veri toplamada kullanılan en güncel sistem haline geldi.

- ROOT'un günümüzde CERN dahil birçok önemli bilimsel merkezde kullanılan temel bir veri analiz paketi olmasının nedeni: ROOT'un verileri kendi formatında (.root) kaydetmesi ve bu verileri yüksek bir işlem verimliliği içinde geri yükleyerek kullanabilmesidir. Bu nedenle bir çok deneyde alınan veriler ROOT formatında alınıp işleme ROOT ile devam edilmektedir. İçerisindeki önemli kütüphaneler de ROOT'un kullanım nedenleri arasındadır.

- ROOT içerisinde programları derlemek için yüklü bir C++ yorumlayıcısı (5.34 sürümüne kadar CINT, 6.00 sürümünden sonra CLING) ile gelmektedir.

ROOT

- ROOT Windows, Linux ve MacOS işletim sistemlerinde desteklenmektedir.
- ROOT kendi içinde bir C++ **yorumlayıcı** bulundurur.
- Derleyiciye ihtiyaç duyulmadan C/C++ kodları ile çalışılabilir.
- Etkileşimli komut satırı "root" yazarak başlatılır.
- root –l ile de başlatılabilir.

ROOT'DA ÖZEL KOMUTLAR

ROOT içinde olan fakat C++'da olmayan özel komutlar bir . işareti ile birlikte yazılır.

- Örnekler:
  - ROOT'dan çıkmak için .q
  - kabuk komutu kullanmak için .!
  - ROOT'u çalıştırmak için root veya root -l
  - bir makro yüklemek için .L dosya adı
  - bir makro çalıştırmak için .x dosya adı
  - yardım için .help veya .?

Makro: Belli bir işi yapmak üzere her istediğinizde çalıştırabileceğiniz bir dosya içerisine yazılmış komutlar ve fonksiyonlar dizisidir.
ROOT'da ise çizimle ilgili veya hesapla ilgili komutları yazdığımız betik (script)

ROOT UYGULAMALARI

- ✓ ROOT içinde çalıştırma
- ✓ ROOT'da hesaplama
- ✓ ROOT'da Grafik -1
- ✓ ROOT'da Grafik -2
- ✓ ROOT'da Histogram
- ✓ İsimsiz Makro
- ✓ İsimli Makro
- ✓ Histogram ve Grafiği Üstüste Çizdirmek
- ✓ Kanvas Yapmak -1
- ✓ Kanvas Yapmak -2
- ✓ Dağılımı Bir fonksiyona fit etmek
- ✓ Eksenleri isimlendirme
- ✓ Etiket Ekleme

- ✓ TGraph hata grafiği-1
- ✓ TGraph Hata Garfiği -2
- ✓ Histogram Fit
- ✓ Histogram Normalizayon
- ✓ İki boyutlu grafik
- ✓ Yığın (Stack) Grafiği
- ✓ ROOT GUI mod
- ✓ GUI'den fonksiyona parameter aktarma
- ✓ Veri dosyasından okuma ve grafik çizme
- ✓ Root dosyalarının yazılması
- ✓ Komut satırında root dosyasındaki histogramı çizdirebilme
- ✓ Ek Örrnek
- ✓ Satır ve Sütun Şeklindeki Bilgiyi okuma ve yazma (NTuple)

ROOT'DA HESAPLAMA

Matematik fonksiyon f(x)=1/(1-x) seriye açılabilir.

Problem:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \ldots$$
$$= \sum_{n=0}^{\infty} x^n$$

Çözüm: Bu hesap için önce değişken tanımlanır ve sonra for kontrol yapısı kullanılır.

```
ILKAYs-MBP:PHBUO ilkayturkcakir$ root -l
root [0] double x=.5
 (double) 0.50000000
root [1] int N=30
 (int) 30
root [2] double gs=0;
root [3] for (int i=0; i<N; ++i) gs +=pow(x,i)
root [4] std::abs(gs-(1/(1-x)))
 (double) 1.8626451e-09
root [5] gs
 (double) 2.0000000
 root [6]
```

ROOT İÇİNDE ÇALIŞTIRMA

ROOT içinde makro ile çalışma (1.C)

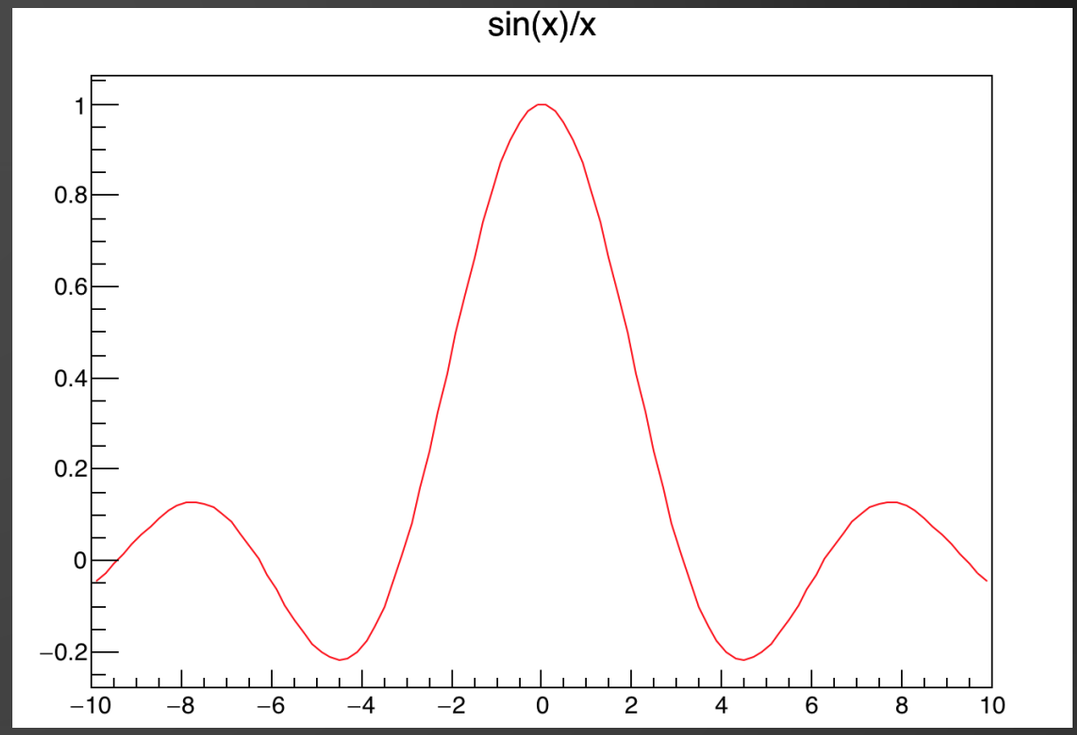
```
{

// °C derecesini °F'ne çeviren program

Int_t ct;

cin>>ct;

cout<<ct*1.8+32<<endl;

}
```

```
ILKAYs-MBP:PHBUO ilkayturkcakir$ root
   ------------------------------------------------------------
  | Welcome to ROOT 6.18/04                        https://root.cern |
  |                                (c) 1995-2019, The ROOT Team |
  | Built for macosx64 on Sep 11 2019, 15:38:23                 |
  | From tags/v6-18-04@v6-18-04                                 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------

root [0] .x 1.C
100
212
root [1]
```

ROOT'DA GRAFİK-1

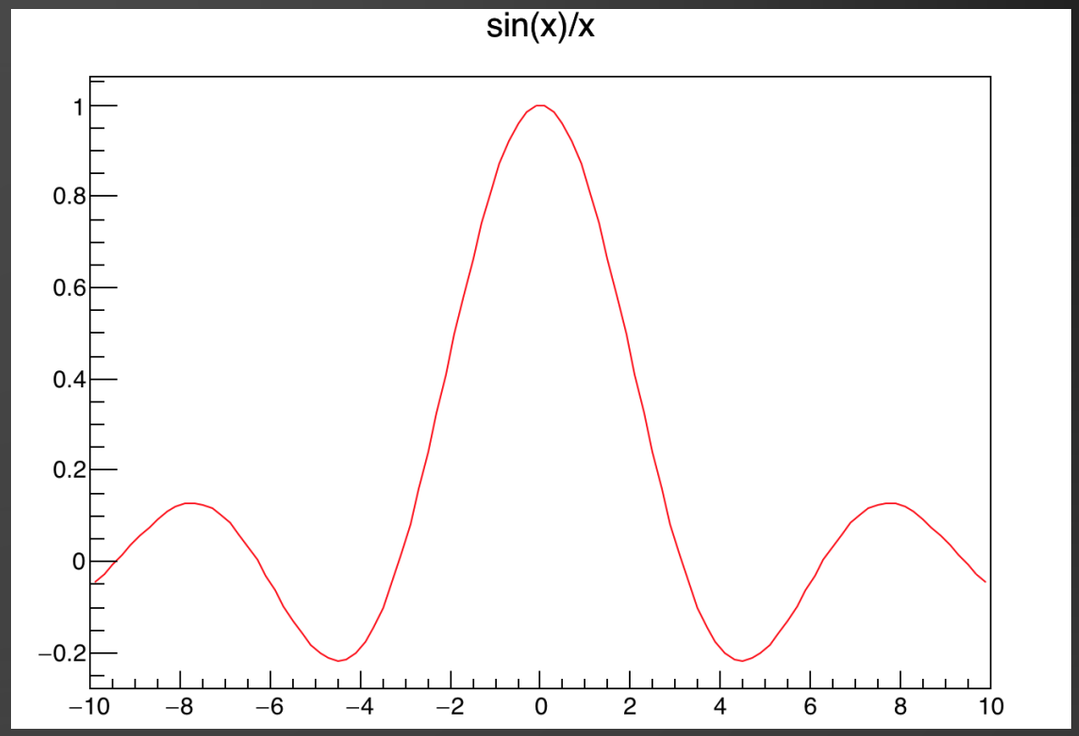
```
ILKAYs-MacBook-Pro:root_v6.18.04 ilkayturkcakir$ root
   ------------------------------------------------------------
  | Welcome to ROOT 6.18/04                        https://root.cern |
  |                                (c) 1995-2019, The ROOT Team |
  | Built for macosx64 on Sep 11 2019, 15:38:23                 |
  | From tags/v6-18-04@v6-18-04                                 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------

root [0] TF1 fonk1("","sin(x)/x",-10,10);
root [1] fonk1.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [2]
```

TF1 ile 1 değişkenli (1 boyutlu) fonksiyon tanımlanır, bu fonksiyon sin(x)/x dir.
x-in değer aralığı -10 ie 10 dur.



