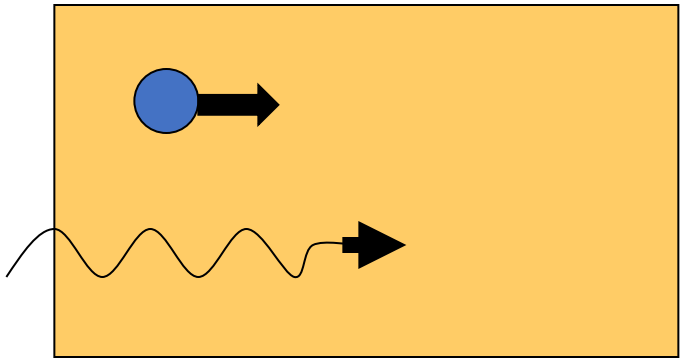
Monte Carlo ゼミ

2019/10/17

相互作用の確率過程　1

- 素粒子・原子核の相互作用を考えたとき、非常に大くの確率過程が存在します。
- 例えば粒子が物質中を飛んでいるとしましょう。



荷電粒子だったら微小距離Δx通ったとき
- 崩壊する？崩壊するならどのモードに出る？
- エネルギー損失はどれくらいする？
- 多重散乱で進行方向はどうなる？
- 反応を起こす？

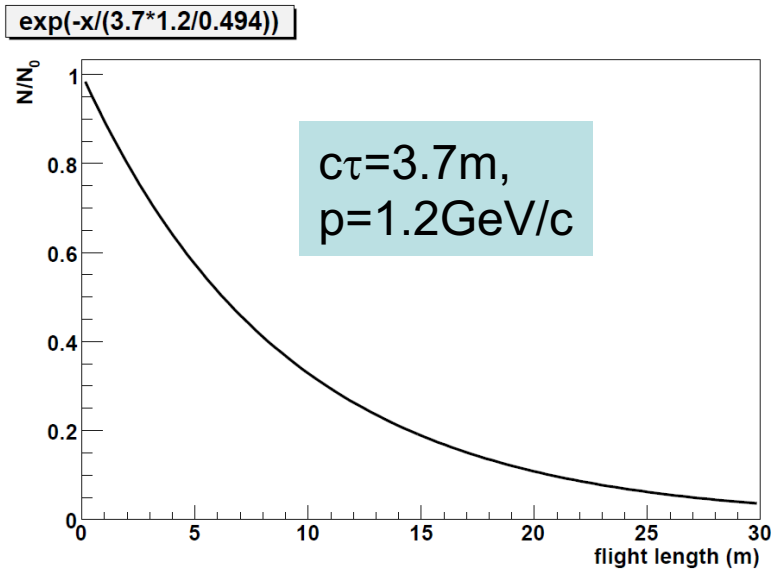
例えば光だったら
- 相互作用をする？
- 相互作用をするならどれを起こす？
  - 光電効果？
  - コンプトン散乱？
  - 電子対生成？
- コンプトンなら電子はどの角度に出る？

