

(17:00~19:00)

ROOT講習会

第5回 TTree, 亂数

2023/06/14

伊藤博士 (東京理科大)

itoh.hiroshi@rs.tus.ac.jp

※GitHubで

[ympateam/root-lecture/macos/hrs](https://github.com/ympateam/root-lecture/macos/hrs)
を事前にダウンロードしておいてください。

- 本資料の著作権、文責は著者に帰属し、所属機関を代表したり、機関の意見を表明するものではありません。
- 学校、研究機関の教育、研究目的であれば自由に使用することができます。報告なく改変、再配布可能です。
- ただし使用者は誤りや誤字を報告する義務があります。

自己紹介



いとう ひろし
伊藤 博士

東京理科大学

参加している実験：SK, HK, J-PARC E36

～興味のある物理や最近やっていること～

- 宇宙起源の反電子ニュートリノ探索
- レプトン普遍性破れ探索
- 暗黒物質・新物理探索
- 表面アルファ線分析
- Gd分析、BGOシンチの中性子応答

これまでに講習会でやってきたこと

TH1 : ヒストグラム

```
TH1F*h=new TH1F("h","h",100,0,10);
```

TGraph : グラフ

```
TGraph*g=new TGraph();
```

TF1 : 関数

```
TF1* f=new TF1("f","[0]*x+[1]",0,10);
```

第5回でやること

1. 復習かねて

相対論、粒子の運動をTH1, TGraphつかって遊ぼう

乱数も使って分布を作ってみよう

粒子識別っぽいことできるかな

2. TTree

- TTreeってなに？
- 粒子の崩壊を疑似的に再現してみよう

“root-lecture/macros/hrs/”をベースに解説してきます。

1. 復習かねて相対論、粒子の運動をTH1, TGraphつかって遊ぼう

せっかく素粒子物理を勉強したんだし、より実践に近いことしたいよね...

(復習：相対論)

ある系Sから、系Sにおいて速度比 $\beta=v/c$ で動く系S'に、ローレンツ変換した場合、粒子のエネルギーと運動量の関係は、

$$\begin{pmatrix} E' \\ p' \end{pmatrix} = \begin{pmatrix} \gamma & \gamma\beta \\ \gamma\beta & \gamma \end{pmatrix} \begin{pmatrix} E \\ p \end{pmatrix}$$

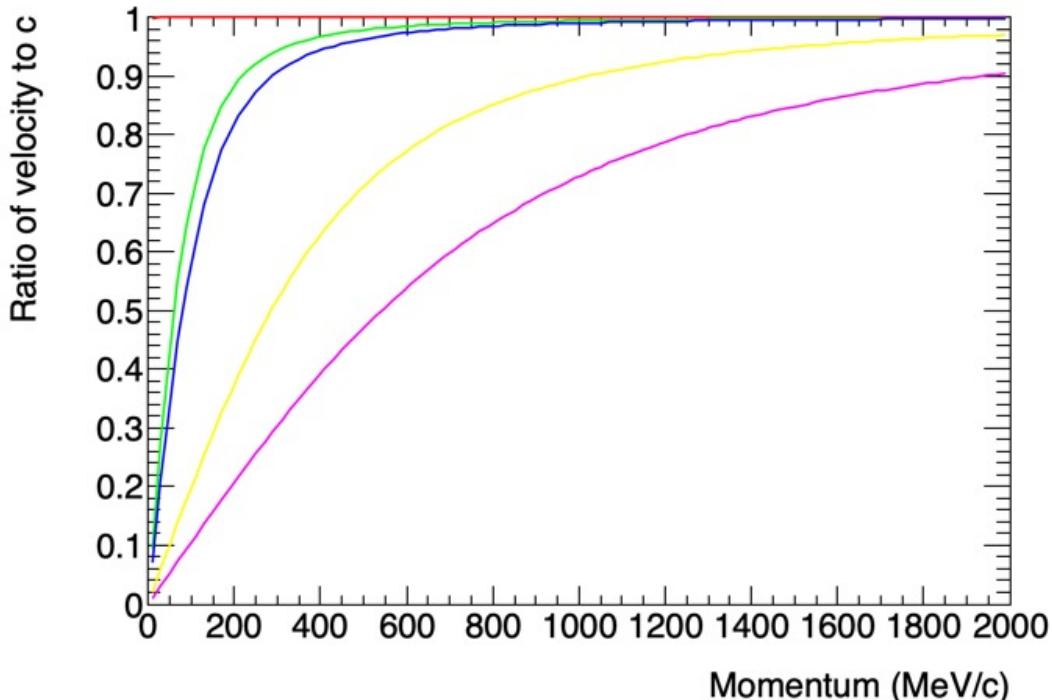
で表せます。粒子静止系S: $p=(E, \mathbf{p})=(m, \mathbf{0})$ とすると簡単。

$$E = \gamma m, p = \beta \gamma m, \beta = 1/\sqrt{1 + (m/p)^2}$$

※ β の正負は簡単のため無視してます。
細かなことは言わないのでくださいね。

例題1) この関係を使って、荷電粒子
 e^+ , μ^+ , π^+ , K^+ , 陽子の運動量と速度
比の関係を書いてみよう。

```
$cd [workDIR]/root-lecture/macros/hrs/  
$root -l src/exam1.cxx
```



```
1 inline void exam1();                                exam1.cxx  
2 const double m_electron=0.511; // MeV/c2  
3 const double m_muon=105.6;    // MeV/c2  
4 const double m_pion=139.6;   // MeV/c2  
5 const double m_kaon=493.7;   // MeV/c2  
6 const double m_proton=938.3; // MeV/c2  
7 const int n = 5;  
8  
9 void exam1(){  
10   gROOT->SetStyle("ATLAS");  
11  
12   double mom, beta;  
13   double m[n]={  
14     m_electron, m_muon, m_pion,  
15     m_kaon, m_proton  
16   };
```

まず、粒子質量など定義していますね。
違う粒子を使いたい場合は、
PDGのサイトから、質量を探しましょう。

補足: “**inline void exam1()**”