ROOT'DA GRAFİK-2

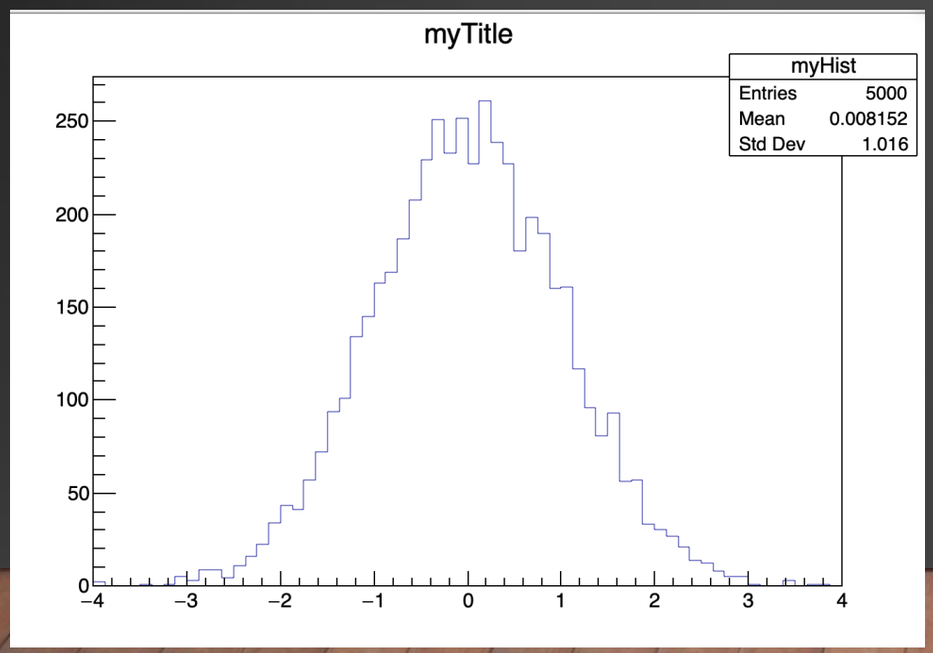
```
ILKAYs-MBP:PHBUO ilkayturkcakir$ root
   ------------------------------------------------------------
  | Welcome to ROOT 6.18/04                        https://root.cern |
  |                                (c) 1995-2019, The ROOT Team |
  | Built for macosx64 on Sep 11 2019, 15:38:23                 |
  | From tags/v6-18-04@v6-18-04                                 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------

root [0] TF1 *fonk1=new TF1("","sin(x)/x",-10,10)
(TF1 *) 0x7fc8beae8b90
root [1] fonk1->Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [2]
```



ROOT'DA HİSTOGRAM

TH1F ile 1 değişkenli (1-boyutlu) histogram tanımlanır, bu histogram rastgele Gauss dağılımı ile doldurulur, dağılımın ortalaması 0'dadır.

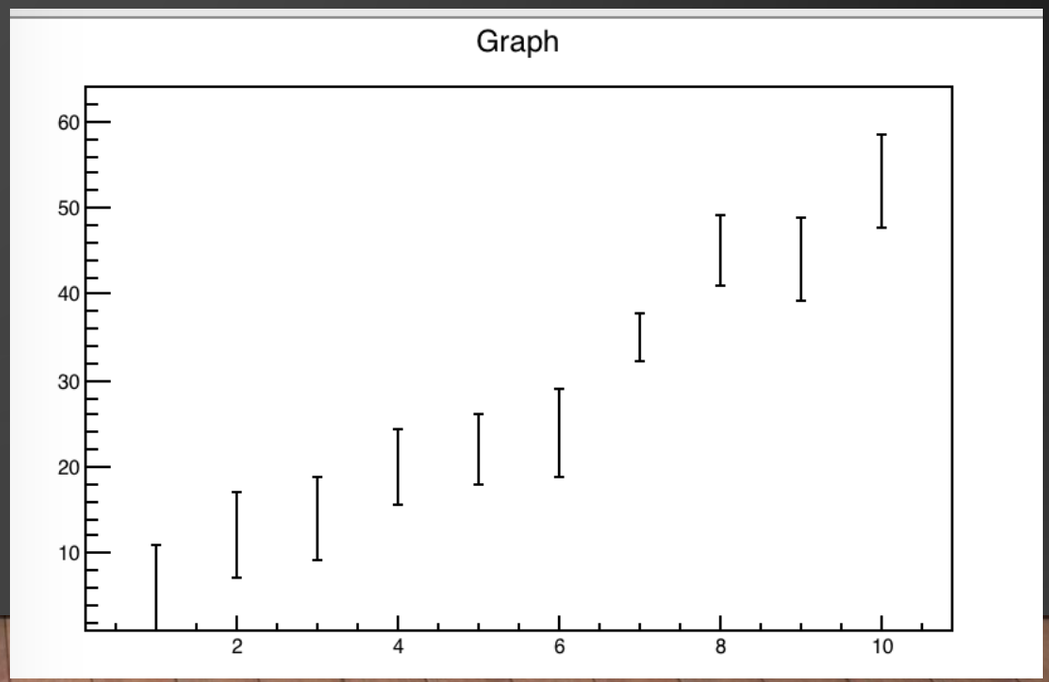
```
ILKAYs-MBP:PHBUO ilkayturkcakir$ root
   ------------------------------------------------------------
  | Welcome to ROOT 6.18/04                        https://root.cern |
  |                                (c) 1995-2019, The ROOT Team |
  | Built for macosx64 on Sep 11 2019, 15:38:23                 |
  | From tags/v6-18-04@v6-18-04                                 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------------

root [0] TH1F hist("myHist","myTitle",64,-4,4)
(TH1F &) Name: myHist Title: myTitle NbinsX: 64
root [1] hist.FillRandom("gaus")
root [2] hist.Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3]
```



İSİMSİZ MAKRO (FONKSİYONSUZ MAKRO)

- Makrolar isimsiz bir şekilde tanımlanabilir.
- Fonksiyon şeklinde çağrılmaz.
- ROOT içinde .x Makroadı.C şeklinde çalıştırılır.
- Örnek: macro1.C verilen x ve y değerleri ile y üzerindeki hataları içeren veri için yazılmış makro ve üretilen grafik

```
 GNU nano 2.0.6                          File: macro1.C

{
const int n_points = 10;
double x_val[n_points] = {1,2,3,4,5,6,7,8,9,10};
double y_val[n_points] = {6,12,14,20,22,24,35,45,44,53};
double y_err[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};


//Instance of the graph
auto graph = new TGraphErrors(n_points,x_val,y_val,nullptr,y_err);

graph->Draw("APE");

}
```



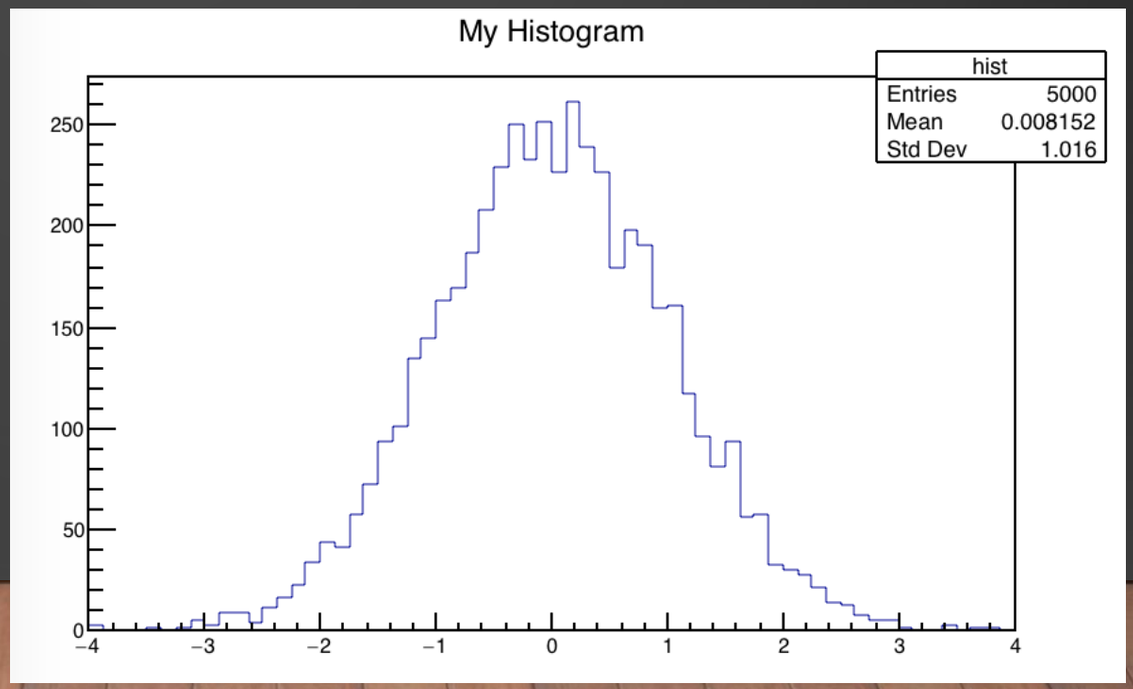
İSİMLİ MAKRO (FONKSİYONLU MAKRO)

İsimli makro fonksiyonun ismi makeHist(), root içinde .L macro2.C ile yüklenir, makeHist() ile de çalıştırılır. Gauss dağılımını veren grafik yanda gösterilmiştir.

```
 GNU nano 2.0.6                                File: macro2.C

void makeHist(){
TH1F *hist=new TH1F("hist","My Histogram",64,-4,4);
hist->FillRandom("gaus");
hist->Draw();
}
```

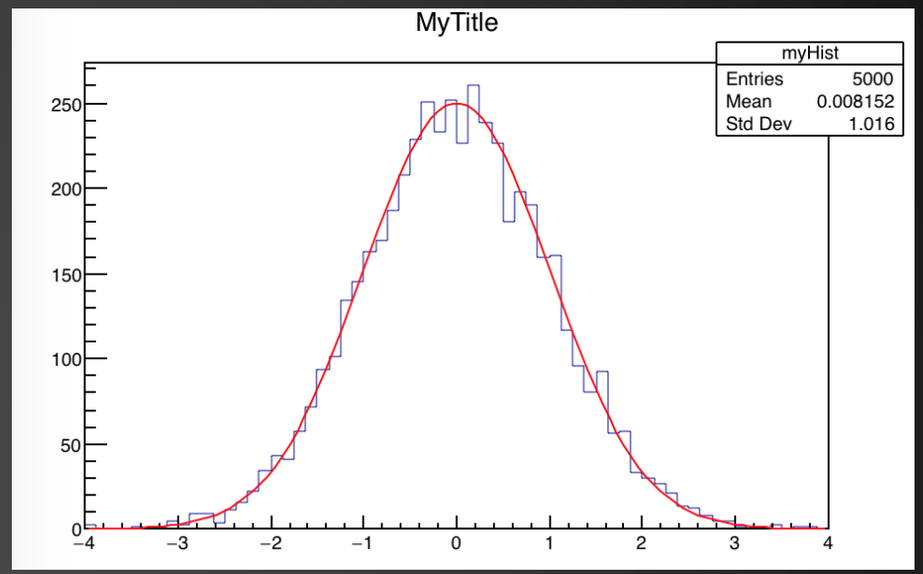
```
ILKAYs-MBP:PHBUO ilkayturkcakir$ root -l
root [0] .L macro2.C
root [1] make
makeHist
makedev
root [1] makeHist()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [2]
```

SORU: ROOT içinde .x macro2.C şeklinde çalıştırmak için kodda nasıl bir değişiklik yapmak gerekir?