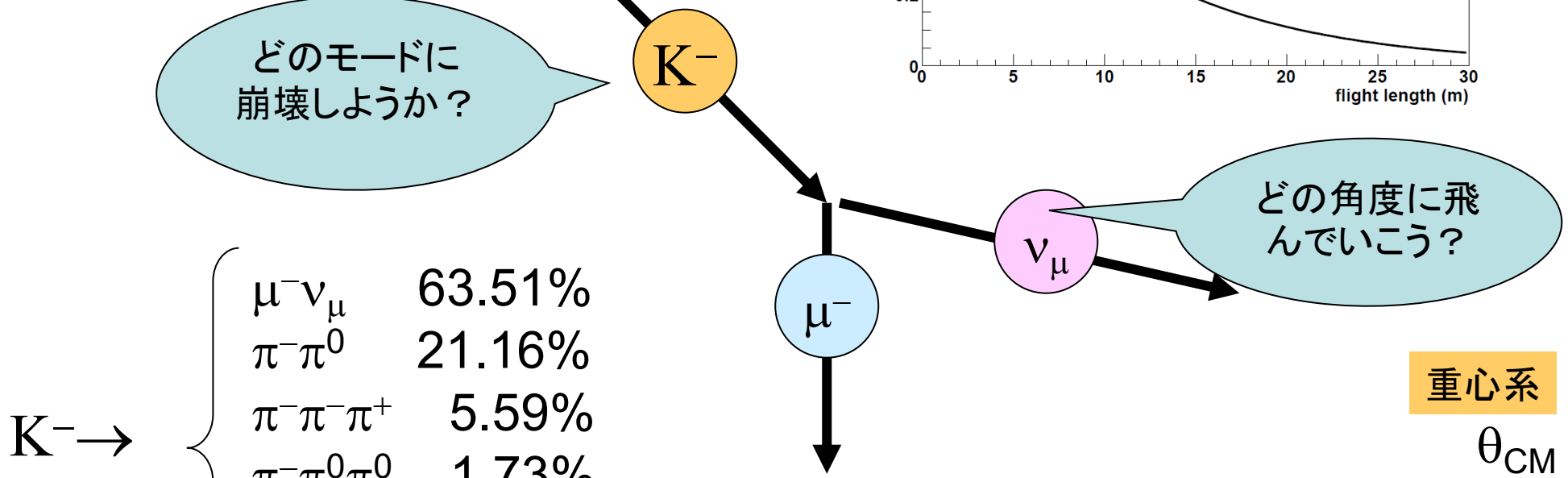
相互作用の確率過程　2

どれくらい
飛んだら
崩壊しよう？

どのモードに
崩壊しようか？

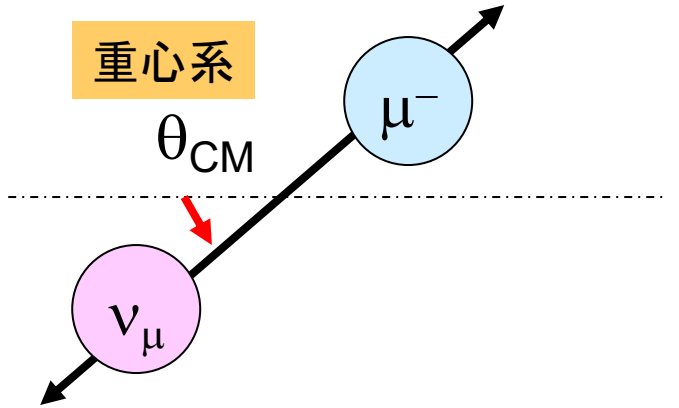
どの角度に飛
んでいこう？



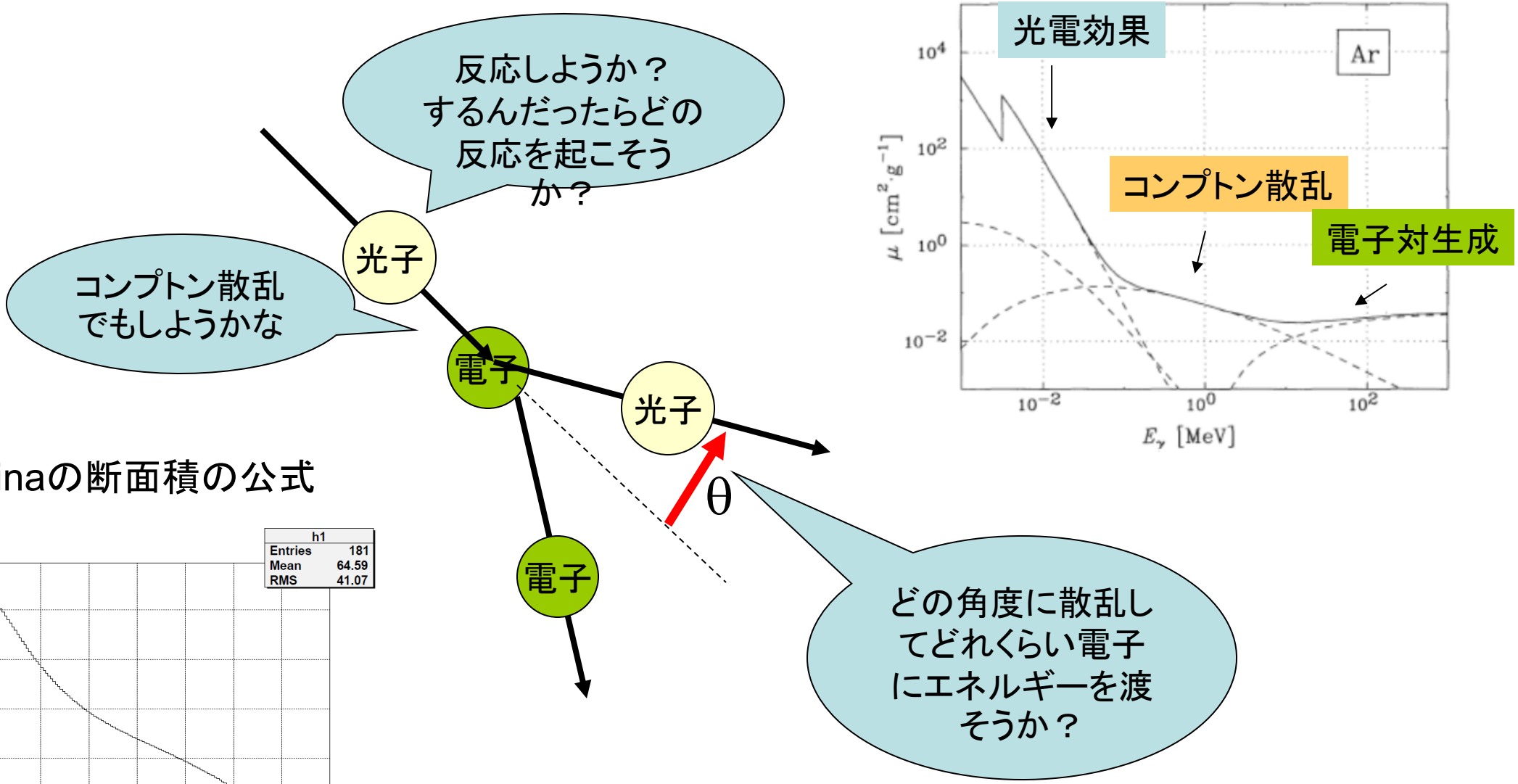