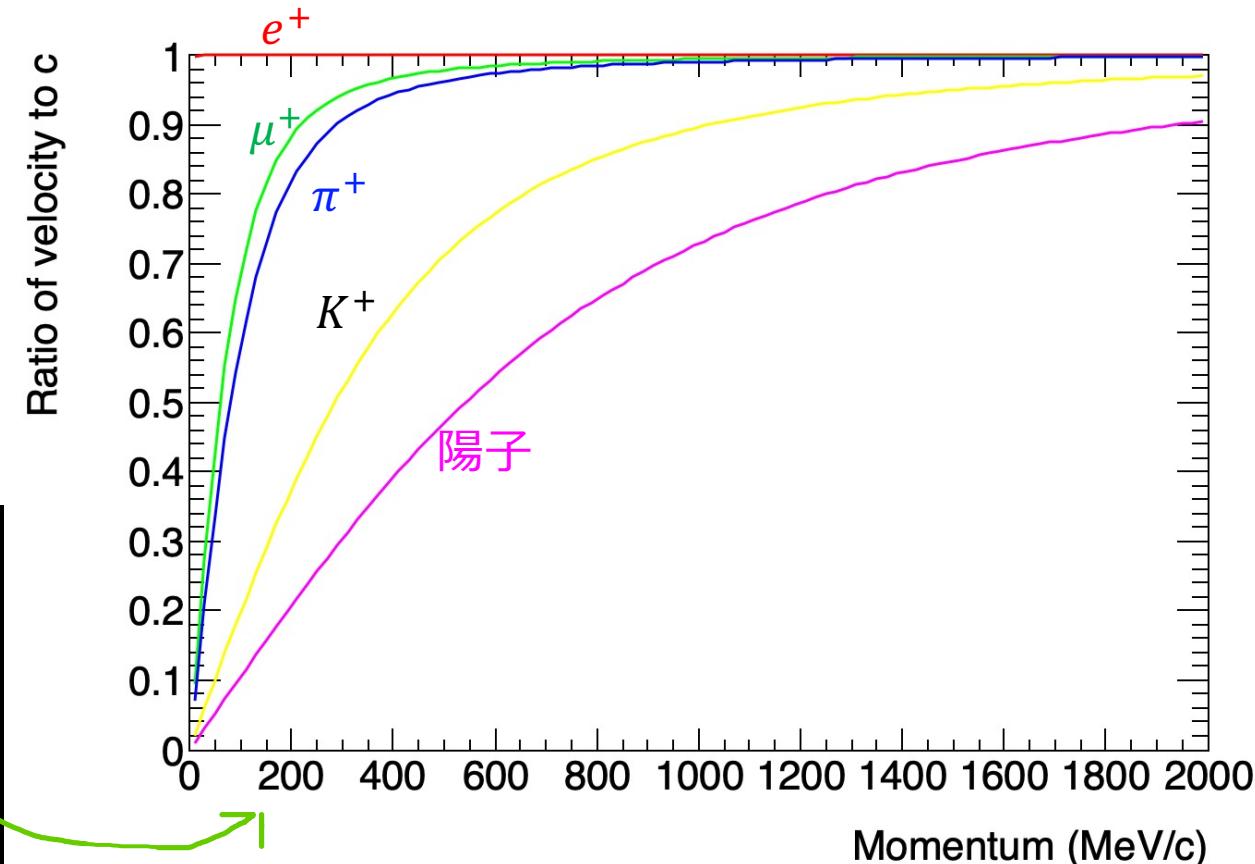
コード下で定義した自作関数を事前に、
宣言しておく関数です。

例題1) この関係を使って、荷電粒子
 e^+ , μ^+ , π^+ , K^+ , 陽子の運動量と速度
 比の関係を書いてみよう。

```

20   TGraph* graph[n];
21   for(int j=0;j<n;j++){
22     graph[j] = new TGraph();
23     for(int i=0;i<100;i++){
24       mom = i*20+10; //MeV/c
25       beta = 1./sqrt(1. + pow(m[j]/mom,2));
26       graph[j]->SetPoint(i, mom, beta);
27     }
28     graph[j]->SetLineColor(j+2);
29     graph[j]->SetLineWidth(3);
30     if(j==0){
31       graph[0]->GetHistogram()->SetMinimum(0);
32       graph[0]->GetHistogram()->SetMaximum(1);
33       graph[0]->GetXaxis()->SetLimits(0,2000);
34       graph[0]->GetXaxis()->SetTitle("Momentum (MeV/c)");
35       graph[0]->GetYaxis()->SetTitle("Ratio of velocity to c");
36       graph[0]->Draw("al");

```



0 ~ 2000まで、20ステップで
 ループさせてますね。

$$\beta = 1/\sqrt{1 + (m/p)^2}$$

実習1) このコードを改造して、 π^0 , D^+ , B^+ , B_s^0 , B_c^+ 粒子の運動量と速度比の関係を書いてみよう。(5分)

1. 粒子質量をPDGのサイトから探し
て、コードに追記しましょう。

<https://pdglive.lbl.gov/Viewer.action>

検索ワード「PDG」

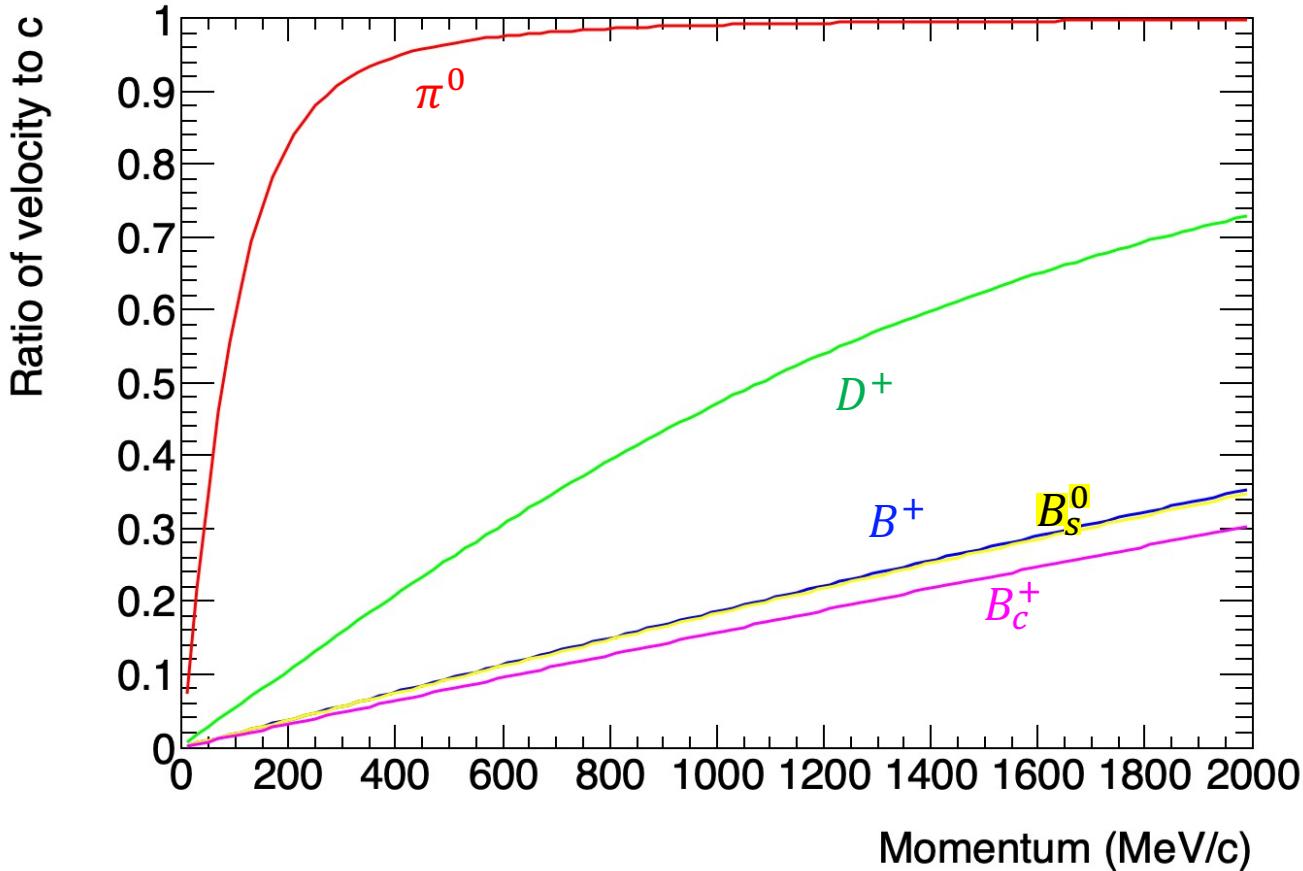
2. $m[n]=\{ \dots \}$
の箇所を編集して、適切な粒子
をセットしましょう。

```
1 inline void exam1(); // exam1.cxx
2 const double m_electron=0.511; // MeV/c2
3 const double m_muon=105.6; // MeV/c2
4 const double m_pion=139.6; // MeV/c2
5 const double m_kaon=493.7; // MeV/c2
6 const double m_proton=938.3; // MeV/c2
7 const int n = 5;
8
9 void exam1(){
10     gROOT->SetStyle("ATLAS");
11
12     double mom, beta;
13     double m[n]={
14         m_electron, m_muon, m_pion,
15         m_kaon, m_proton
16     };

```

画像をスクショしてどんな感じになったか、
チャットでお知らせください。

実習1) このコードを改造して、 π^0 , D^+ , B^+ , B_s^0 , B_c^+ 粒子の運動量と速度比の関係を書いてみよう。(5分)



```

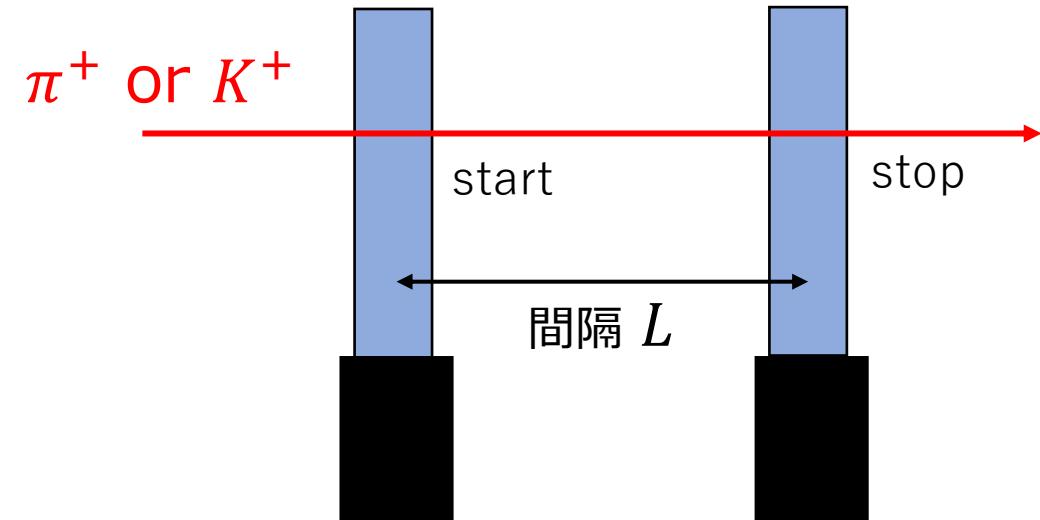
1 inline void exam1ex();
2 const double m_electron=0.511; // MeV/c2
3 const double m_muon=105.6; // MeV/c2
4 const double m_pion=139.6; // MeV/c2
5 const double m_kaon=493.7; // MeV/c2
6 const double m_proton=938.3; // MeV/c2
7
8 const double m_pi0=134.9768; // MeV/c2
9 const double m_D=1869.66; // MeV/c2
10 const double m_B=5279.34; // MeV/c2
11 const double m_Bs=5366.92; // MeV/c2
12 const double m_Bc=6274.47; // MeV/c2
13 const int n = 5;
14
15 void exam1ex(){
16     gROOT->SetStyle("ATLAS");
17
18     double mom, beta;
19     double m[n]={
20         //m_electron, m_muon, m_pion, m_kaon, m_proton
21         m_pi0, m_D, m_B, m_Bs, m_Bc
22     };
23 }
```

