```
Double_t effErf(double* x, double* p) {
//p[0]==media
//p[1]==sigma
//p[2]==valore di saturazione
//offset 1 e fattore 1/2 per riscalare Erf in [0,1]

  return (TMath::Erf(  ( x[0] - p[0] ) / p[1] ) +1) / 2. * p[2];
}

void efficiency(Int_t nGen=1E6){
  gRandom->SetSeed();
  cout <<"random generation seed: " <<gRandom->GetSeed() <<endl;
  TH1F *h[2];
  TString histName="h";
  TString title[2]={"generated distribution","observed distribution"};
  for(Int_t i=0;i<2;i++){

    h[i] =new TH1F(histName+i,title[i],100,0,10);
//cosmetics
    h[i]->SetLineColor(1);
    h[i]->GetYaxis()->SetTitleOffset(1.2);
    h[i]->GetXaxis()->SetTitleSize(0.04);
    h[i]->GetYaxis()->SetTitleSize(0.04);
    h[i]->GetXaxis()->SetTitle("x");
    h[i]->GetYaxis()->SetTitle("Entries");
    h[i]->Sumw2();//Important
  }

  h[0]->SetFillColor(4);
  h[1]->SetFillColor(2);

//The efficiency profile
  TF1* myErf = new TF1("myErf", effErf, 0, 10., 3);
  myErf->SetParameter(0, 5.);//flexus
  myErf->SetParameter(1, 0.5);//width
  myErf->SetParameter(2, 0.7);//saturation value
  TCanvas *cFunc = new TCanvas("cFunc","Efficiency Profile",200,10,600,400);
  myErf->SetLineColor(1);
  myErf->SetMaximum(1);
  myErf->Draw();


//case: uniform distribution
  Double_t x=0,xRNDM=0;

  for(Int_t i=0;i<nGen;i++){
   //x=gRandom->Uniform(0,10);
     x=gRandom->Exp(1); //negative exponential with mu=1
     h[0]->Fill(x); //generated
     xRNDM=gRandom->Rndm();
     if( xRNDM<myErf->Eval(x))h[1]->Fill(x); //observed
  }

  TCanvas *cHisto = new TCanvas("cHisto","Efficiency effects, Generated and Observed
",200,10,600,400);
```

```
  cHisto->Divide(1,2);
  for(Int_t i=0;i<2;i++){
  cHisto->cd(i+1);
  h[i]->SetMinimum(0);
  h[i]->Draw("H");
  h[i]->Draw("E,SAME");
  }

  TH1F *hEff=new TH1F(*h[0]);
  hEff->SetTitle("0bserved Efficiency");
  hEff->SetName("hEff");
  hEff->SetFillColor(3);
  hEff->Divide(h[1],h[0],1,1,"B");
  TCanvas *cEff = new TCanvas("cEff","Observed Efficiency",200,10,600,400);
  hEff->SetMaximum(1.);
  hEff->Draw("H");
  hEff->Draw("E,SAME");
  cout << "Integral efficiency =" <<h[1]->Integral()/h[0]->Integral()<<endl;

}
```