$K^- \rightarrow$

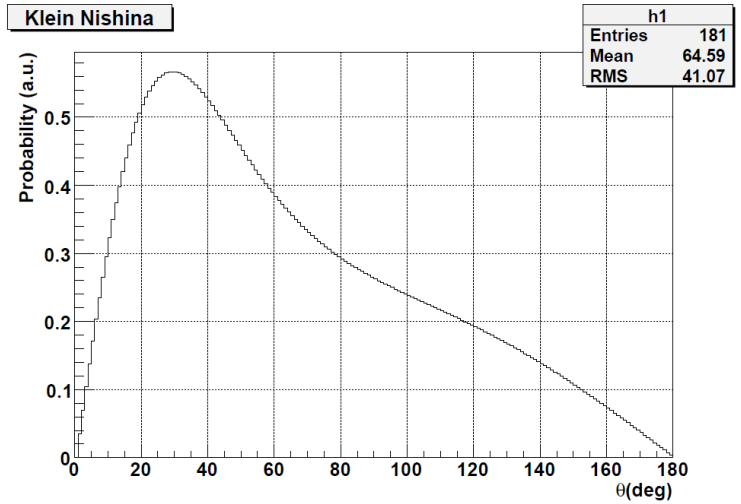
$\mu^-\nu_\mu$	63.51%
$\pi^-\pi^0$	21.16%
$\pi^-\pi^-\pi^+$	5.59%
$\pi^-\pi^0\pi^0$	1.73%
$\pi^0\mu^-\nu_\mu$	3.18%
$\pi^0 e^-\nu_e$	4.82%