HİSTOGRAM VE GRAFİĞİ ÜSTÜSTE ÇİZDİRMEK

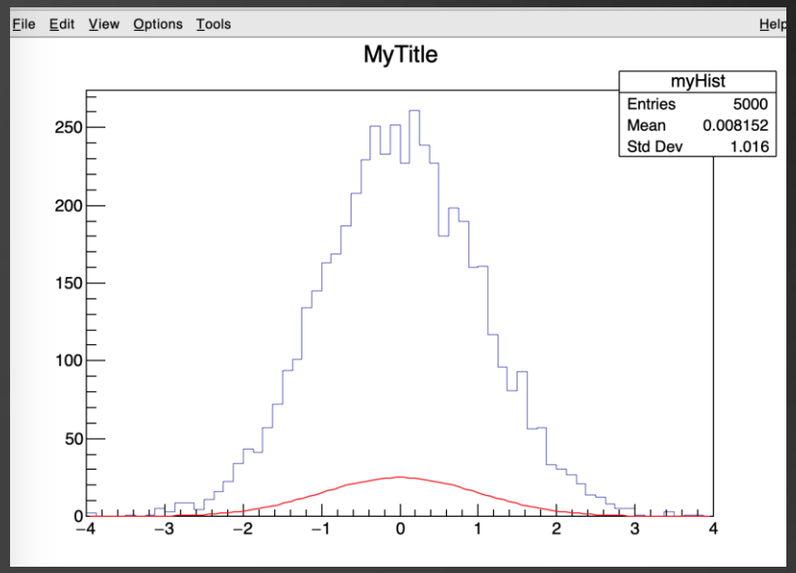
```
 GNU nano 2.0.6                                File: macro3.C

{
TH1F h("myHist","MyTitle",64,-4,4);
h.FillRandom("gaus");
h.Draw();

TF1 f("g","gaus",-8,8);
f.SetParameters(250,0,1);
f.Draw("Same");
}
```



İsimsiz makro içinde önce Histogram tanımlandı, sonra fonksiyon tanımlandı ve üstüste çizdirildi.

Not: Dağılımın ortalaması sıfır ve standart sapması 1, histogramda aralık sayısı 64'dür ve fonksiyonda nokta sayısı 250'dir.

SORU: Fonksiyonda nokta sayısı 25 alınırsa nasıl bir eğri beklersiniz?



KANVAS YAPMAK-1

ROOT içinde kanvas yapmak için:

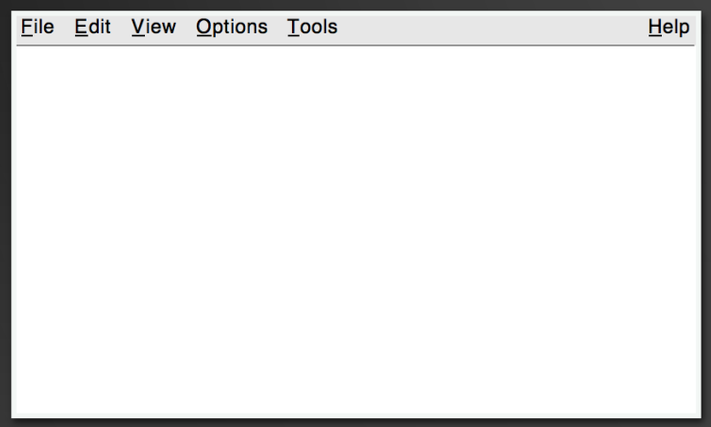
- TCanvas *c1 = new TCanvas(“c1”,”” 800,600);

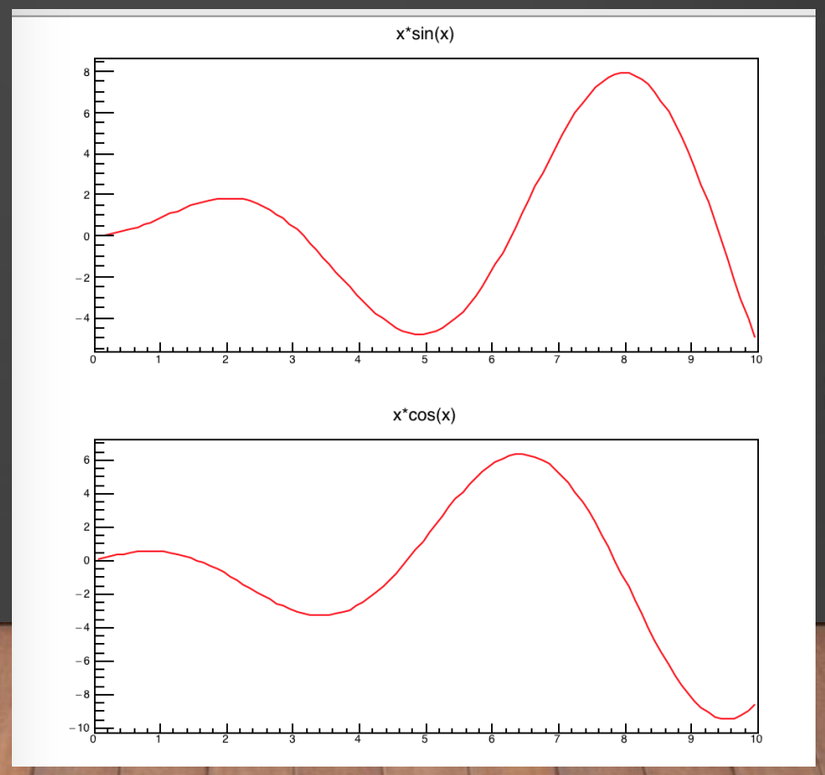
komutu yazılır. Bunun çizdirilmesi için:

- c1->Draw()

yazılır.

```
 GNU nano 2.0.6                File: kanvas.C

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


c1->Divide(1,2);

TF1 *f1 = new TF1("f1","x*sin(x)",0,10);
TF1 *f2 = new TF1("f2","x*cos(x)",0,10);

c1->cd(1);
f1->Draw();

c1->cd(2);
f2->Draw();


}
```



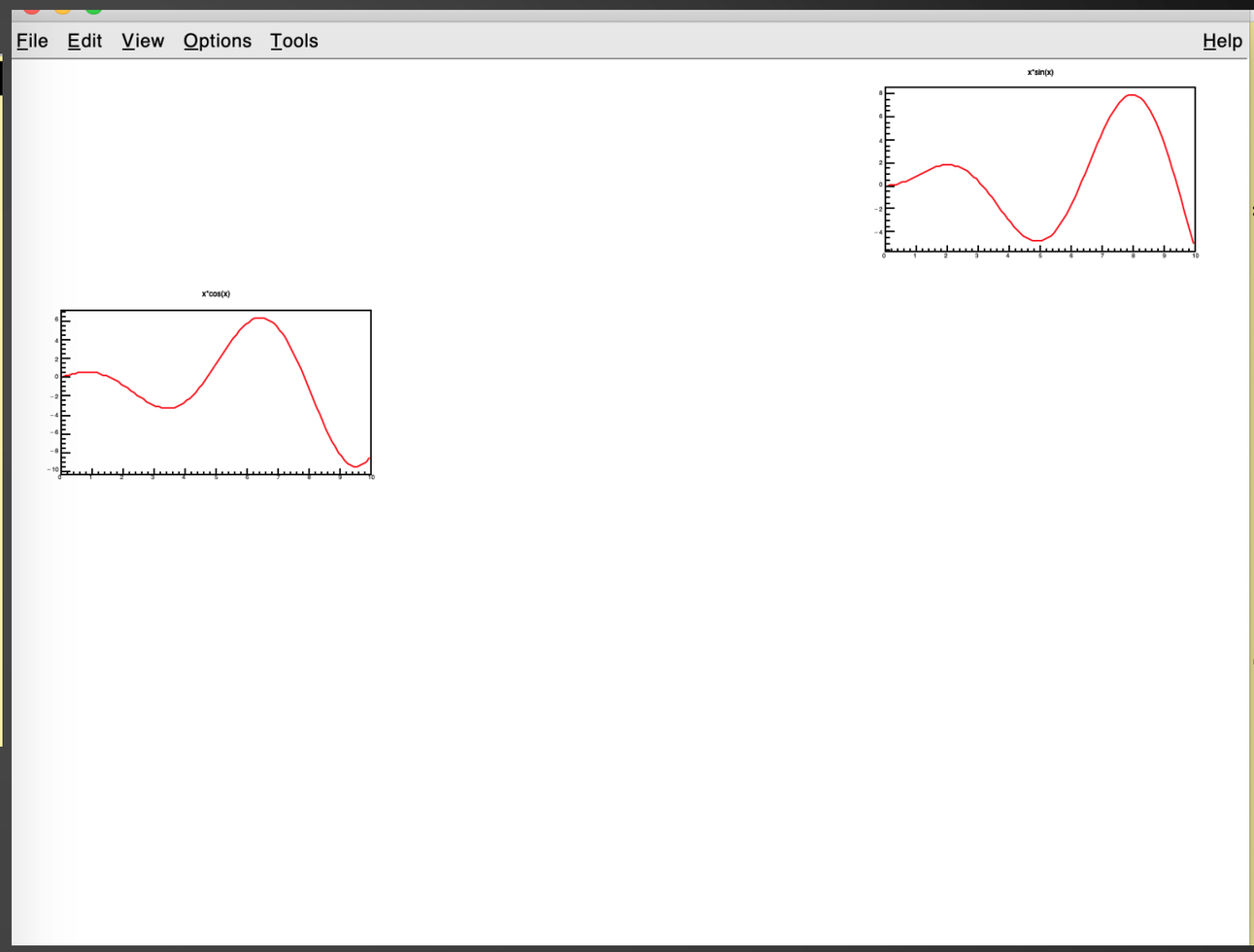


Kanvas ikiye bölündü.

SORU: Düşey ikiye bölmek isteseydik nasıl yapardık?

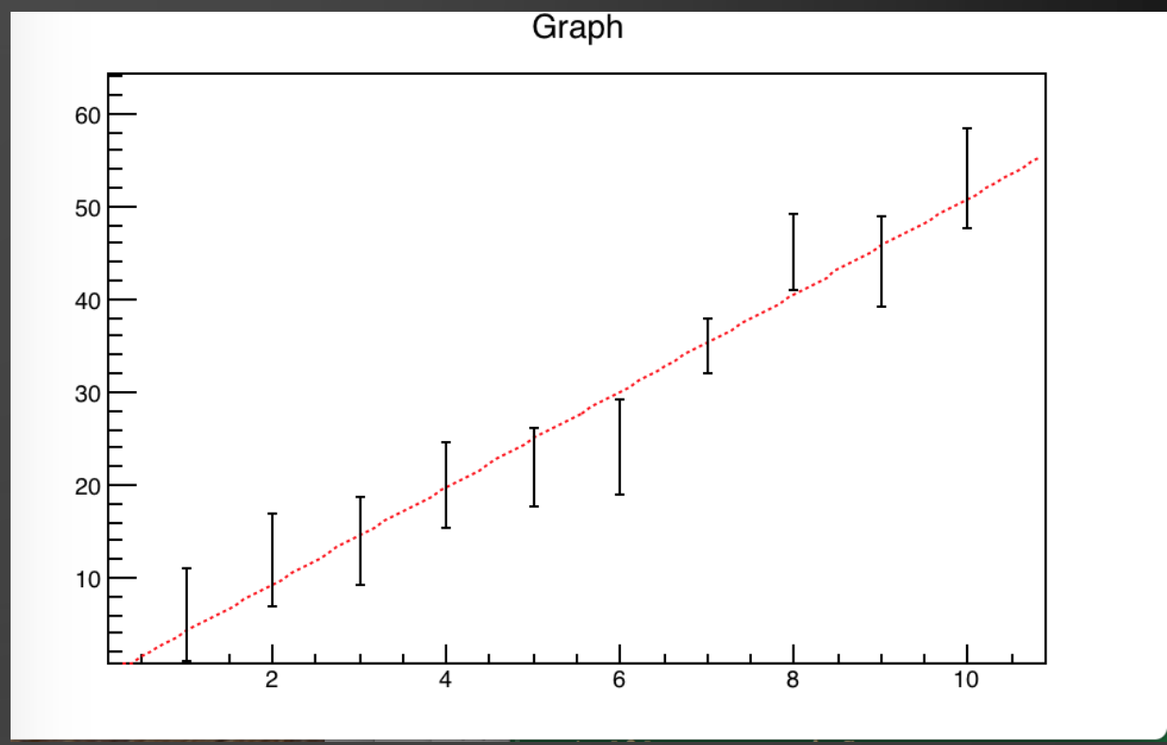
KANVAS YAPMAK-2

```
  GNU nano 2.0.6                File: kanvas.C

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

//3 sutun 4 satır
c1->Divide(3,4);

TF1 *f1 = new TF1("f1","x*sin(x)",0,10);
TF1 *f2 = new TF1("f2","x*cos(x)",0,10);

c1->cd(3);
f1->Draw();

c1->cd(4);
f2->Draw();


}
```