こんな感じでしょうか

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

前提条件

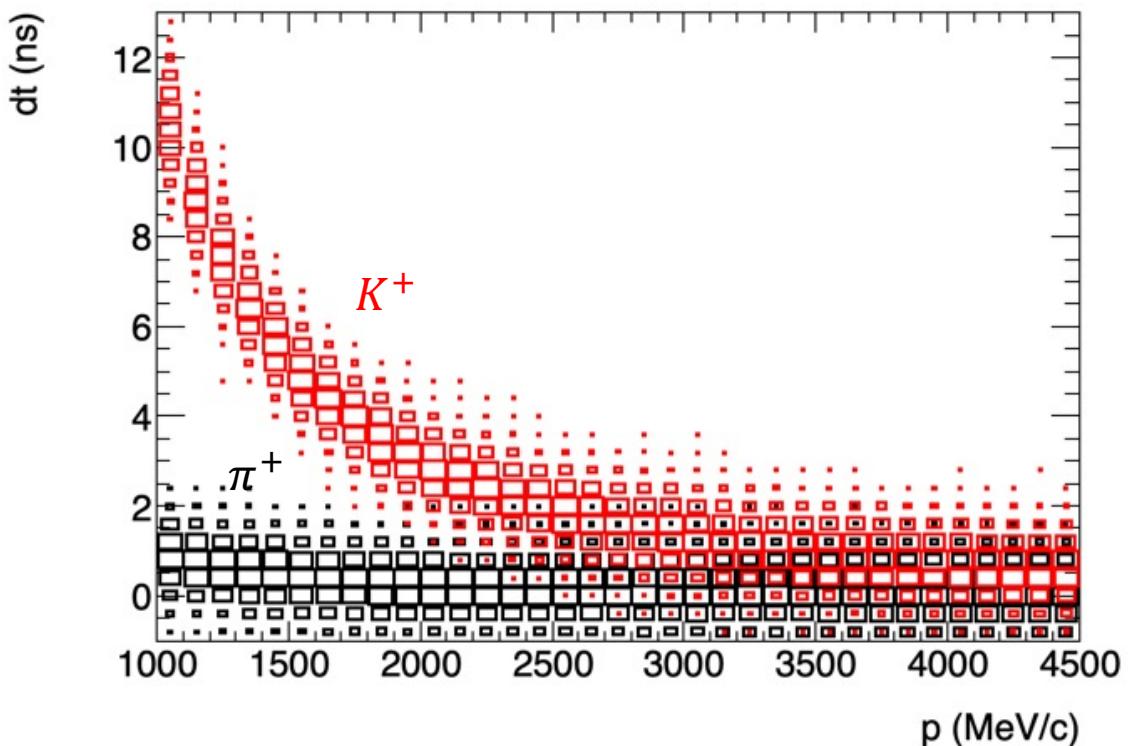
- プラシン検出器の間隔 $L = 30 \text{ m}$
- プラシン検出器の時間分解能 $\sigma_t = 500 \text{ ps}$
- 運動量は別の検出器でわかっているものとして $p=1\sim4.5 \text{ GeV}/c$
- までみてみる。運動量分解能は $5 \text{ MeV}/c$ としてみましょうか。



Time-of-Flight法: 飛行時間による粒子識別のこと。同じ運動量でも質量によって飛行時間が異なるため、粒子識別が可能である。

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

```
$root -l src/exam2.cxx
```

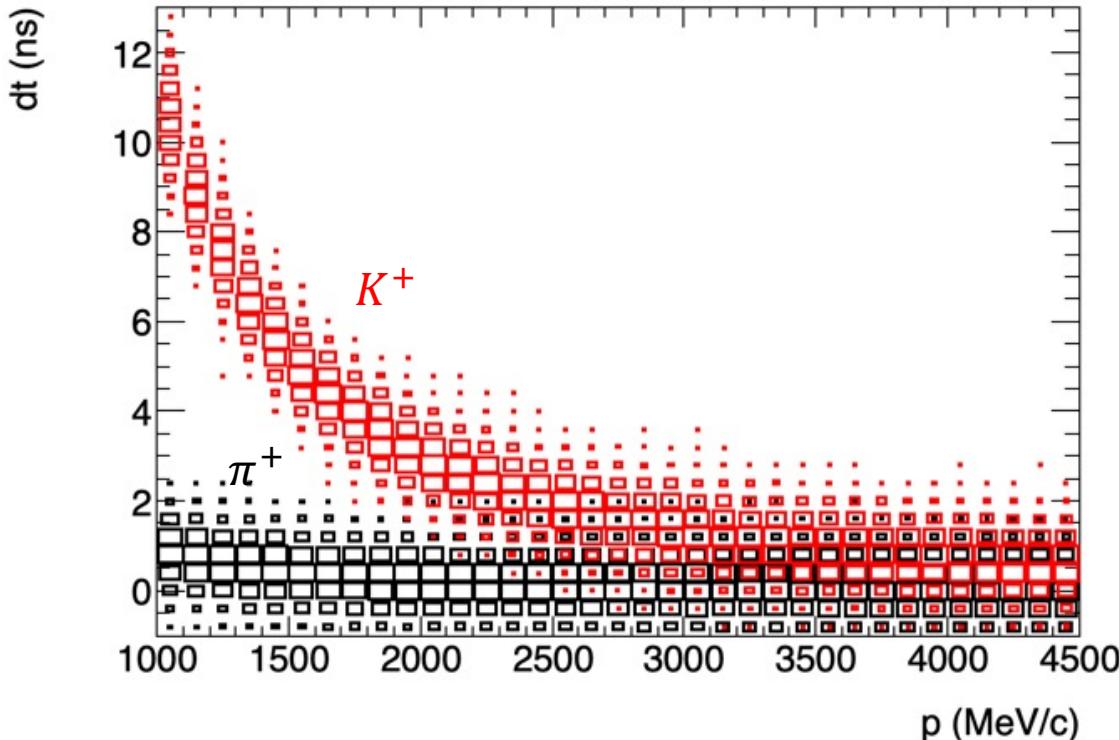


src/exam2.cxxを開きましょう。
まずは不变な変数を定義します。

```
1 inline void exam2();  
2  
3 const double m_pion=139.6; // [MeV/c2]  
4 const double m_kaon=493.7; // [MeV/c2]  
5  
6 // timing resolution [defo: 500 ps]  
7 const double time_resolution = 500e-12;  
8  
9 // momentum resolution [defo: 5 MeV/c]  
10 const double mom_resolution = 5;  
11 const double L = 30.0; // [m]  
12 const double numevents = 1e5;  
13 const double c = 2.99792458 * 1e8; // m/s  
14  
15 const int verbose = 0;
```

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

\$root -l src/exam2.cxx



```

21     double mom,mom_true, beta, dt, t0, t1;           exam2.cxx
22
23     auto h_pion = new TH2F("h_pion","h_pion",35,1000,4500,35,-1,13);
24     auto h_kaon = new TH2F("h_kaon","h_kaon",35,1000,4500,35,-1,13);
25
26     for(int j=0;j<numevents;j++){
27
28         mom_true = gRandom->Uniform(1000, 4500);
29         mom = gRandom->Gaus(mom_true, mom_resolution);
30         if(verbose)
31             cout<<"momentum ="<<mom<<" MeV/c; ";
32
33         // - pion
34         beta = 1./sqrt(1. + pow(m_pion/mom_true,2));
35         t0 = L/c;
36         t1 = L / beta / c;
37         dt = t1 - t0;
38         dt = gRandom->Gaus(dt, time_resolution);
39         dt *=1e9;
40         h_pion->Fill(mom,dt);

```

乱数を使って、擬似データを作ります。

gRandom->Uniform(a, b)

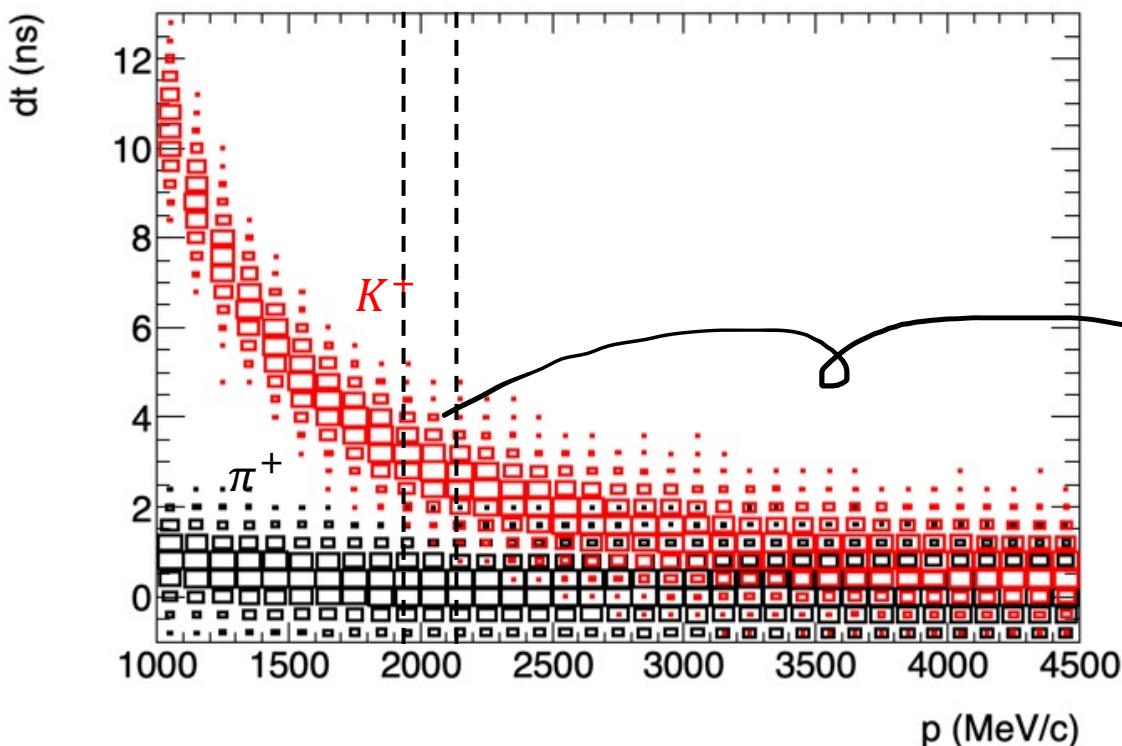
[a,b]の領域を一様乱数を生成します。

gRandom->Gaus(μ , σ)