相互作用の確率過程　3



Klein-Nishinaの断面積の公式



物理プロセスとMonte Carlo simulation

- こうして見てきたように、様々な物理プロセスは確率分布に従って発生します。
- このような現象を記述するプロセスが良く知られていたとしても全体を直接計算することは難しい
- このようなときに威力を発揮するのがモンテカルロ法である。モンテカルロ法は個々のプロセスの起こる確率、あるいは出現確率の比によって乱数を発生させ、多数回の試行を行うことによって、実際の現象をシミュレートするものである。

乱数の発生

- 様々な乱数発生の関数が用意されています
  - stdlib.h c言語の標準関数にも
  - int rand()　→ 0~RAND_MAXまでの整数値を一様に生成
  - (double)rand()/((double)RAND_MAX+1.0)で0から1まで一様に発生することができます。
- 僕らがよく使うライブラリーとしては
  - ROOT → TRandom classのUniform(double x1, x2)
    - http://root.cern.ch/root/html/TRandom.html 参照
  - GEANT4, CLHEP　→ RandFlat class
    - RandFlat::shoot()
    - http://lcgapp.cern.ch/doxygen/CLHEP/CLHEP_2_0_2_2/doxygen/html/class_c_l_h_e_p_1_1_hep_random.html 参照

一様な乱数の確認

- ROOTという解析ツールの中に入っている乱数を発生させるクラスを用いる

私のLinux PC (dhcp2576, IP 172.25.25.118)にログインして、まずテストしましょう。

> ssh –X sks@dhcp2576　(sksというアカウント名でログインする)

- cd user
- mkdir hoge(各自のディレクトリ名)　(各自のwork directoryを作成)
- cd hoge　(先ほどのdirectoryに移動)
- mkdir MonteCarlo　(このdirectoryで、MonteCarloというdirectoryを作成)
- cd MonteCarlo　(作成したdirectoryに移動)
- cp ../../miwa/MonteCarlo/test1.C .　(予め用意したrootのマクロファイルをコピー)
- root　(rootというプログラムを立ち上げる)
- [root] .x test1.C　(rootのプログラムの中で、先ほどのマクロファイルを実行する)

一様な乱数の確認

マクロのファイル名test1.Cに対応する関数名

```
void test1()
{
  TCanvas *c1 = new TCanvas("c1","c1");

  TH1F *h1 = new TH1F("h1","test", 100, 0, 1);

  gRandom->SetSeed(time(NULL));

  int Nevent=1000000;

  for (int i=0; i<Nevent; i++) {
    double val = gRandom->Uniform(0, 1);
    h1->Fill(val);
  }

  h1->SetMinimum(0);
  h1->Draw();

}
```

キャンバスを作る

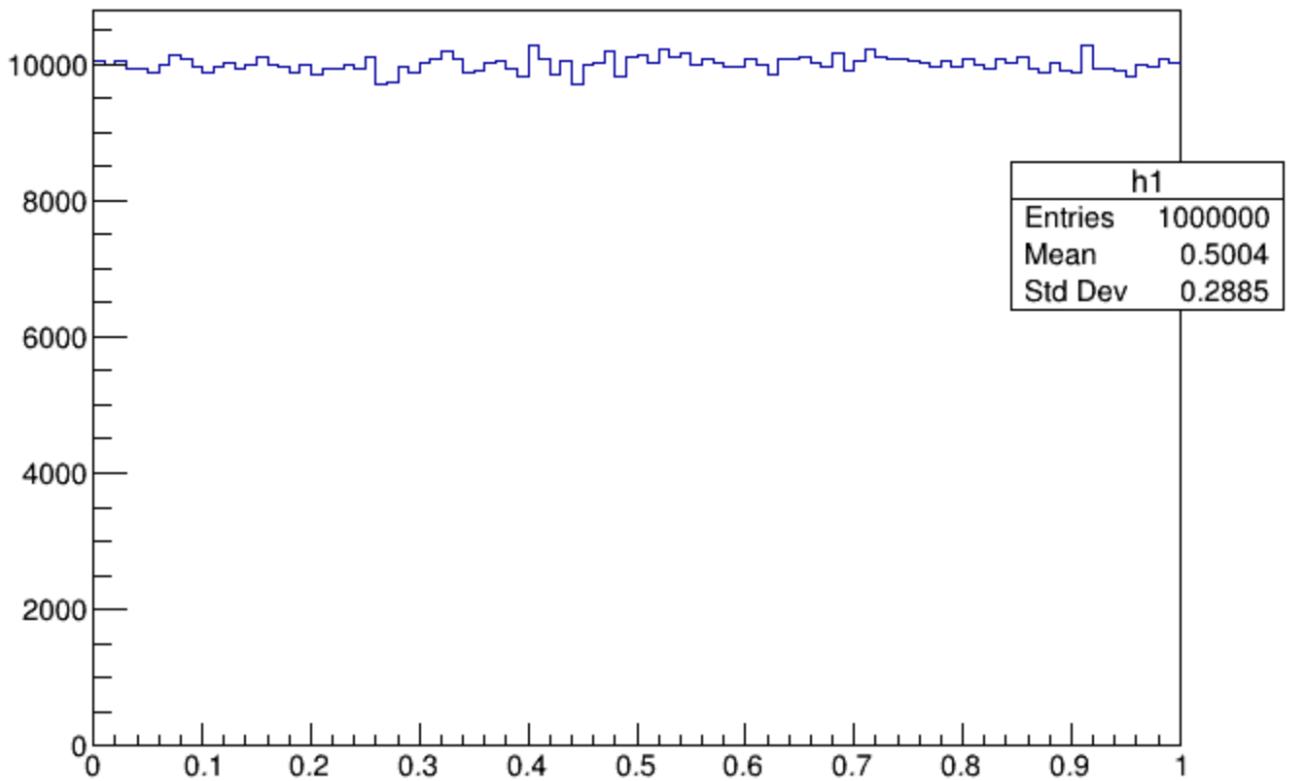
0から1の間を100個の区切り(bin)で分けた
h1という名前の１次元のヒストグラムを作る