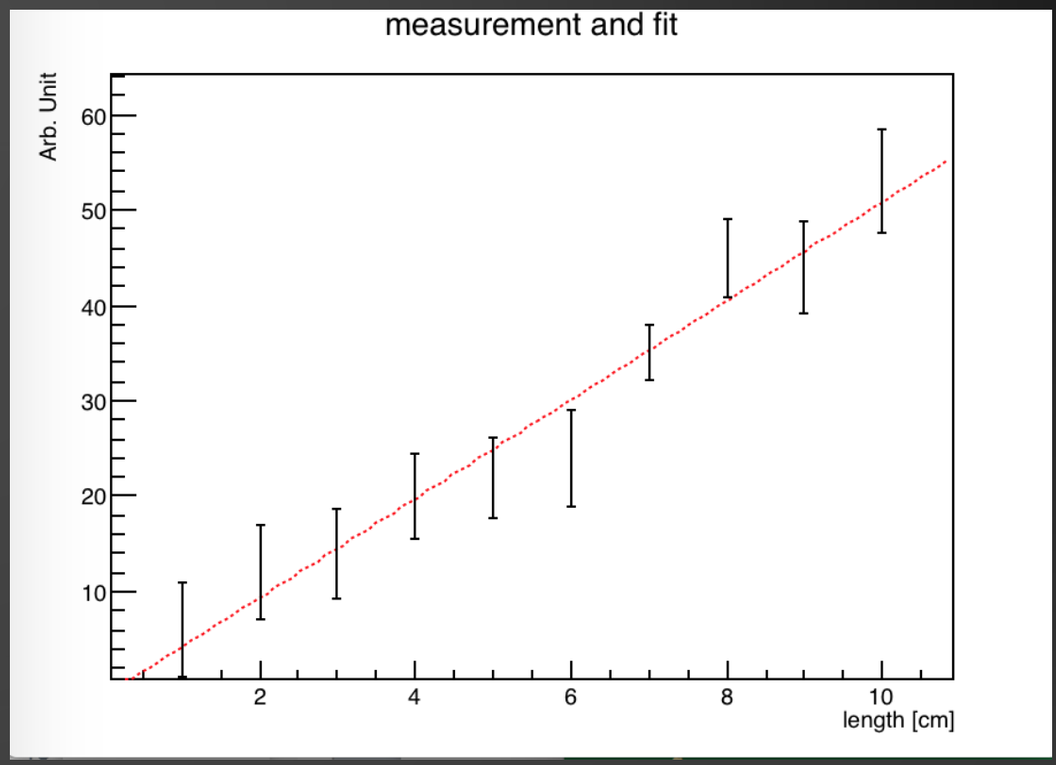
DAĞILIMI BİR FONKSİYONA FİT ETMEK

```
 GNU nano 2.0.6                          File: macro4.C

{
const int n_points = 10;
double x_val[n_points] = {1,2,3,4,5,6,7,8,9,10};
double y_val[n_points] = {6,12,14,20,22,24,35,45,44,53};
double y_err[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};


//Instance of the graph
auto graph = new TGraphErrors(n_points,x_val,y_val,nullptr,y_err);

graph->Draw("APE");

auto f = new TF1 ("Linear","[0]+x*[1]",.5,10.5);
f->SetLineColor(kRed);
f->SetLineStyle(2);
// Fit it to th graph

graph->Fit(f);

}
```



EKSENLERİ İSİMLENDİRME

```
 GNU nano 2.0.6                                File: macro5.C

{
const int n_points = 10;
double x_val[n_points] = {1,2,3,4,5,6,7,8,9,10};
double y_val[n_points] = {6,12,14,20,22,24,35,45,44,53};
double y_err[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};


//Instance of the graph
auto graph = new TGraphErrors(n_points,x_val,y_val,nullptr,y_err);

graph->Draw("APE");
graph->SetTitle("measurement and fit");
graph->GetXaxis()->SetTitle("lenght [cm]");
graph->GetYaxis()->SetTitle("Arb. Unit");

//define a function
auto f = new TF1 ("Linear","[0]+x*[1]",.5,10.5);
f->SetLineColor(kRed);
f->SetLineStyle(2);
// Fit it to th graph

graph->Fit(f);

}
```

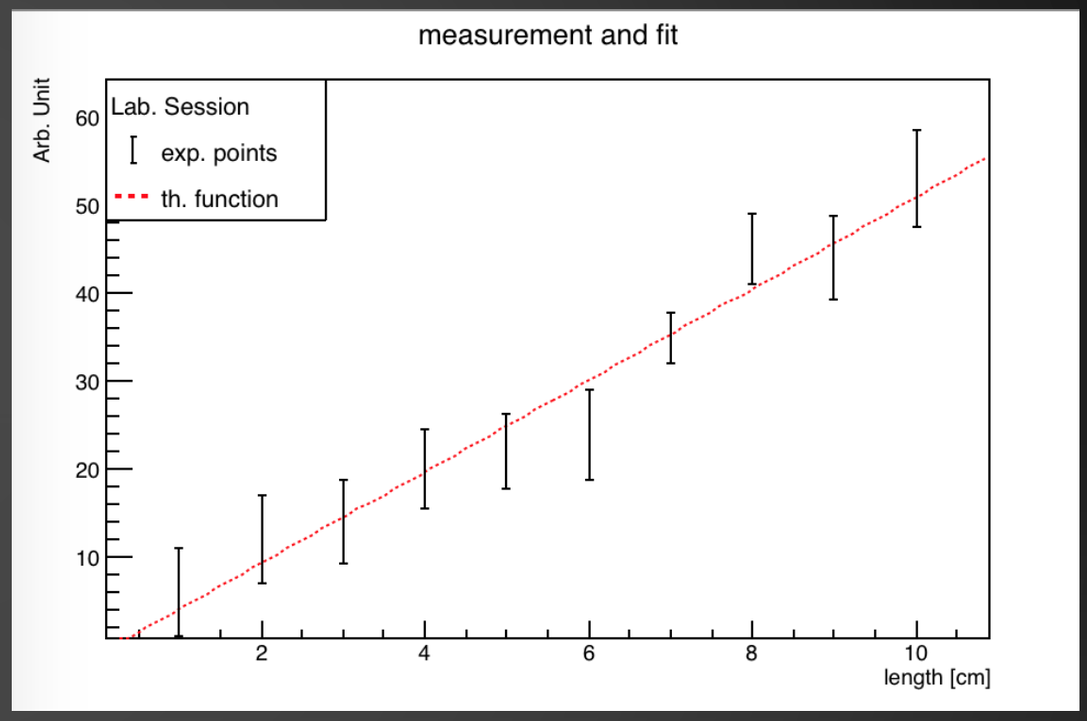


Ek Bilgi:

- Eksenlerin bölmelemesini değiştirmek için graph->GetXaxis()->SetNdivisions(20,5,0); kullanılabilir.
- Logaritmik eksen (y) yapmak için gPad->SetLogy(); yazılabilir.

ETİKET EKLEME

```
 GNU nano 2.0.6                          File: macro6.C

{
const int n_points = 10;
double x_val[n_points] = {1,2,3,4,5,6,7,8,9,10};
double y_val[n_points] = {6,12,14,20,22,24,35,45,44,53};
double y_err[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};


//Instance of the graph
auto graph = new TGraphErrors(n_points,x_val,y_val,nullptr,y_err);

graph->Draw("APE");
graph->SetTitle("measurement and fit");
graph->GetXaxis()->SetTitle("lenght [cm]");
graph->GetYaxis()->SetTitle("Arb. Unit");

//define a function
auto f = new TF1 ("Linear","[0]+x*[1]",.5,10.5);
f->SetLineColor(kRed);
f->SetLineStyle(2);
// Fit it to th graph

graph->Fit(f);
//

auto legend = new TLegend(.1,.7,.3,.9,"Lab. Session");
legend->AddEntry(graph, "exp. points","PE");
legend->AddEntry(f,"th.function","L");
legend->Draw();


}
```

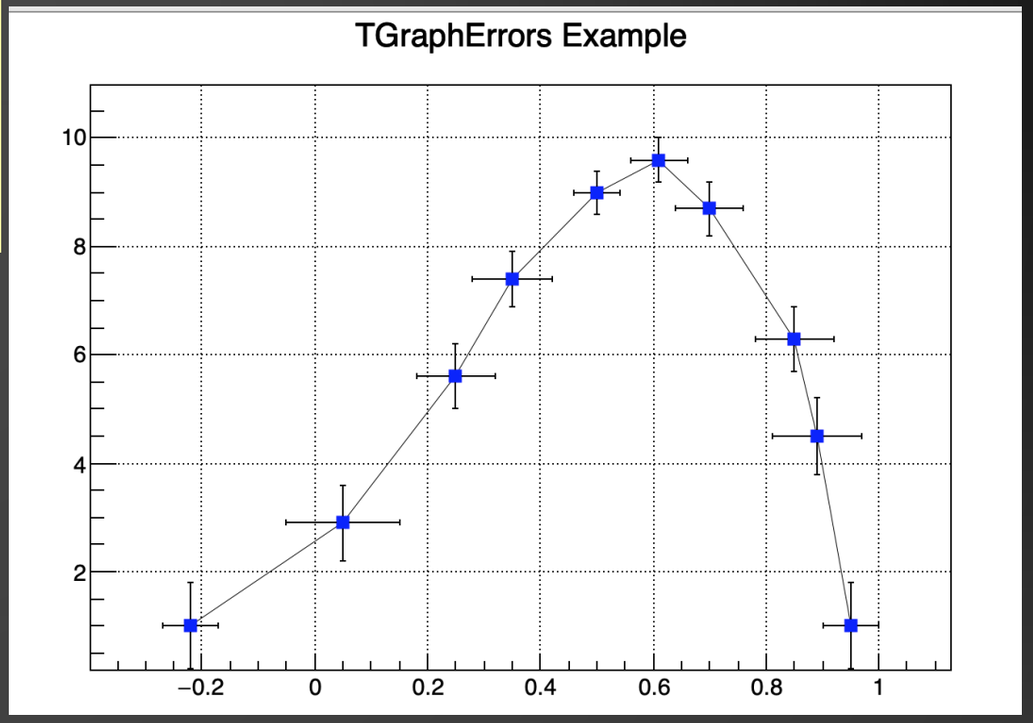


- TLegend yapıcısı (constructor) Legend'in konumunu ve onun ismini tanımlar.
- Her bir Legend elemanı AddEntry ile eklenir.
- "L" harfi ile çizgi tanımlaması, "PE" ile marker ve error bar gösterilir.