平均値 μ , 分散 σ のガウス分布で乱数を生成します。
これを使って、検出応答を擬似再現します。

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

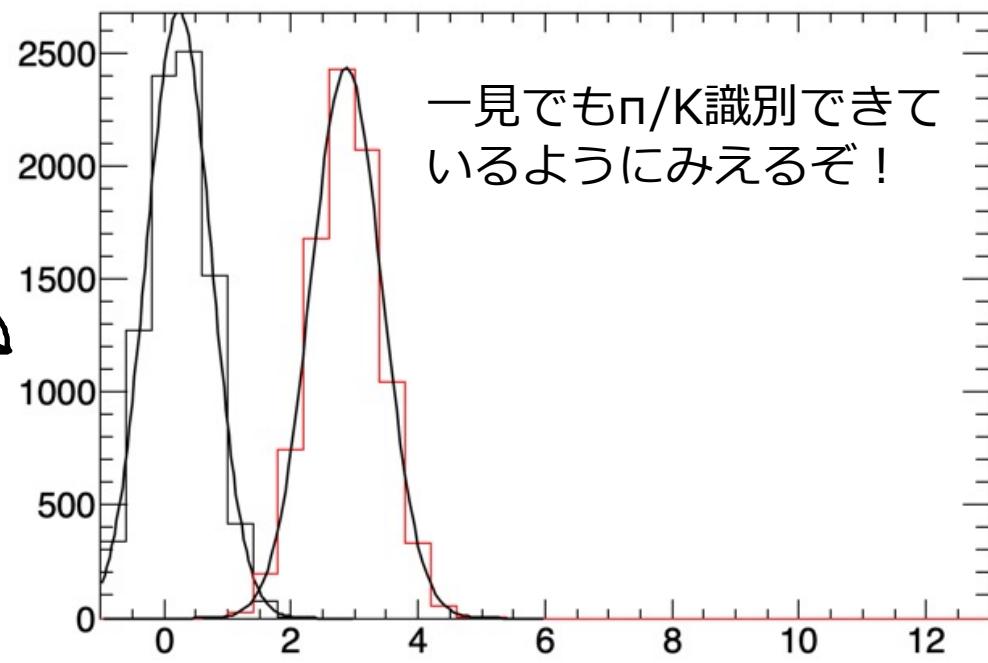
```
$root -l src/exam2.cxx
```



「もうちょっと、遊んでみたい！」
たとえば、粒子識別能力はどのくらいなのか、定量できないか？

「縦にスライスして1次元分布に射影して、K, π の分布を比べてみましょう。」

L16. **const int sw=1;** とスイッチ変えて、再度回してみてください。

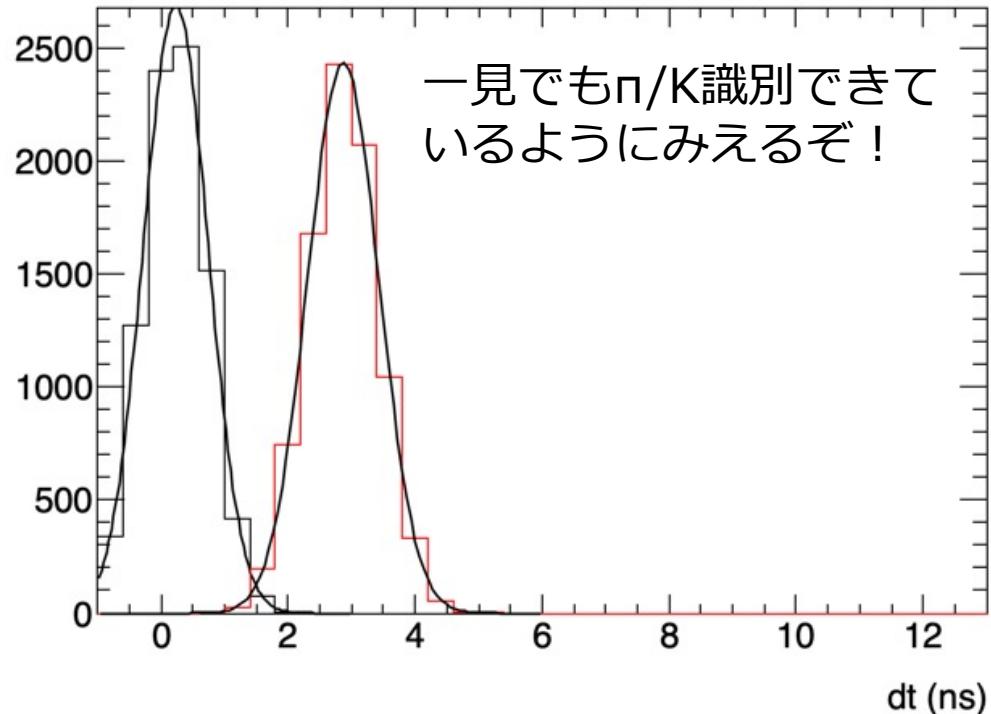


一見でも π/K 識別できているように見えるぞ！

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

\$root -l src/exam2.cxx

L16. const int sw=1;



```

56 auto c1=new TCanvas("c1");
57 h_pion->SetXTitle("p (MeV/c)");
58 h_pion->SetYTitle("dt (ns)");
59 h_pion->SetLineColor(1);
60 h_pion->Draw("box");
61 h_kaon->SetLineColor(2);
62 h_kaon->Draw("boxsame");
63
64 if(sw<1) return;
65
66 auto hk=new TH1D("hk","hk",35,-1,13);
67 auto hp=new TH1D("hp","hp",35,-1,13);
68 h_kaon->ProjectionY("hk",10,12);
69 h_pion->ProjectionY("hp",10,12);
70 hp->Draw();
71 hk->Draw("same");
72 hp->Fit("gaus","","",-1,5);
73 hk->Fit("gaus","","",-1,5);
74

```

2次元ヒストグラムをスライスしているコマンド

```

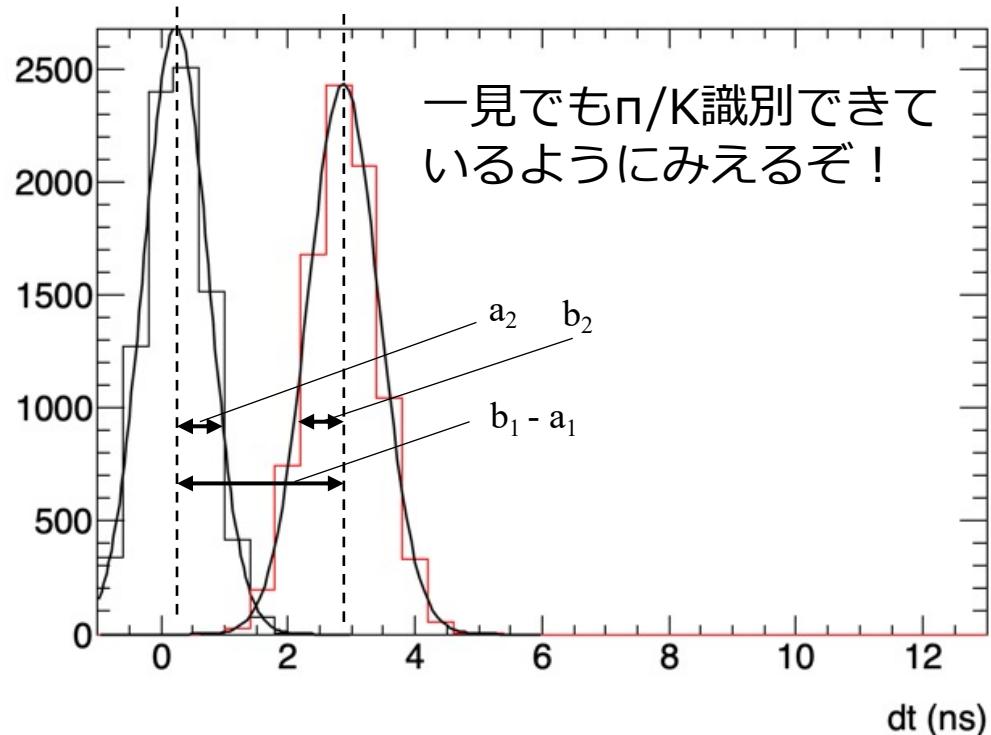
root [0]
Processing src/exam2.cxx...
FCN=6.91273 FROM MIGRAD    STATUS=CONVERGED    64 CALLS      65 TOTAL
                           EDM=1.06262e-08   STRATEGY= 1    ERROR MATRIX ACCURATE
EXT PARAMETER            STEP         FIRST
NO. NAME        VALUE       ERROR        SIZE      DERIVATIVE
 1 Constant     2.69400e+03  3.61448e+01  4.00688e-02 -3.41646e-07
 2 Mean          2.32665e-01  5.68860e-03  7.75130e-06 -2.47440e-02
 3 Sigma         5.08436e-01  4.24337e-03  3.10743e-06  5.81256e-03
FCN=2.28018 FROM MIGRAD    STATUS=CONVERGED    52 CALLS      53 TOTAL
                           EDM=3.09897e-11   STRATEGY= 1    ERROR MATRIX ACCURATE
EXT PARAMETER            STEP         FIRST
NO. NAME        VALUE       ERROR        SIZE      DERIVATIVE
 1 Constant     2.43916e+03  3.21900e+01  2.32609e-02 -2.64789e-07
 2 Mean          2.87170e+00  6.07161e-03  5.36875e-06 -5.63041e-05
 3 Sigma         5.62594e-01  4.28504e-03  1.83474e-06 -3.37583e-03
root [1]