乱数の種を設定

0から1までの間で一様に乱数を発生

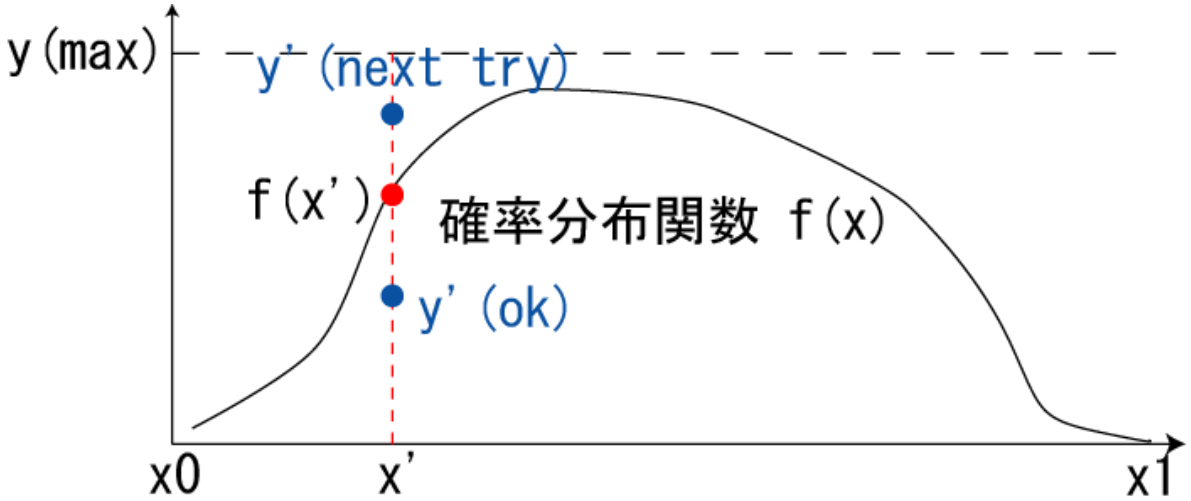
その値をヒストグラム(h1)につめる

ヒストグラムを表示する



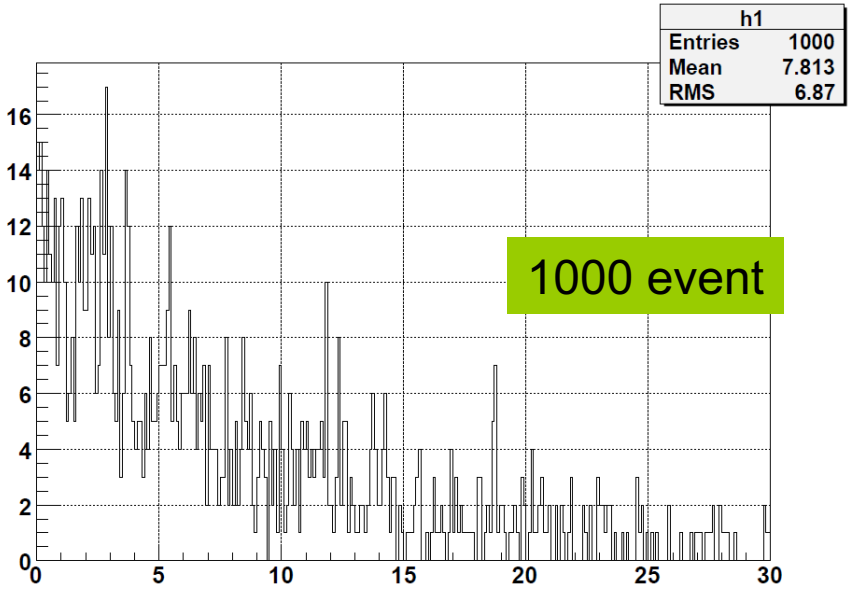
分布に従う乱数の発生

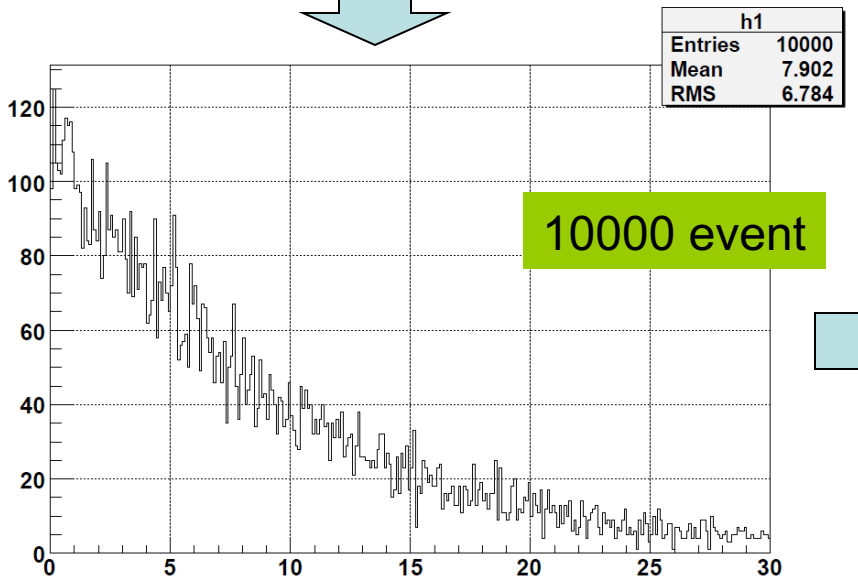
- 一様乱数から確率分布にしたがう乱数へ

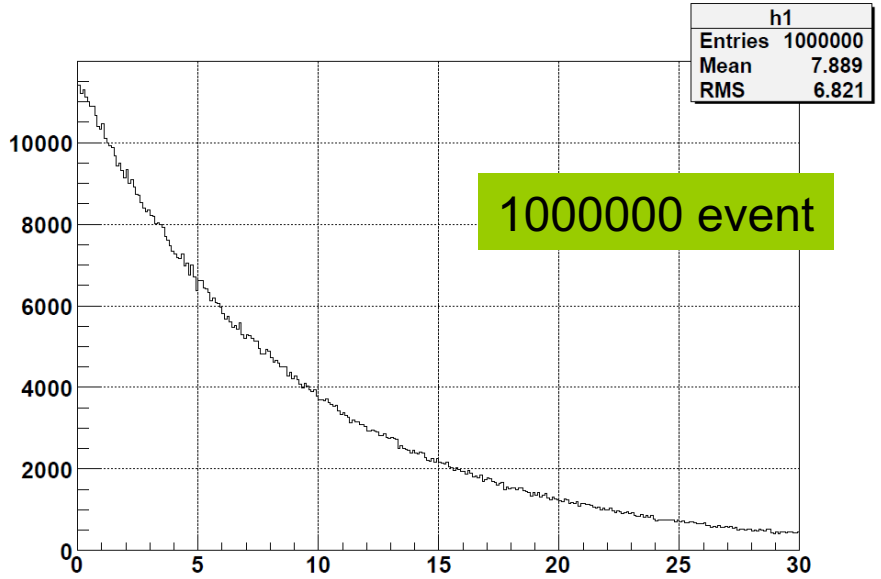


1. まずx0~x1までの間で一様な乱数である値x'を決める。
2. 0~y(max)まで一様な乱数である値y'を決める。
3. y'<f(x')ならこのx'を返してあげればOK。y'>f(x')ならばまた1.から繰り返す。