TGRAPH HATA GRAFİĞİ-1

errors.C

```
void gerrors() {
   TCanvas *c1 = new TCanvas("c1","A Simple Graph with error bars",200,10,700,500);

   c1->SetGrid();
   c1->GetFrame()->SetBorderSize(12);

   const Int_t n = 10;
   Float_t x[n]  = {-0.22, 0.05, 0.25, 0.35, 0.5, 0.61,0.7,0.85,0.89,0.95};
   Float_t y[n]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};

// Degisken errors
   Float_t ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
   Float_t ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};

   TGraphErrors *gr = new TGraphErrors(n,x,y,ex,ey);
   gr->SetTitle("TGraphErrors Example");
   gr->SetMarkerColor(4);
   gr->SetMarkerStyle(21);
   gr->Draw("ALP");
// ALP Axsix, line, point
   c1->Update();
}
```



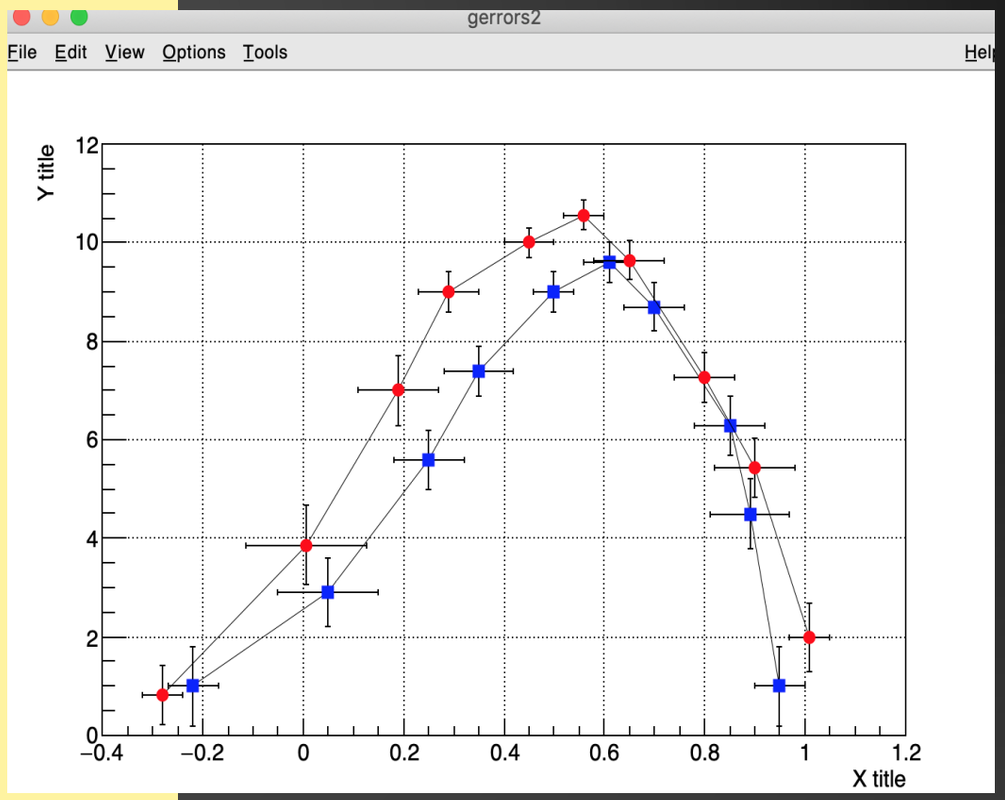
Her bir noktada farklı hatalar gösterilebilir.

Verilerin istatistik ve sistematik hatalar anlatılacaktır.

TGRAPH HATA GRAFİĞİ-2

gerrors2.C

```
 GNU nano 2.0.6                File: gerrors2.C

/// Draw two graphs with error bars

void gerrors2() {
   TCanvas *c1 = new TCanvas("c1","gerrors2",200,10,700,500);
   c1->SetGrid();

   // draw a frame to define the range
   TH1F *hr = c1->DrawFrame(-0.4,0,1.2,12);
   hr->SetXTitle("X title");
   hr->SetYTitle("Y title");
   c1->GetFrame()->SetBorderSize(12);

   // create first graph
   const Int_t n1 = 10;
   Double_t xval1[]  = {-0.22, 0.05, 0.25, 0.35, 0.5, 0.61,0.7,0.85,0.89,0.95};
   Double_t yval1[]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
   Double_t ex1[] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
   Double_t ey1[] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
   TGraphErrors *gr1 = new TGraphErrors(n1,xval1,yval1,ex1,ey1);
   gr1->SetMarkerColor(kBlue);
   gr1->SetMarkerStyle(21);
   gr1->Draw("LP");

   // create second graph
   const Int_t n2 = 10;
   Float_t xval2[]  = {-0.28, 0.005, 0.19, 0.29, 0.45, 0.56,0.65,0.80,0.90,1.01};
   Float_t yval2[]  = {0.82,3.86,7,9,10,10.55,9.64,7.26,5.42,2};
   Float_t ex2[] = {.04,.12,.08,.06,.05,.04,.07,.06,.08,.04};
   Float_t ey2[] = {.6,.8,.7,.4,.3,.3,.4,.5,.6,.7};
   TGraphErrors *gr2 = new TGraphErrors(n2,xval2,yval2,ex2,ey2);
   gr2->SetMarkerColor(kRed);
   gr2->SetMarkerStyle(20);
   gr2->Draw("LP");
}
```

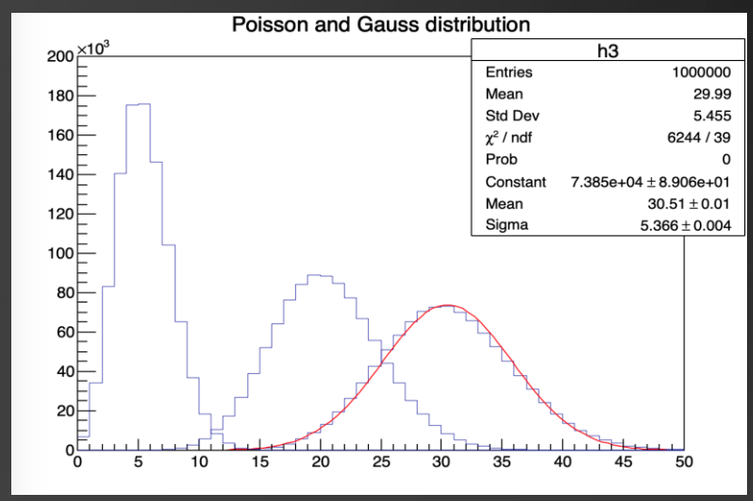


HİSTOGRAM FİT

```
  GNU nano 2.0.6                          File: macro7.C

{
gStyle->SetOptStat();
gStyle->SetOptFit(1111);

//
TH1F *h1 = new TH1F("h1","Poisson",50,0,50);
TH1F *h2 = new TH1F("h2","Poisson",50,0,50);
TH1F *h3 = new TH1F("h3","Poisson",50,0,50);


//Random Number Generator
TRandom3 RndGen;
Int_t N=1000000;
for(int i=0; i<N; i++){

h1->Fill(RndGen.Poisson(5));
h2->Fill(RndGen.Poisson(20));
h3->Fill(RndGen.Poisson(30));

}


h3->SetTitle("Poisson and Gauss distribution");
h3->SetMinimum(0);
h3->SetMaximum(200000);
h3->Draw();
h3->Fit("gaus");

h2->Draw("same");
h1->Draw("same");

}
```

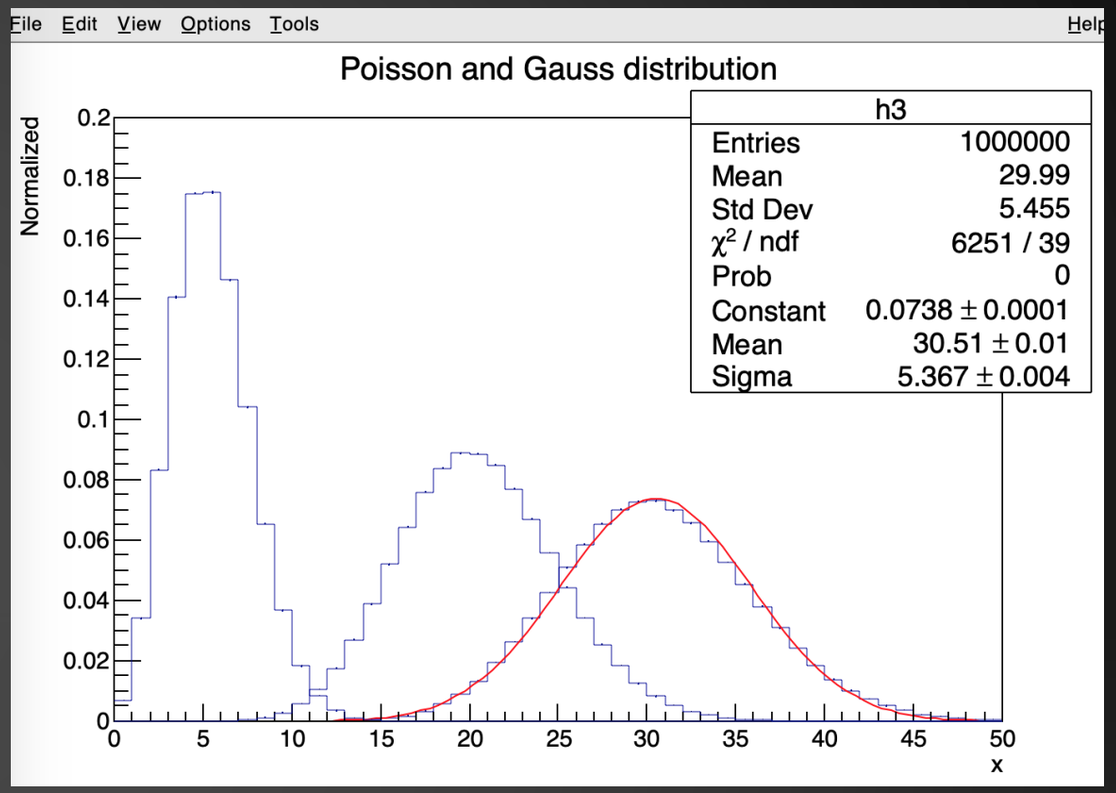
- Poisson Dağılımları:
x_{ort}=5, 20, 30

- Gauss fit:
x_{ort} ~30 ve Std ~5.4



HİSTOGRAM NORMALİZASYON

```
{
gStyle->SetOptStat();
gStyle->SetOptFit(1111);

//
TH1F *h1 = new TH1F("h1","Poisson",50,0,50);
TH1F *h2 = new TH1F("h2","Poisson",50,0,50);
TH1F *h3 = new TH1F("h3","Poisson",50,0,50);


//Random Number Generator
TRandom3 RndGen;
Int_t N=1000000;
Float_t w=1.0/(float)N;
for(int i=0; i<N; i++){

h1->Fill(RndGen.Poisson(5),w);
h2->Fill(RndGen.Poisson(20),w);
h3->Fill(RndGen.Poisson(30),w);

}

//
h3->SetTitle("Poisson and Gauss distribution");
h3->GetXaxis()->SetTitle("x");
h3->GetYaxis()->SetTitle("Normalized");

h3->SetMinimum(0);
h3->SetMaximum(0.2);
h3->Draw("H");
h3->Fit("gaus");

h2->Draw("sameH");
h1->Draw("sameH");

}
```



Normalize histogram:

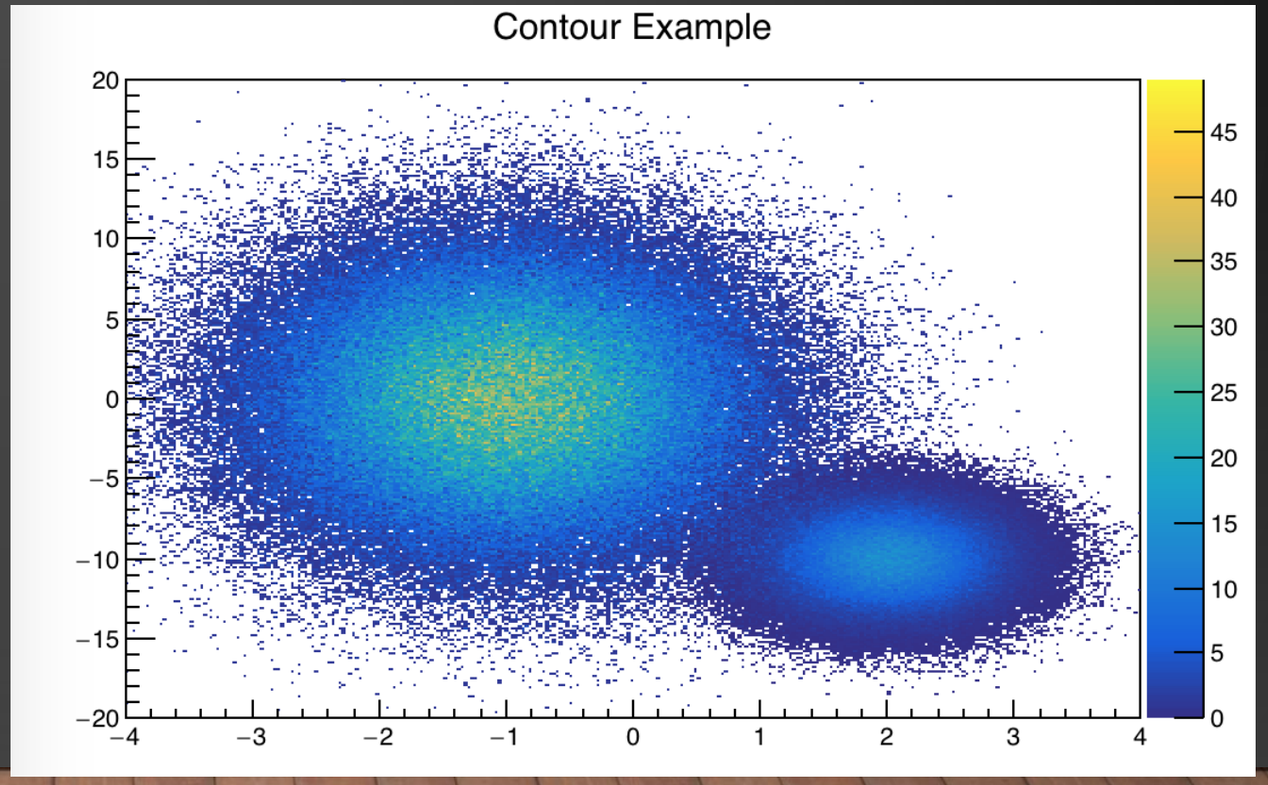
- y-ekseni ağırlık faktörü (w) ile normalize edildi
- x- eksen ismi ve y-eksen ismi yazıldı
- Draw komutunda “H” kullanılarak histogram çizildi.

İKİ BOYUTLU GRAFİK

```
 GNU nano 2.0.6                                  File: macro9.C

void macro9(){
TH2F *h =new TH2F("h","Contour Examples", 300, -4, 4,300,-20,20);
h->SetStats(0);
h->SetContour(200);

float px, py;
for (int i = 0; i < 250000; i++){

gRandom->Rannor(px,py);
h->Fill(px-1,5*py);
h->Fill(2+0.5*px,2*py-10.,0.1);
}
h->Draw("colz");
}
```

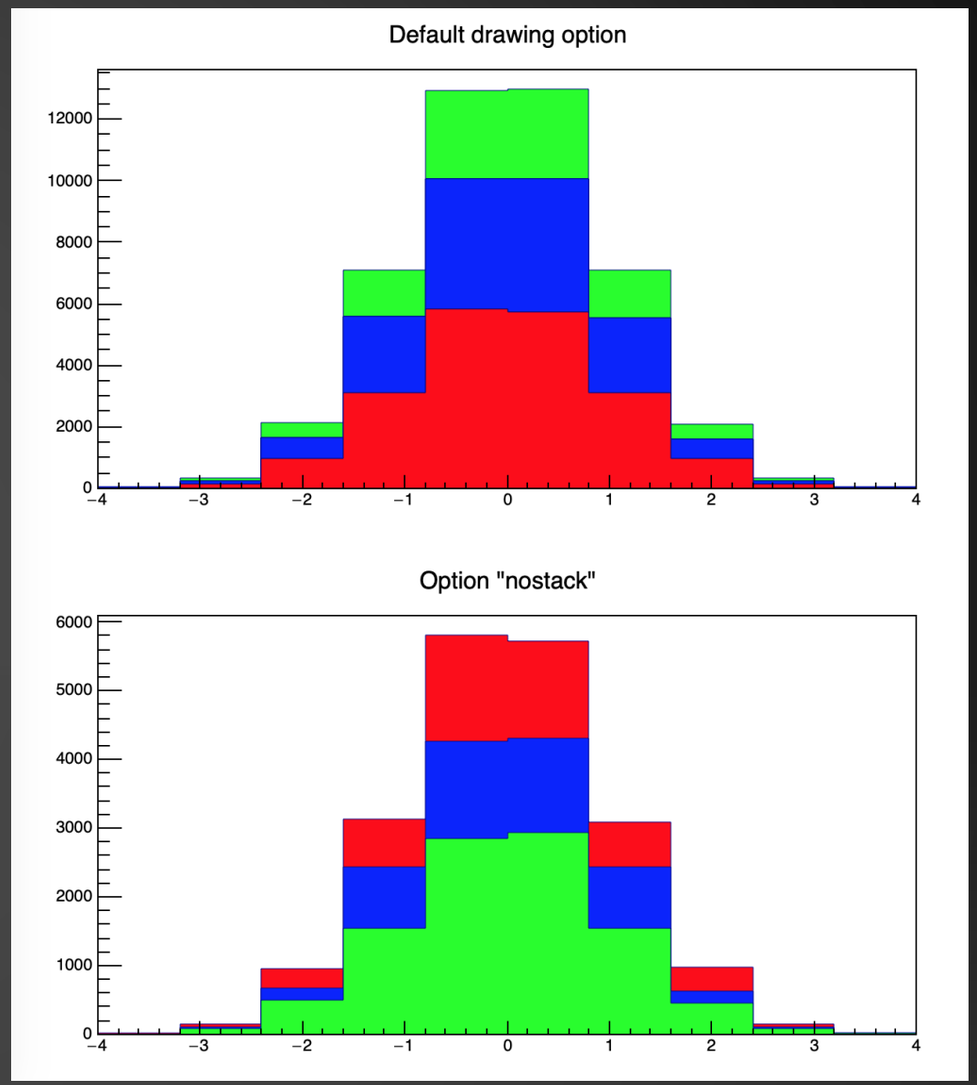
Bu makro iki boyutlu Gaussian grafiği oluşturur.

Rastgele sayı (normal) üretimi için döngü değişkeni daha büyük bir sayı Örneğin, 25M seçip grafik elde edilebilir.



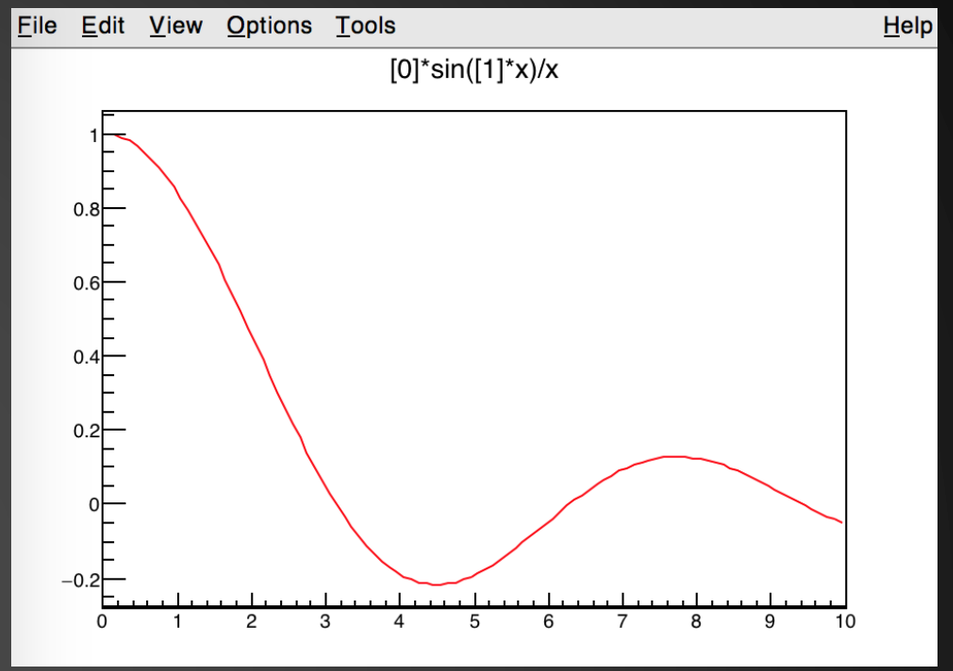
YIĞIN (STACK) GRAFİĞİ

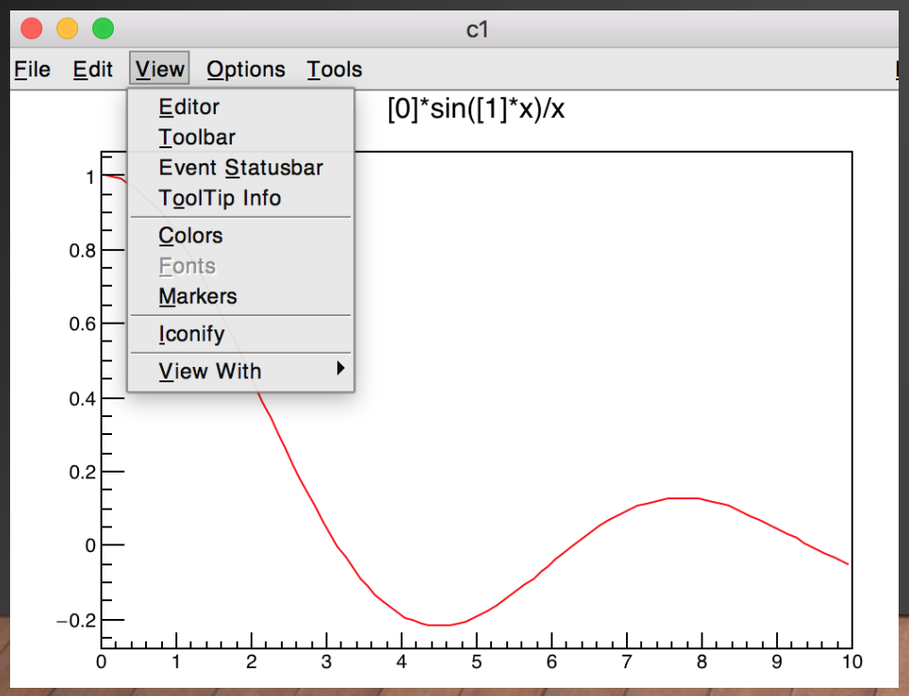
```
 GNU nano 2.0.6                          File: stack.C

{
   THStack *hs = new THStack("hs","");
   TH1F *h1 = new TH1F("h1","test hstack",10,-4,4);
   h1->FillRandom("gaus",20000);
   h1->SetFillColor(kRed);
   hs->Add(h1);
   TH1F *h2 = new TH1F("h2","test hstack",10,-4,4);
   h2->FillRandom("gaus",15000);
   h2->SetFillColor(kBlue);
   hs->Add(h2);
   TH1F *h3 = new TH1F("h3","test hstack",10,-4,4);
   h3->FillRandom("gaus",10000);
   h3->SetFillColor(kGreen);
   hs->Add(h3);
   TCanvas *cs = new TCanvas("cs","cs",10,10,700,900);
   TText T; T.SetTextFont(42); T.SetTextAlign(21);


   cs->Divide(1,2);
   cs->cd(1); hs->Draw(); T.DrawTextNDC(.5,.95,"Default drawing option");
   cs->cd(2); hs->Draw("nostack"); T.DrawTextNDC(.5,.95,"Option \"nostack\"");
//   cs->cd(3); hs->Draw("nostackb"); T.DrawTextNDC(.5,.95,"Option \"nostackb\"");
//   cs->cd(4); hs->Draw("lego1"); T.DrawTextNDC(.5,.95,"Option \"lego1\"");
   return cs;
}
```