```

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

\$root -l src/exam2.cxx

L16. **const int sw=1;**



```

62   h_kaon->Draw("boxsame");
63
64   if(sw<1) return;
65
66   auto hk=new TH1D("hk","hk",35,-1,13);
67   auto hp=new TH1D("hp","hp",35,-1,13);
68   h_kaon->ProjectionY("hk",10,12);
69   h_pion->ProjectionY("hp",10,12);
70   hk->Draw();
71   hk->Draw("same");
72   hp->Fit("gaus","","",-1,5);
73   hk->Fit("gaus","","",-1,5);
74

```

2次元ヒストグラムをスライスしているコマンド
ガウスフィット

\$root -l src/exam2.cxxしたら
以下コマンドを書いてみましょうか

```

root [1] double a1 = hk->GetFunction("gaus")->GetParameter(1)
(double) 2.8717003
root [2] double a2 = hk->GetFunction("gaus")->GetParameter(2)
(double) 0.56259443
root [3] double b1 = hp->GetFunction("gaus")->GetParameter(1)
(double) 0.23266471
root [4] double b2 = hp->GetFunction("gaus")->GetParameter(2)
(double) 0.50843620
root [5] (a1-b1)/sqrt(a2*a2+b2*b2)
(double) 3.4801969

```

つまり、TOFにおいて、2.0~2.2 GeV/c領域の π /Kの識別能力は、

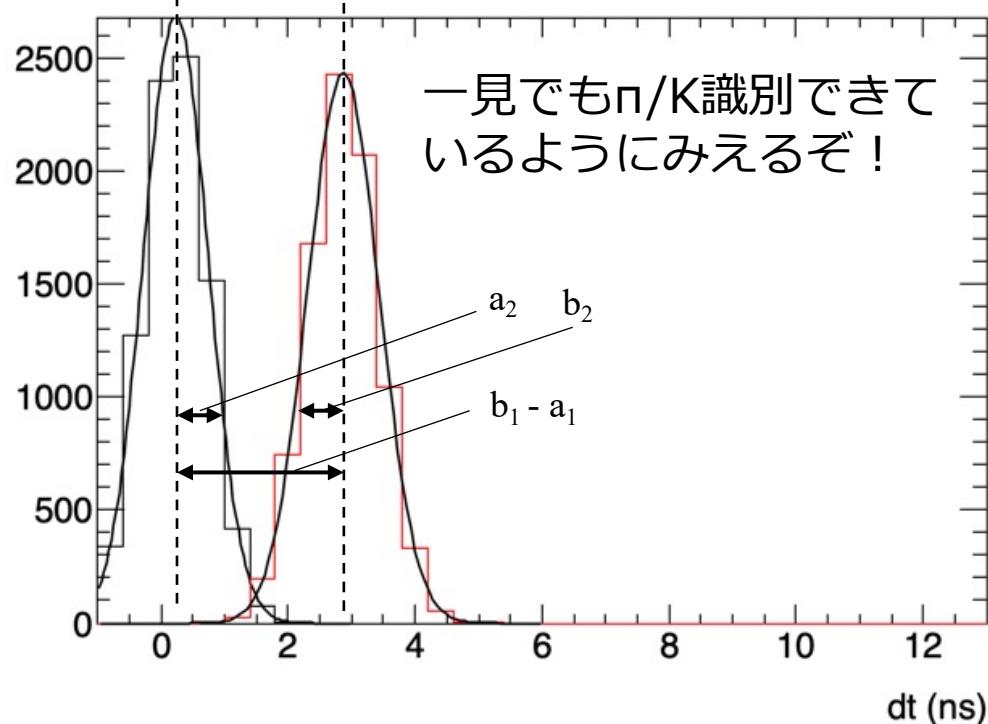
$\sim 3.48\sigma$

であるということでしょう。

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

\$root -l src/exam2.cxx

L16. const int sw=2;



毎回コマンド打つのは嫌なのでマクロに追加しました。
L16. sw=1からsw=2に変更してください。

exam2.cxx

```

71     hk->Draw("same");
72     hp->Fit("gaus","","",-1,5);
73     hk->Fit("gaus","","",-1,5);
74
75     if(sw<2) return;
76
77     double sigma = abs(hp->GetFunction("gaus")->GetParameter(1) -
78         hk->GetFunction("gaus")->GetParameter(1))/sqrt(pow(hp->GetFunction
79             ("gaus")->GetParameter(2),2)+pow(hk->GetFunction("gaus")->GetParameter(2),2));
80
81     cout<<"significant = "<<sigma<<endl;
82 }
```

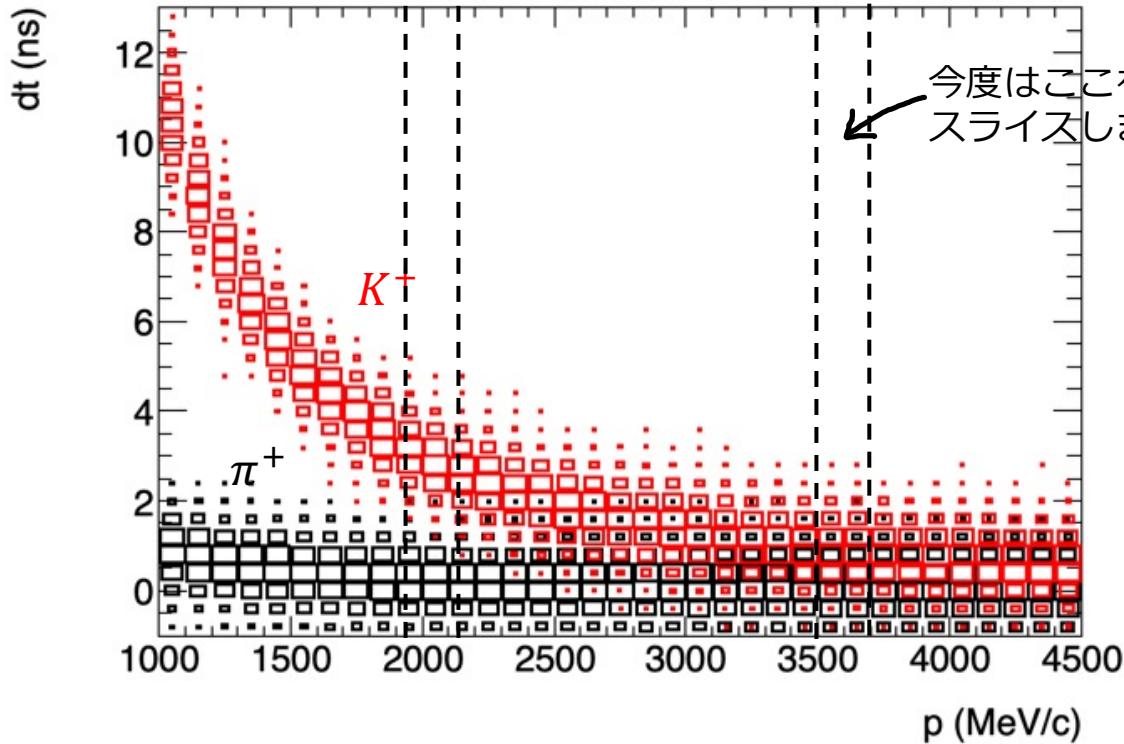
```

root [0]
Processing src/exam2.cxx...
FCN=6.91273 FROM MIGRAD    STATUS=CONVERGED          64 CALLS      65 TOTAL
                           EDM=1.06262e-08   STRATEGY= 1    ERROR MATRIX ACCURATE
EXT PARAMETER            VALUE        ERROR        STEP          FIRST
NO.  NAME        VALUE        ERROR        SIZE        DERIVATIVE
 1  Constant    2.69400e+03  3.61448e+01  4.00688e-02 -3.41646e-07
 2  Mean        2.32665e-01  5.68860e-03  7.75130e-06 -2.47440e-02
 3  Sigma       5.08436e-01  4.24337e-03  3.10743e-06  5.81256e-03
FCN=2.28018 FROM MIGRAD    STATUS=CONVERGED          52 CALLS      53 TOTAL
                           EDM=3.09897e-11   STRATEGY= 1    ERROR MATRIX ACCURATE
EXT PARAMETER            VALUE        ERROR        STEP          FIRST
NO.  NAME        VALUE        ERROR        SIZE        DERIVATIVE
 1  Constant    2.43916e+03  3.21900e+01  2.32609e-02 -2.64789e-07
 2  Mean        2.87170e+00  6.07161e-03  5.36875e-06 -5.63041e-05
 3  Sigma       5.62594e-01  4.28504e-03  1.83474e-06 -3.37583e-03
significant = 3.4802
root [1]
```