例えばexpの形の分布だったら





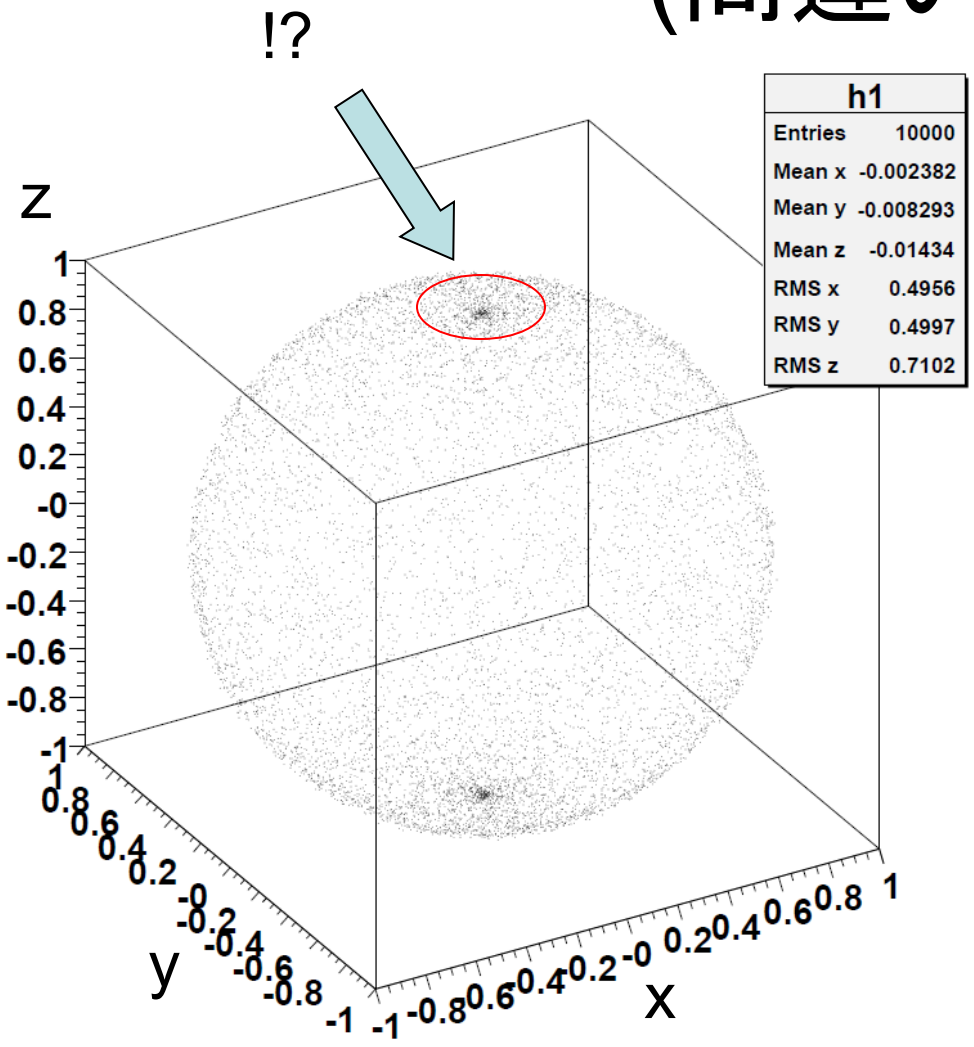


- 試行の数を増やしていくと滑らかな分布になっていく
- 作り方の参考に

練習として

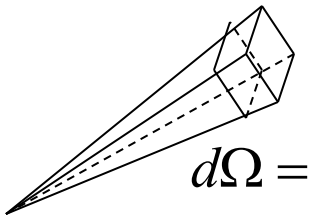
- -1<x<1の間でf(x)=1/2(1+x)の分布に従う乱数を発生させてヒストグラムにつめる。
- 三次元空間に一様に発生させる (θ, ϕ)

三次元空間に等方的に生成
(間違いの例)

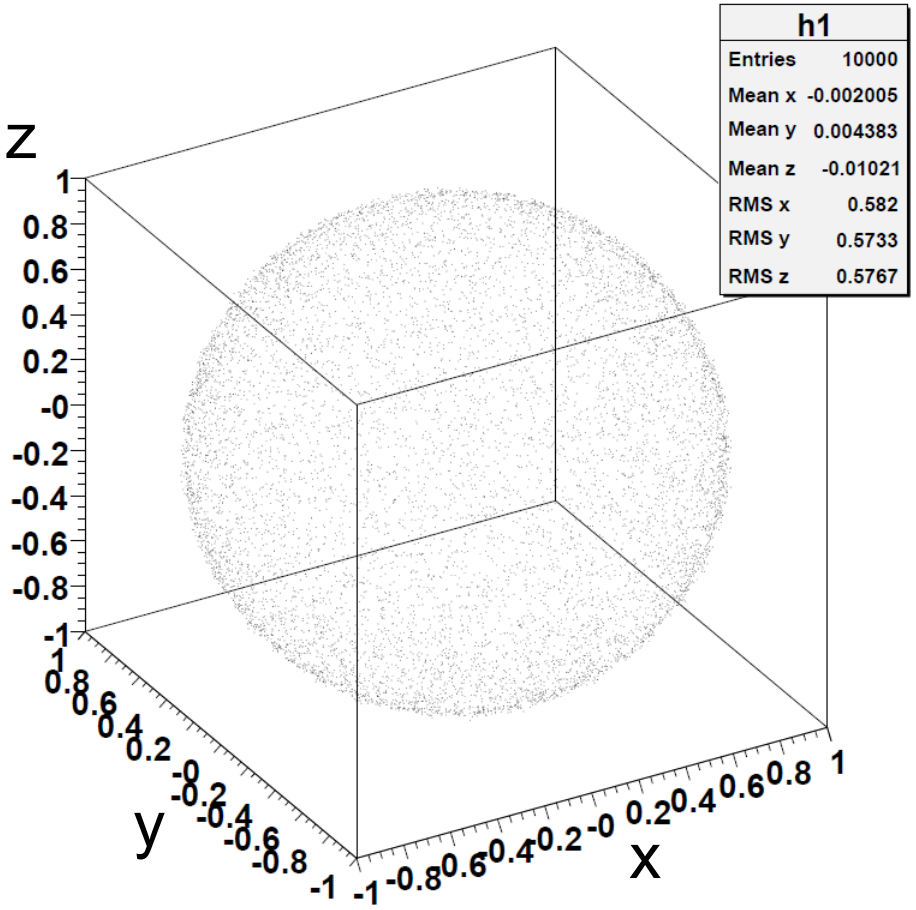


- (θ,ϕ)をどちらも$0<\theta<\pi$, $0<\phi<2\pi$までランダムに振って
- $(x, y, z)=(\sin\theta\cos\phi,$
- $\sin\theta\sin\phi,$
- $\cos\theta)$
- を分布させると、、、、
- $\theta=0, \pi$のところに集中している点がある。
- なぜか？

三次元空間に等方的に生成



$$d\Omega = d\cos\theta d\phi$$
$$= \sin\theta d\theta d\phi$$



- この場合は立体角dΩを等方に一様に振ることを考える。
- dΩ=sinθdθdϕ
- なので、θを振るときにはsinθの重みをかけて振らないといけない。
- またはcosθを-1から1まで一様に振ってもよい。
- ϕについては0から180度まで一様に振る。
- このような乱数はスペクトロメーターのアクセプタンスを求めたりするときや、3次元での運動学を求めるときに用いられる。

練習として2

- Λの崩壊するまでの距離をプロットする
  - pΛ=0.6 GeV/c
  - $c\tau$=7.89 cm

方針
ある距離dx進んだ時に、粒子がまだ崩壊しない確率は
$\exp(-dx/\beta\gamma c\tau)$
0から1まで、一様に乱数を振って、

(1) もし出た数が$\exp(-dx/\beta\gamma c\tau)$よりも大きければ、このdx進んだ間に崩壊したと考える。
(2) もし出た数が$\exp(-dx/\beta\gamma c\tau)$よりも小さければ、この間では崩壊しなかったと考える。