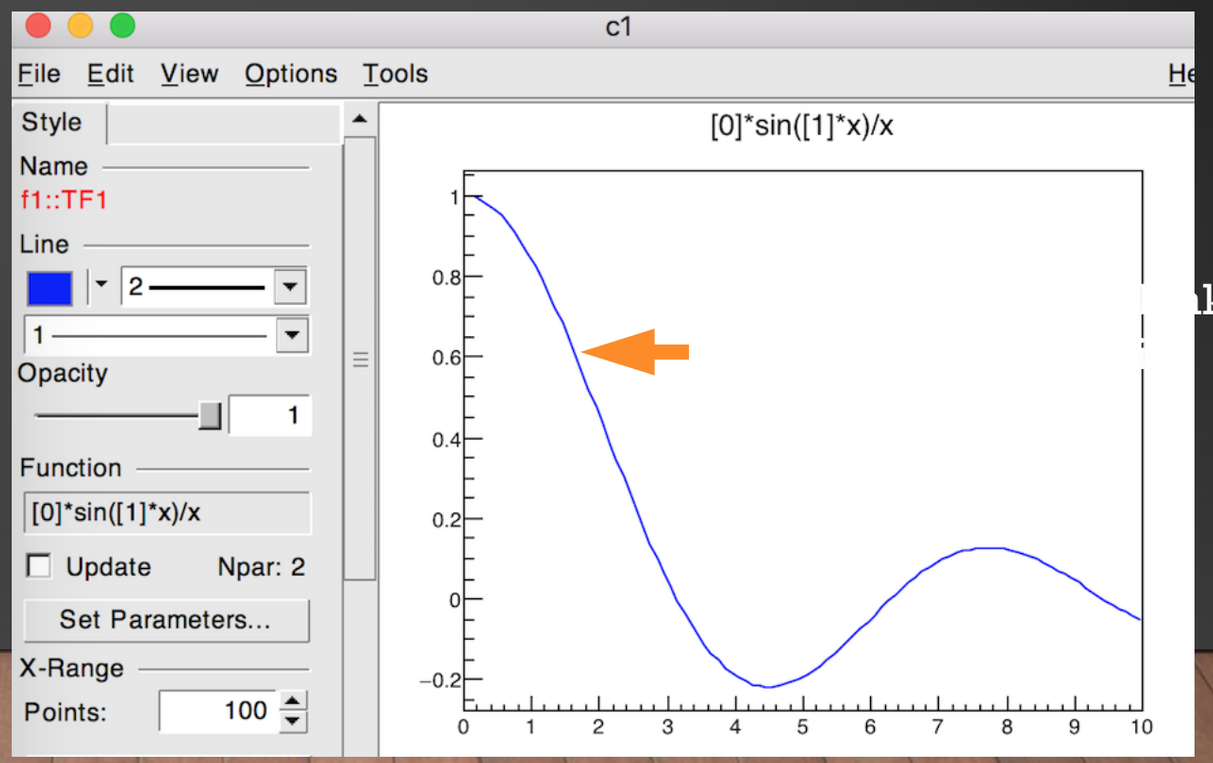


ROOT GUI MOD

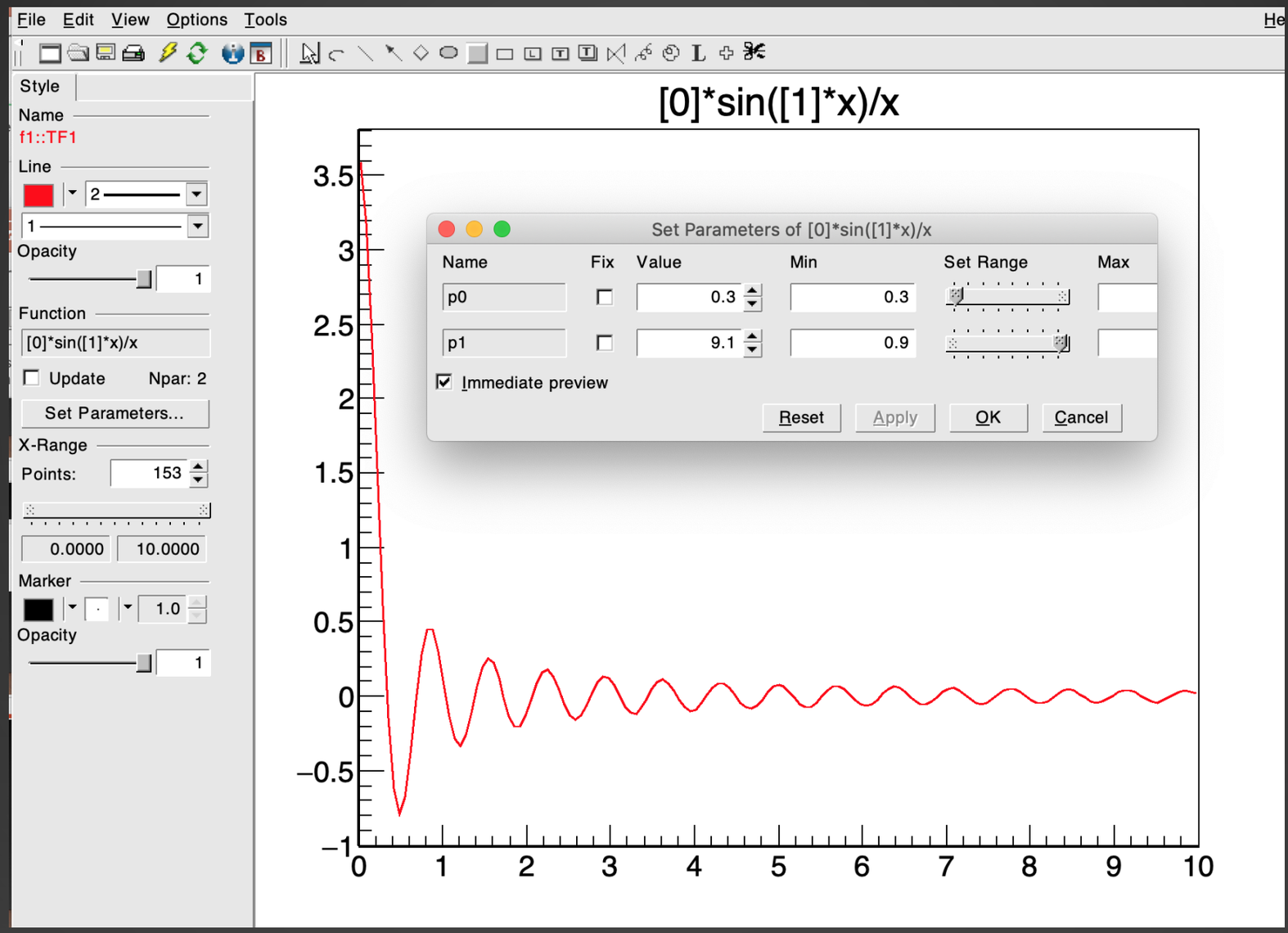
```
 GNU nano 2.0.6                File: gui_1.C

{
TF1 f1("f1","[0]*sin([1]*x)/x",0.,10.);
f1.SetParameters(1.,1.);
f1.Draw();
}
```







GUI'DEN FONKSİYONA PARAMETRE AKTARMA



View menüsünden Toolbar açıp ok simgesinin üzerine tıklayarak istenen yere çizilir.

VERİ DOSYASINDAN (.TXT VEYA .DAT) OKUMA VE HİSTOGRAM ÇİZME

```
 GNU nano 2.0.6                          File: file1.C

{
TH1F h("h","count rate",100,0.,5.);
ifstream inp; double x;
inp.open("expo.dat");

while (inp >> x) {h.Fill(x);}
h.Draw();
inp.close();
}
```

Veri dosyası:

- expo.dat

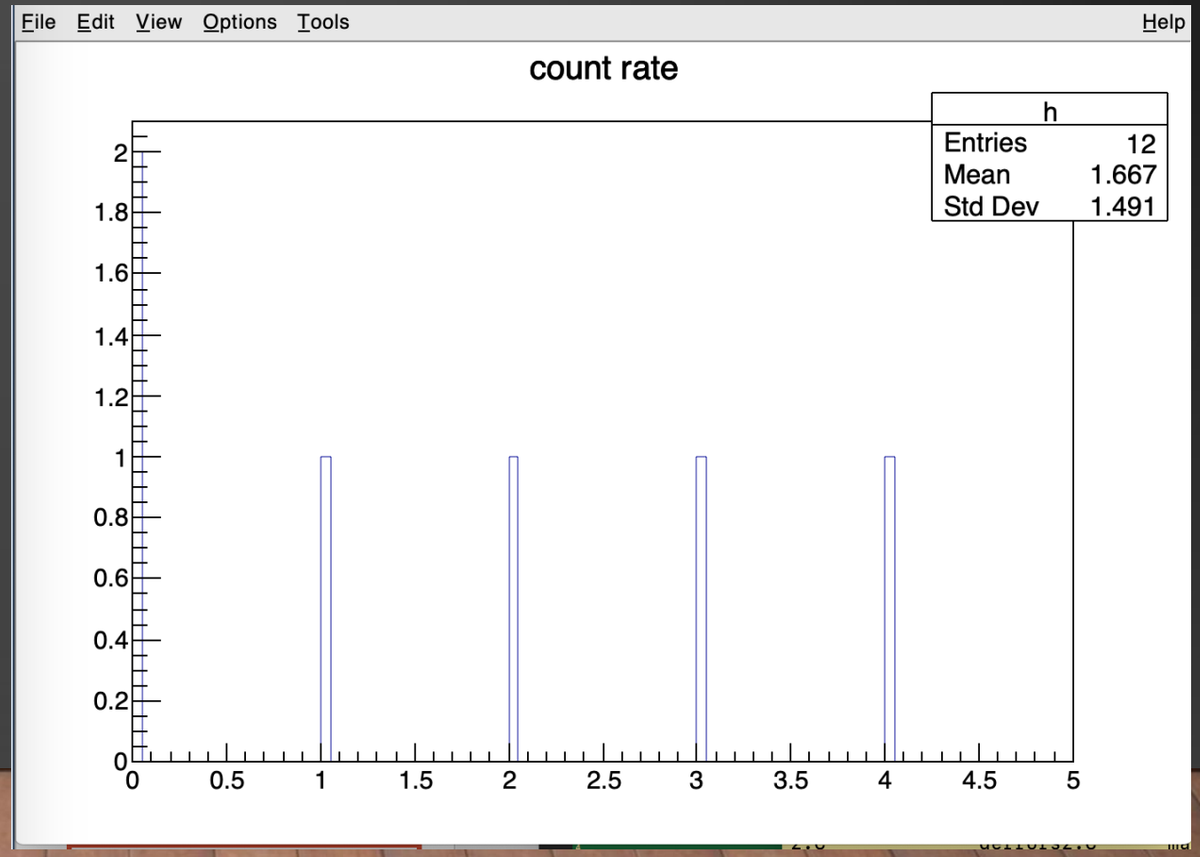
Makro dosyası:

- file1.C

Önce histogram tanımlaması yapılır, dosyadan veriler okunur. Bu veriler ile histogram doldurulur (bir çeşit sayma deneyi), sonuç gösterilir.

```
 GNU nano 2.0.6                          File: expo.dat

0
2
3
4
5
5
6
0
6
5
7
1
```

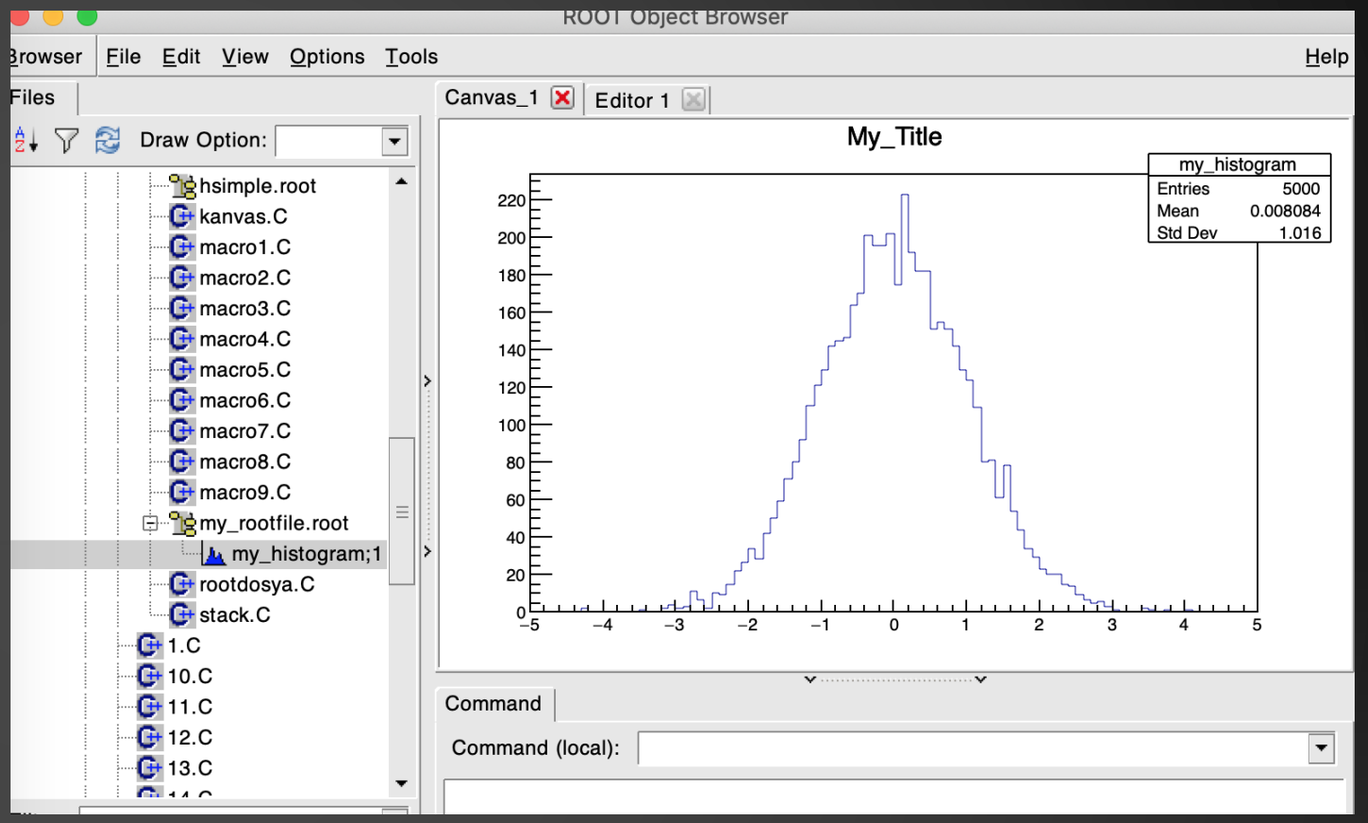
Inp: Bir nesne olarak tanımlanır ve inp.open ile dosya açılır.



ROOT DOSYALARININ (.ROOT) YAZILMASI

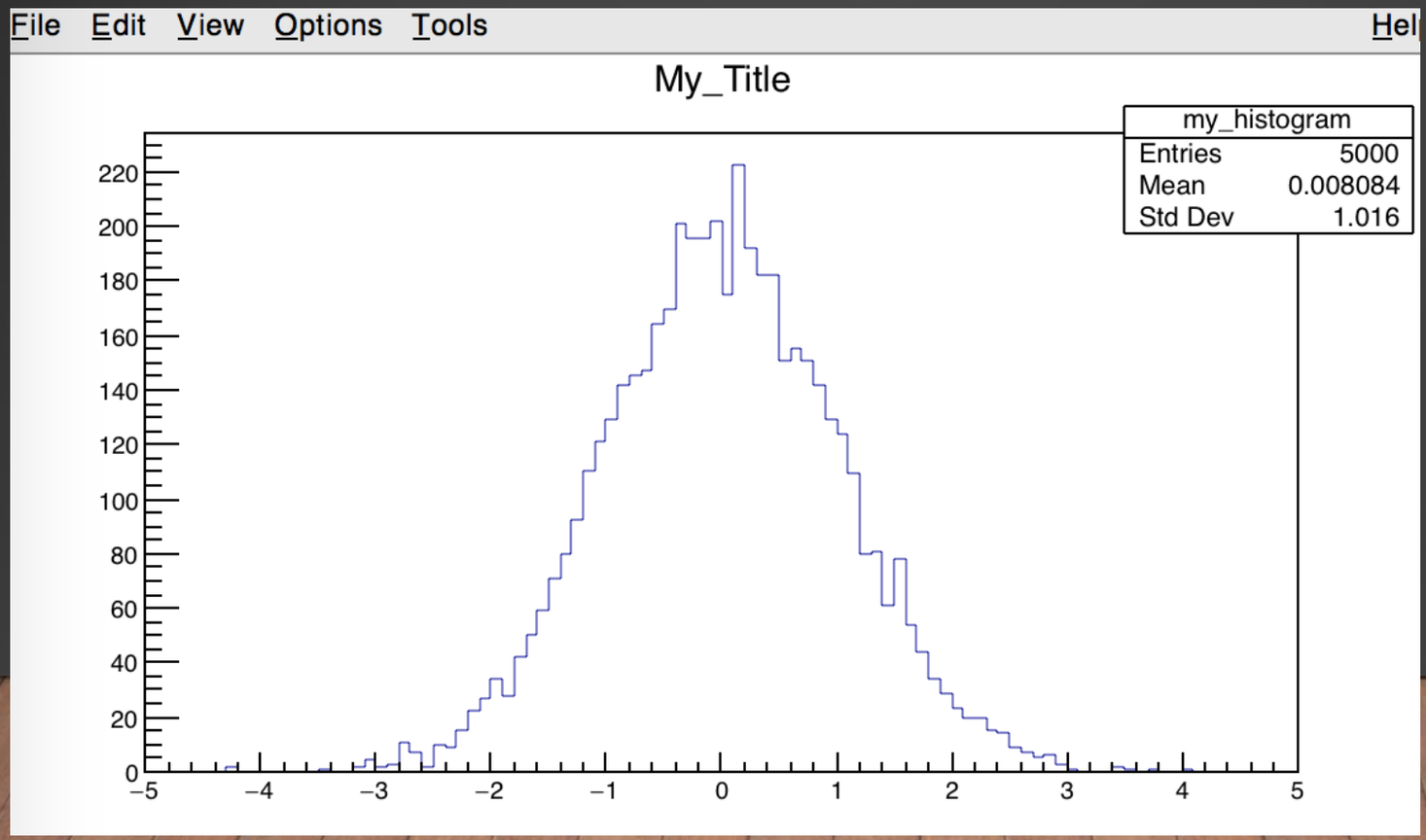
```
 GNU nano 2.0.6                    File: rootdosya.C

void write_to_file(){

//histogram

TH1F h("my_histogram","My_Title",100,-5,5);

// Histogramı rastgele doldur

h.FillRandom("gaus");

// TFile ac
TFile out_file("my_rootfile.root","RECREATE");

// Bir dosyaya histogram yazdır
h.Write();

//Dosyayı kapat

out_file.Close();

}
```

```
^[[AILKAYs-MacBook-Pro:PHBUO ilkayturkcakir$ root -l
root [0] .L rootdosya.C
root [1] write_to_file()
root [2] TBrowser w
```



KOMUT SATIRINDANDA ROOT DOSYASINDAKİİ HİSTOGRAMI ÇİZDİREBİLİRİZ (TBROWSER KULLANMADAN)

```
ILKAYs-MacBook-Pro:PHBUO ilkayturkcakir$ root -l my_rootfile.root
root [0]
Attaching file my_rootfile.root as _file0...
(TFile *) 0x7f847f86b6a0
root [1] _file0->ls()
TFile**         my_rootfile.root
 TFile*         my_rootfile.root
  KEY: TH1F     my_histogram;1  My_Title
root [2] my_histogram->Draw()
```



EK ÖRNEK

SATIR VEYA SÜTUN ŞEKLİNDEKI BİLGİYİ OKUMA VE YAZMA (N-TUPLE)

```
  GNU nano 2.0.6                          File: ntup1.C

// Fill n-tuple and write it to a file simulating measurement of
// conductivity of a material in different condition of pressure and temperature

void write_ntuple_to_file(){
TFile ofile("conductivity_experiment.root","RECREATE");

//Initialise the NTuple
TNtuple cond_data("cond_data","Example N-Tuple","Potantial:Current:Temperature:Pressure");

//Fill it randomly to fake the acquires data
TRandom3 rndm;
float pot,cur,temp,pres;
for (int i=0; i<10000; ++i) {

pot=rndm.Uniform(0.,10.); // get voltage
temp=rndm.Uniform(250.,350.); // get temperature
pres=rndm.Uniform(0.5,1.5); // get pressure
cur=pot/(10.+0.05*(temp-300.)-0.2*(pres-1.)); // current
// add some random smearing (meusurement error)

pot*=rndm.Gaus(1.,0.01); // 1% error on voltage
temp+=rndm.Gaus(0.,0.3); // 0.3 abs. error on temperature
pres*=rndm.Gaus(1.,0.02); // 1% error on pressure
cur*=rndm.Gaus(1.,0.01); // 1% error on current

// write to ntuple

cond_data.Fill(pot,cur,temp,pres);
}

// save teh ntuple and close the file

cond_data.Write();
ofile.Close();
}
```

```
ILKAYs-MacBook-Pro:PHBUO ilkayturkcakir$ root -l
root [0] .L ntup1.C
root [1] write_ntuple_to_file()
root [2] TBrowser f
```

N-tuple şeklinde yazmanın faydaları:

- Optimize disk girdi ve çıktısı sağlar

Bir çok N-tuple satırı saklama imkanı tanır.

- ROOT dosyalarının içine N-tuple yazılabilir. Sadece sayıları değil nesneleri de sütunlar şeklinde depolayabilir. Örnek: N-tuple'i yazan örnek bir Makro oluşturalım.

Elde edilen grafikler