もっと、高い運動量の時はどうかな？

例題2) Time-of-flightを考えて、 π^+ と K^+ の到達時間の分布をつくって、粒子識別ができるか試してみよう。

\$root -l src/exam2.cxx



```

56 auto c1=new TCanvas("c1");
57 h_pion->SetXTitle("p (MeV/c)");
58 h_pion->SetYTitle("dt (ns)");
59 h_pion->SetLineColor(1);
60 h_pion->Draw("box");
61 h_kaon->SetLineColor(2);
62 h_kaon->Draw("boxsame");
63
64 if(sw<1) return;
65
66 auto hk=new TH1D("hk","hk",35,-1,13);
67 auto hp=new TH1D("hp","hp",35,-1,13);
68 h_kaon->ProjectionY("hk",10,12);
69 h_pion->ProjectionY("hp",10,12);
70 hp->Draw();
71 hk->Draw("same");
72 hp->Fit("gaus","","",-1,5);
73 hk->Fit("gaus","","",-1,5);
74

```

exam2.cxx

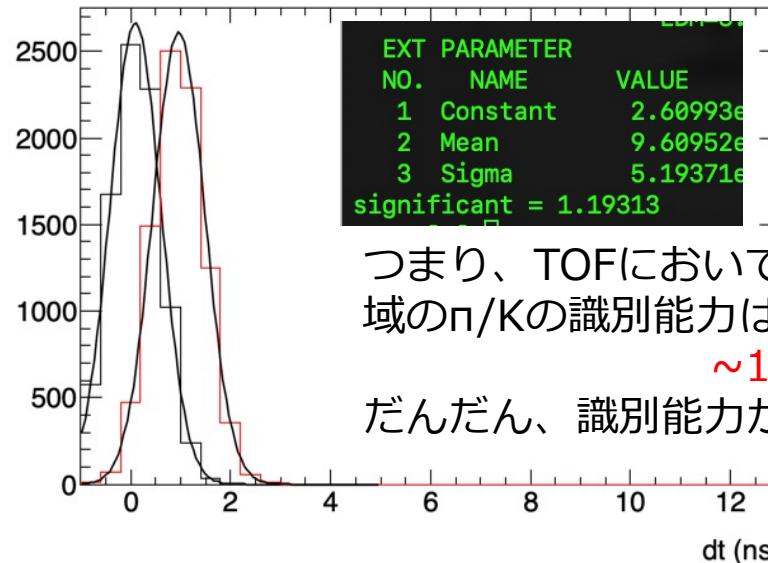
$(10,12) \Leftrightarrow (25,27)$

```

66 auto hk=new TH1D("hk","hk",35,-1,13);
67 auto hp=new TH1D("hp","hp",35,-1,13);
68 h_kaon->ProjectionY("hk",25,27);
69 h_pion->ProjectionY("hp",25,27);
70 hp->Draw();

```

L68, 69の引数2, 3を(10,12)から(15,17)に
変えて、再度root -l src/exam2.cxxを回してください。



実習2) exam2.cxx を改造して、
3.5~3.7 MeVでPID 3σ以上のために、
時間分解能は何psが要求される？

```
1 inline void exam2();  
2  
3 const double m_pion=139.6; // [MeV/c2]  
4 const double m_kaon=493.7; // [MeV/c2]  
5  
6 // timing resolution [defo: 500 ps]  
7 const double time_resolution = 500e-12;  
8  
9 // momentum resolution [defo: 5 MeV/c]  
10 const double mom_resolution = 5;  
11 const double L = 30.0; // [m]  
12 const double numevents = 1e5;  
13 const double c = 2.99792458 * 1e8; // m/s  
14  
15 const int verbose = 0;
```

10分：計算できたら答えをチャットに書いてみよう。

誰が早く計算できるかな？ 10分経ったら解説します（ヒント）

1. exam2.cxxを同じ場所にコピーしましょう。名前は、例えば、exam2ex.cxxとします。
2. double exam2ex(double resolution){
として、関数に引数を与えます。
3. 最後に、sigmaは
$$(b1-a1)/\sqrt{a2^2 + b2^2}$$

で計算して、最後にreturn sigma;をつける。
4. 外部でexam2ex(resolution);を実行したら、
sigmaを返す関数として機能します。
5. 新たに、マクロを作成しましょう。例えば、
exam2out.cxxとかでしょうか。
6. for文でresolution = 500e-12から徐々に減らして、coutで確認すると、時間分解能に応じたPID significantが羅列されます。

実習2) exam2.cxx を改造して、
3.5~3.7 MeVでPID 3 σ 以上のために、
時間分解能は何psが要求される？

```
1 double exam2ex(double time_resolution){  
2  
3     gROOT->SetStyle("ATLAS");  
4     const int verbose = 0;  
5     const double m_pion=139.6;  
6     const double m_kaon=493.7;  
7     const double mom_resolution = 5;  
8     const double L = 30.0;// [m]  
9     const double numevents = 1e5;  
10    const double c = 2.99792458 * 1e8; // m/s  
11  
12    double mom,mom_true, beta, dt, t0, t1;  
13  
14    auto h_pion = new TH2F("h_pion","h_pion",35,1000,4500,35,-1,13);  
15    auto h_kaon = new TH2F("h_kaon","h_kaon",35,1000,4500,35,-1,13);  
16  
17    for(int j=0;j<numevents;j++){  
18  
19        mom_true = gRandom->Uniform(1000, 4500);  
20        mom = gRandom->Gaus(mom_true, mom_resolution);  
21        if(verbose)  
22            cout<<"momentum ="<<mom<<" MeV/c; "  
23
```

} 関数の中に入れましょう

(ヒント)

1. exam2.cxxを同じ場所にコピーしましょう。名前は、例えば、exam2ex.cxxとします。
2. double exam2ex(double resolution){
として、関数に引数を与えます。
3. 最後に、sigmaは
$$(b_1 - a_1) / \sqrt{a_2^2 + b_2^2}$$

で計算して、最後にreturn sigma;をつける。
4. 外部でexam2ex(resolution);を実行したら、
sigmaを返す関数として機能します。
5. 新たに、マクロを作成しましょう。例えば、
exam2out.cxxとかでしょうか。
6. for文でresolution = 500e-12から徐々に減らして、coutで確認すると、時間分解能に応じたPID significantが羅列されます。

実習2) exam2.cxx を改造して、
3.5~3.7 MeVでPID 3σ以上のために、
時間分解能は何psが要求される？

```
53     auto hk=new TH1D("hk","hk",35,-1,13);
54     auto hp=new TH1D("hp","hp",35,-1,13);
55     h_kaon->ProjectionY("hk",25,27);
56     h_pion->ProjectionY("hp",25,27);
57     hp->Draw();
58     hk->Draw("same");
59     hp->Fit("gaus","Q","", -1,4);
60     hk->Fit("gaus","Q","", -1,4);
61     double sigma = abs(hp->GetFunction("gaus")->GetParameter(1) -
62     hk->GetFunction("gaus")->GetParameter(1))/sqrt(pow(hp->GetFunction
63     ("gaus")->GetParameter(2),2)+pow(hk->GetFunction("gaus")->GetParameter(2),2));
64     cout<<sigma<<endl;
65
66     h_pion->Delete();
67     h_kaon->Delete();
68     hk->Delete();
69     hp->Delete();
70     c1->Close();
71
72 }  
ポインタを消去します。  
毎回auto h_pion=new TH1F***などでポイ  
ント生成しているので、これがないと「すでに  
存在するよ」とエラーが出ます。
```

- (ヒント)
1. exam2.cxxを同じ場所にコピーしましょう。名前は、例えば、exam2ex.cxxとします。
 2. double exam2ex(double resolution){
として、関数に引数を与えます。
 3. 最後に、sigmaは
$$\frac{(b_1-a_1)}{\sqrt{a_2^2+b_2^2}}$$