(2)となった場合は、dxをさらに進めて、同様に0-1の一様乱数をふり、値を比べ、崩壊するまで、このプロセスを繰り返す。
崩壊するまでdxを足しあげる事によって、粒子が飛んだ距離を求める。
その距離をヒストグラムに詰める。

この操作を何度か(各自が決めた粒子の数の分だけ)繰り返して見てみる。

出来上がったヒストグラムをexpの関数でfitして、$\beta\gamma c\tau$に対応する値が得られるか確かめる

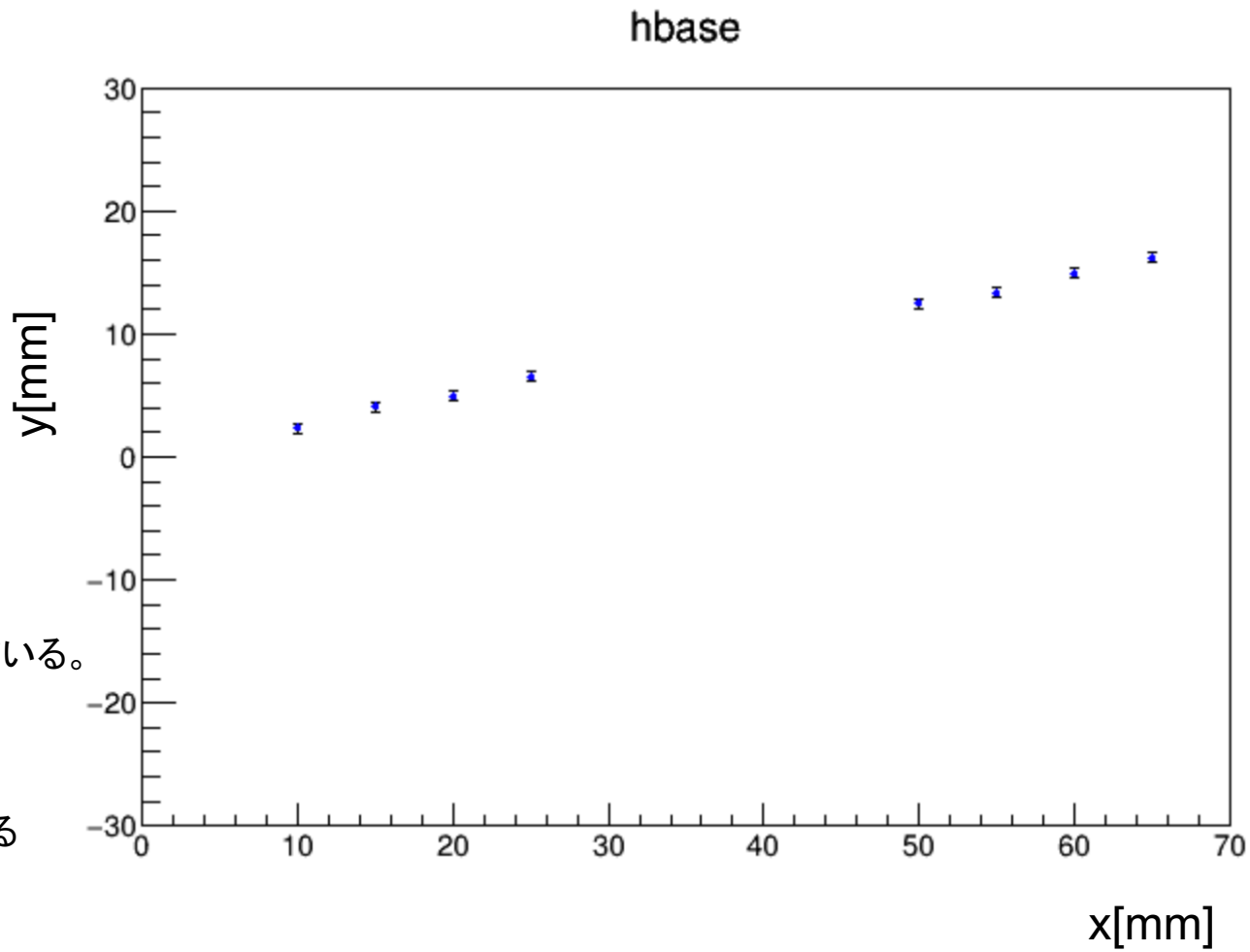
練習として3

```
void tracking()
{
  gStyle->SetOptStat(0);

  const int NumOfData = 8;
  double x[NumOfData]={10.,15., 20., 25., 50., 55., 60., 65. };
  double y[NumOfData];
  double sigma[NumOfData]={0.4, 0.4,  0.4, 0.4, 0.4, 0.4, 0.4, 0.4 };
  double x_sigma[NumOfData]={0., 0.,  0., 0., 0., 0., 0., 0. };

  TCanvas *c1 = new TCanvas("c1", "c1") ;
  TH2F *h2 = new TH2F("h2","hbase",100, 0, 70, 100, -30, 30);
  h2->Draw();

  for (int i=0; i<1000; i++) {
    double a = gRandom->Gaus(0, 0.25);
    std::cout << a << std::endl;

    for (int j=0; j<NumOfData; j++) {
      y[j] = a*x[j]+gRandom->Gaus(0, sigma[j]);
    }

    TGraphErrors *gr = new TGraphErrors(NumOfData, x, y, x_sigma, sigma);
    gr->SetMarkerStyle(20);
    gr->SetMarkerSize(0.5);
    gr->SetMarkerColor(kBlue);
    gr->Draw("p");
    c1->Update();
    gSystem->ProcessEvents();
    getchar();

    // fitting routine



    gr->Delete();

  }
}
```