で計算して、最後にreturn sigma;をつける。
 4. 外部でexam2ex(resolution);を実行したら、
sigmaを返す関数として機能します。
 5. 新たに、マクロを作成しましょう。例えば、
exam2out.cxxとかでしょうか。
 6. for文でresolution = 500e-12から徐々に減らし
て、coutで確認すると、時間分解能に応じたPID
significantが羅列されます。

実習2) exam2.cxx を改造して、
3.5~3.7 MeVでPID 3σ以上のために、
時間分解能は何psが要求される？

```
1 #include "../src/exam2ex.cxx" ←読み込みファイル指定  
2 void exam2out(){  
3     double reso;  
4     for(int i=0; i<40; i++){  
5         reso = (500-i*10)*1e-12;  
6         cout<<"resolution = "<<reso*1e12<<" ps: "  
7         cout<<"pi/K pid significant = "  
8         exam2ex(reso); ←ソースコードの位置からのアドレスです。  
9     }  
10  
11     return;  
12 }  
13 }
```

- (ヒント)
1. exam2.cxxを同じ場所にコピーしましょう。名前は、例えば、exam2ex.cxxとします。
 2. double exam2ex(double resolution){として、関数に引数を与えます。
 3. 最後に、sigmaは $(b_1 - a_1)/\sqrt{a_2^2 + b_2^2}$ で計算して、最後にreturn sigma;をつける。
 4. 外部でexam2ex(resolution);を実行したら、sigmaを返す関数として機能します。
 5. 新たに、マクロを作成しましょう。例えば、exam2out.cxxとかでしょうか。
 6. for文でresolution = 500e-12から徐々に減らして、coutで確認すると、時間分解能に応じたPID significantが羅列されます。

実習2) exam2.cxx を改造して、
3.5~3.7 MeVでPID 3σ以上のために、
時間分解能は何psが要求される？

```
[MacBook-Pro-81:hrs hiroshi$ root -l src/exam2out.cxx
root [0]
Processing src/exam2out.cxx...
resolution = 500 ps: pi/K pid significant = 1.19313
resolution = 490 ps: pi/K pid significant = 1.24607
resolution = 480 ps: pi/K pid significant = 1.29115
resolution = 470 ps: pi/K pid significant = 1.28375
resolution = 460 ps: pi/K pid significant = 1.33036
resolution = 450 ps: pi/K pid significant = 1.36324
resolution = 440 ps: pi/K pid significant = 1.37355
resolution = 430 ps: pi/K pid significant = 1.41224
resolution = 420 ps: pi/K pid significant = 1.42472
resolution = 410 ps: pi/K pid significant = 1.48383
resolution = 400 ps: pi/K pid significant = 1.52846
resolution = 390 ps: pi/K pid significant = 1.51949
resolution = 380 ps: pi/K pid significant = 1.57266
resolution = 370 ps: pi/K pid significant = 1.6312
resolution = 360 ps: pi/K pid significant = 1.68954
resolution = 350 ps: pi/K pid significant = 1.69874
resolution = 340 ps: pi/K pid significant = 1.74045
resolution = 330 ps: pi/K pid significant = 1.77258
resolution = 320 ps: pi/K pid significant = 1.82647
resolution = 310 ps: pi/K pid significant = 1.89821
resolution = 300 ps: pi/K pid significant = 1.96387
```

```
resolution = 290 ps: pi/K pid significant = 2.01532
resolution = 280 ps: pi/K pid significant = 2.08084
resolution = 270 ps: pi/K pid significant = 2.11484
resolution = 260 ps: pi/K pid significant = 2.17685
resolution = 250 ps: pi/K pid significant = 2.27831
resolution = 240 ps: pi/K pid significant = 2.35389
resolution = 230 ps: pi/K pid significant = 2.45109
resolution = 220 ps: pi/K pid significant = 2.49984
resolution = 210 ps: pi/K pid significant = 2.59257
resolution = 200 ps: pi/K pid significant = 2.71632
resolution = 190 ps: pi/K pid significant = 2.79941
resolution = 180 ps: pi/K pid significant = 2.92918
resolution = 170 ps: pi/K pid significant = 3.0355
resolution = 160 ps: pi/K pid significant = 3.18418
resolution = 150 ps: pi/K pid significant = 3.27259
resolution = 140 ps: pi/K pid significant = 3.49313
resolution = 130 ps: pi/K pid significant = 3.67183
resolution = 120 ps: pi/K pid significant = 3.89055
resolution = 110 ps: pi/K pid significant = 3.9924
root [1]
```

こうしてみると、
時間分解能 170ps以下が要求されるとわかる。

ここまでで習得したこと

- ・乱数を使って、擬似データを生成した。
- ・#includeを使って、外部ソースの自作関数を呼び出した。

相対論よりローレンツ変換をテーマにROOTでC/C++ベース
で数値計算し、粒子識別の性能評価っぽいことまで実施した。

検出器応答に見立てて乱数を振り、粒子識別性能を満たすため
に必要な分解能値を決定した。

TTree

TTree とは？

変数データなどをひとまとめにして管理するクラス。

- ・「イベント」データの処理・保存に特化している。
- ・膨大な量のデータを扱うのに適している。
- ・さまざまな型が保存できる。int, float, double, char, string, …
- ・表（テーブル）に似ている。

例題3) π^+ と K^+ の到達時間の分布を

TTreeで生成してみよう。

とりあえず、TTreeを確認しよう

```
$ls  
exam3.root include src
```

```
$root -l exam3.root  
root [0]  
Attaching file exam3.root as _file0...  
(TFile *) 0x7ffc7ed04080  
root [1] .ls  
TFile**          exam3.root  
TFile*           exam3.root  
KEY: TTree       tree;1  tree
```



これがTTreeです。

```
root [2] tree->Print()
```

```
*****
*Tree :tree      : tree
*Entries : 100000 : Total =          2912121 bytes  File Size = 1756444 *
* :           : Tree compression factor = 1.63
*****
*Br 0 :event    : event/I
*Entries : 100000 : Total Size= 401617 bytes  File Size = 3156 *
*Baskets : 12    : Basket Size= 32000 bytes  Compression= 121.66 *
*.....
*Br 1 :particle : particle/C
*Entries : 100000 : Total Size= 904613 bytes  File Size = 239507 *
*Baskets : 40    : Basket Size= 32000 bytes  Compression= 3.71 *
*.....
*Br 2 :mom      : mom/D
*Entries : 100000 : Total Size= 802811 bytes  File Size = 741138 *
*Baskets : 25    : Basket Size= 32000 bytes  Compression= 1.08 *
*.....
*Br 3 :dt       : dt/D
*Entries : 100000 : Total Size= 802780 bytes  File Size = 772643 *
*Baskets : 25    : Basket Size= 32000 bytes  Compression= 1.04 *
*.....
```

```
root [1] tree->Scan("event:particle:mom:dt")
*****
*   Row   *   event *   particle *   mom   *   dt   *
*****
*   0   *   0   *   kaon+ * 1575.7125 * 4.8148843 *
*   1   *   1   *   kaon+ * 3600.5643 * 0.8966494 *
*   2   *   2   *   kaon+ * 3312.1841 * 0.6176421 *
*   3   *   3   *   pi+  * 2657.1373 * -0.393070 *
*   4   *   4   *   pi+  * 1683.3579 * -0.022932 *
*   5   *   5   *   kaon+ * 3339.7782 * 0.4767434 *
*   6   *   6   *   pi+  * 3267.7656 * -0.158584 *
*   7   *   7   *   kaon+ * 1347.3026 * 7.4939353 *
*   8   *   8   *   pi+  * 2756.2997 * 0.8983353 *
*   9   *   9   *   pi+  * 2923.4234 * 0.7230007 *
*  10  *  10  *   pi+  * 2510.4630 * 0.0075913 *
*  11  *  11  *   pi+  * 2537.0715 * 0.1568643 *
*  12  *  12  *   pi+  * 2788.2557 * -0.535496 *
*  13  *  13  *   pi+  * 3195.7657 * -0.028305 *
*  14  *  14  *   pi+  * 1309.4244 * -0.442897 *
*  15  *  15  *   pi+  * 3910.2850 * 0.2120104 *
*  16  *  16  *   pi+  * 3171.3856 * -0.202977 *
*  17  *  17  *   kaon+ * 2745.4218 * 1.1466102 *
*  18  *  18  *   pi+  * 1424.5546 * 1.3787472 *
*  19  *  19  *   kaon+ * 2510.0164 * 1.8259434 *
*  20  *  20  *   kaon+ * 3409.1235 * 0.6063273 *
*  21  *  21  *   kaon+ * 1432.5818 * 5.8870849 *
*  22  *  22  *   kaon+ * 1673.8621 * 4.516347 *
*  23  *  23  *   pi+  * 2701.5432 * 0.3137436 *
*  24  *  24  *   kaon+ * 4449.7484 * 1.1053038 *
Type <CR> to continue or q to quit ==>
```

事象ごとに、パラメータが格納されていることが確認できます。

C言語でいうところの構造体が近いでしょうか。

こうすると、
それぞれの相関関係をみる
カット条件を指定

...