- 粒子のトラッキング
  - /home/sks/miwa/MonteCarlo/track.C

y=a*x という直線を引いている。
各面で、位置をgauss型の乱数を振っている。

この続きで、y=a*x+bという関数でfitして、
各面での残差を1次元のヒストグラムに詰める

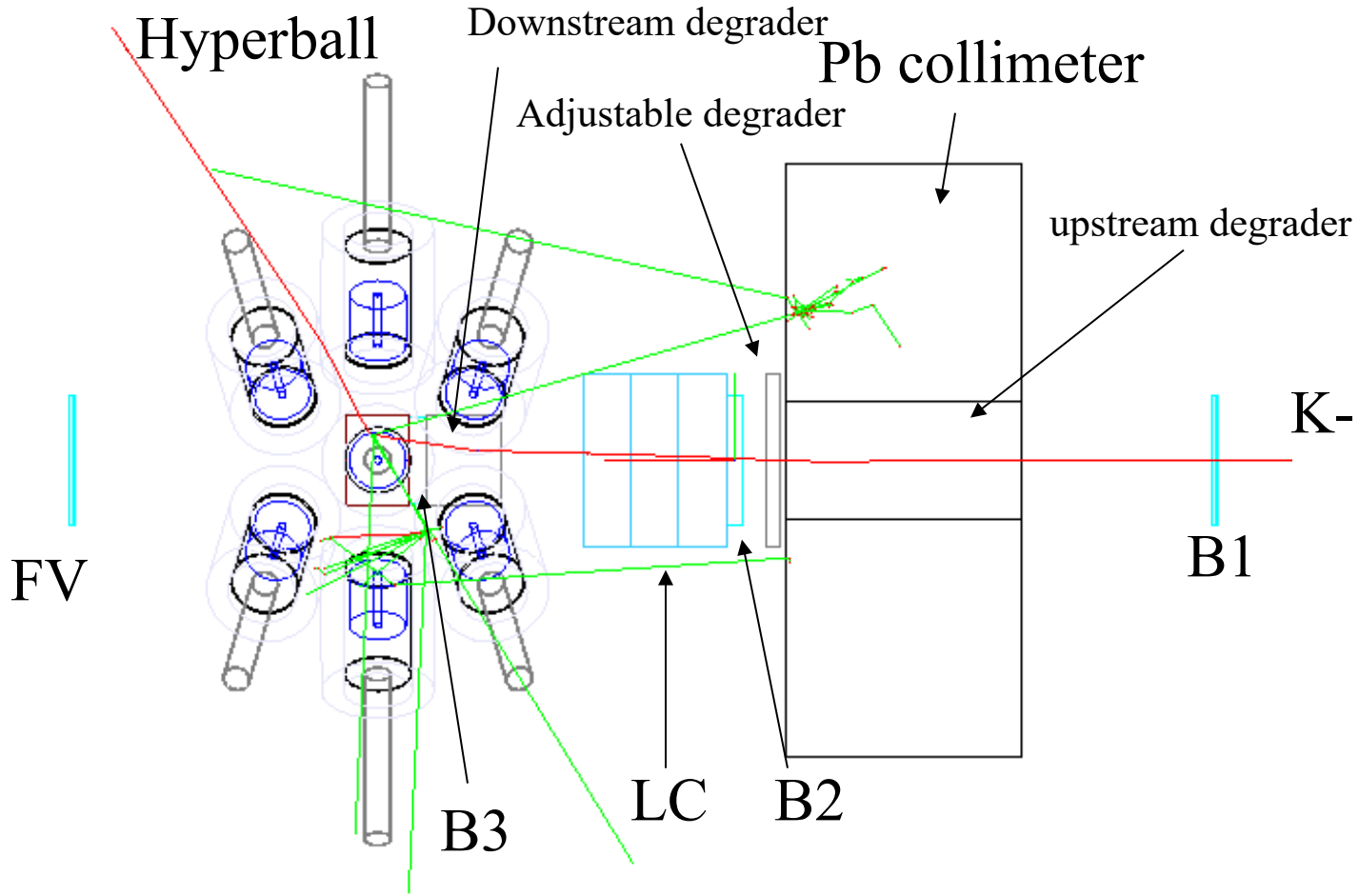


GEANTって何？

- CERNで主に開発された素粒子・原子核実験のシミュレーション用に開発されたツール
- 粒子を発生させ、様々な物理プロセスにしたがって粒子をtransportする。
  - 磁場や電場の中での粒子の運動
  - 電磁相互作用
  - ハドロン相互作用
  - 崩壊
- 検出器を配置して、その中でのエネルギー損失や、粒子の通過した場所などを調べることができる。

やってみると

- なんだか、やっぱりかっこいい。
- 物理やってる気がしてくる。
- 実験や物理を楽しむひとつのツールですね。



でも、よくよく考えてみると

- 粒子を発生させ、様々な物理プロセスにしたがって粒子をtransportする。
  - 磁場や電場の中での粒子の運動
    → 微分方程式を自分で解けは粒子の軌跡は計算できる
  - 電磁相互作用
    → 教科書に載っているプロセスで自分で計算できる
  - ハドロン相互作用
    → これも反応断面積が分かればあとは運動学を解くだけ
  - 崩壊
    → これも角度分布が分かればあとは運動学を解くだけ

GEANTは便利なツールだけれども、それを用いた結果が妥当であるかどうかをチェックできる目を身につけないといけない。
GEANTのツールを組み合わせて使う我々はすぐにミスしてしまうから。