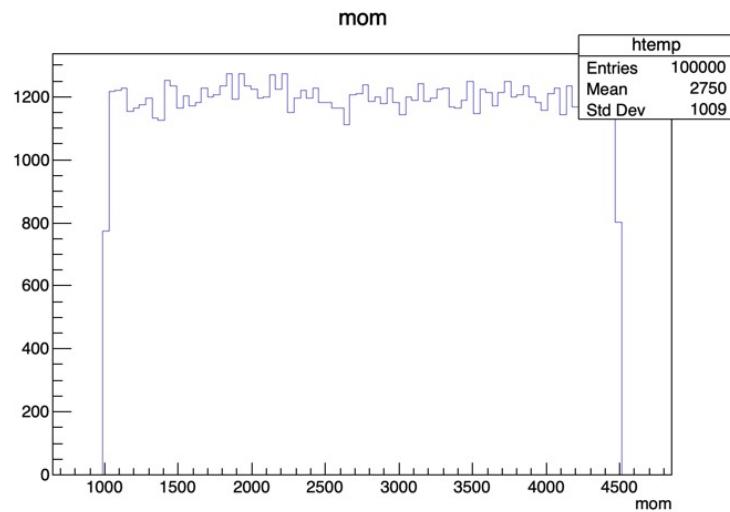
などやりたい放題できます！

[Enter]を押すと、続きを読むできて、
[q]を押すと終了します。

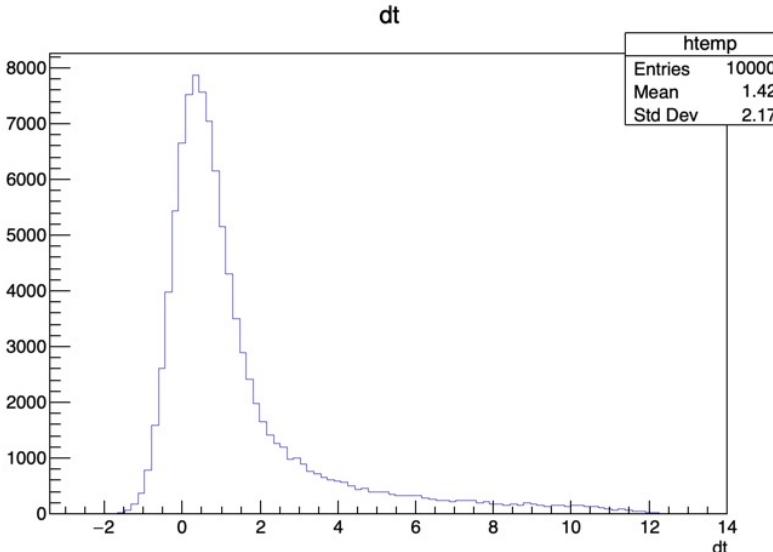
各パラメータをヒストグラムで表示してみましょう。

Y軸
X軸

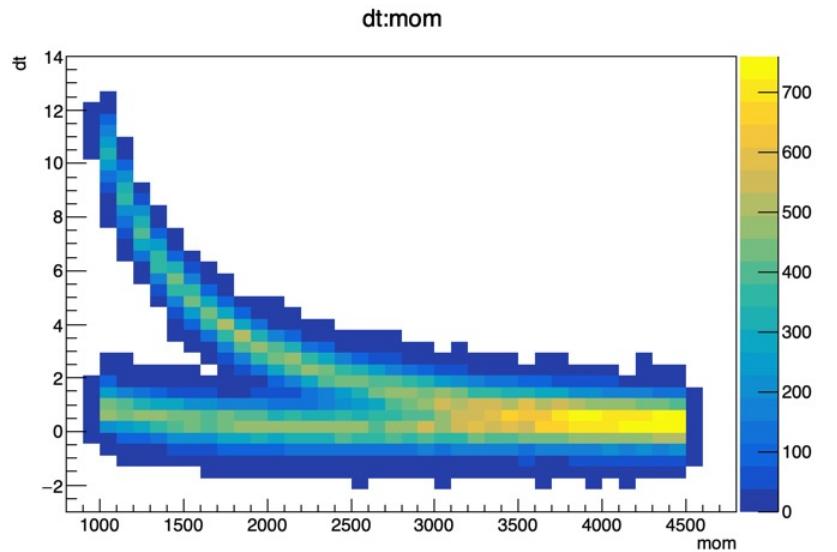
ROOT[2] tree->Draw("mom")



ROOT[3] tree->Draw("dt")



ROOT[4] tree->Draw("dt:mom","","","colz")



それぞれのブランチに格納されているパラメータを
1次元、2次元ヒストグラムとして確認できます

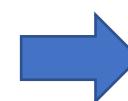
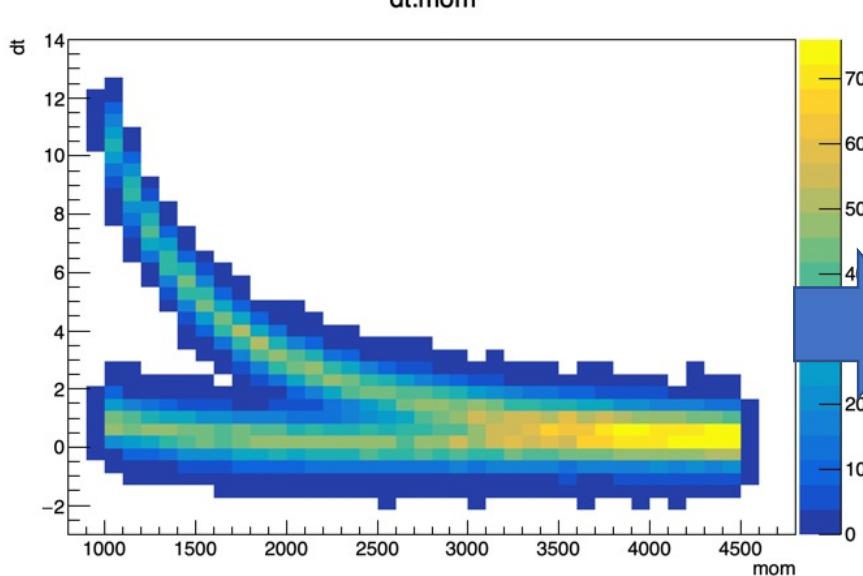
TTreeでできること

```
[1] tree->Draw("dt:mom","","","colz")
```

```
[2] tree->Draw("dt:mom>>h(100,1000,4500,100,-2,14)","","","","colz")
```

```
[3] tree->Draw("dt:mom>>h(1000,1000,4500,1000,-2,14)","","","","colz")
```

TTreeは生に近いデータを扱うので、このように
細かく分布を見るのも朝飯前♪

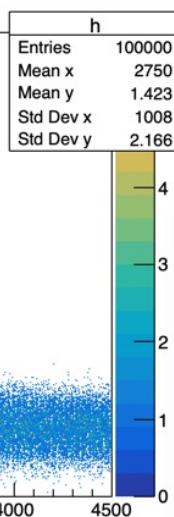


時間がある人は、
例題2をより細かく解析してみよう！

dt:mom

h	0
Entries	100000
Mean x	2750
Mean y	1.423
Std Dev x	1008
Std Dev y	2.166

dt:mom



ROOT[1].q

でROOTを終えます

例題3) π^+ と K^+ の到達時間の分布を
TTreeで生成してみよう。

TTreeを生成してみよう。
exam3.cxxのソースの中を、みて
きましょう。

```
1 inline void exam3();                                src/exam3.cxx
2
3 const double m_pion=139.6; // [MeV/c2]
4 const double m_kaon=493.7; // [MeV/c2]
5
6 // timing resolution [defo: 500 ps]
7 const double time_resolution = 500e-12;
8
9 // momentum resolution [defo: 5 MeV/c]
10 const double mom_resolution = 5;
11 const double L = 30.0; // [m]
12 const double numevents = 1e5;
13 const double c = 2.99792458 * 1e8; // m/s
14
15 const int verbose = 0;
```

```
17 void exam3(){
18
19     char particle[100];
20     int event, sw;
21     double mom,mom_true, beta, dt, t0, t1, m;
22
23     auto file = new TFile("exam3.root","recreate");
24     auto tree = new TTree("tree","tree");
25     tree->Branch("event",&event,"event/I");
26     tree->Branch("particle",&particle,"particle/C");
27     tree->Branch("mom",&mom,"mom/D");
28     tree->Branch("dt",&dt,"dt/D");
29 }
```

auto file=new TFile("ファイル名", "option");

TFileの2番目の引数に"recreate"を入れると、
指定したファイルは、真っ白になって作り替えられます。

既存のデータを指定した場合、消えるので注意！

普通に読むだけなら2番目の引数は"なし"

tree->Branch()の引数は

1: [Br名]、2: 変数、3: Br名/型の頭文字
* floatならFです。

例題3) π^+ と K^+ の到達時間の分布を
TTree で生成してみよう。

```
30     for(int j=0;j<numevents;j++){           src/exam3.cxx
31         event = j;
32         sw = (int) gRandom->Uniform(0,2);
33
34         mom_true = gRandom->Uniform(1000, 4500);
35         mom = gRandom->Gaus(mom_true, mom_resolution);
36         if(verbose)
37             cout<<"momentum ="<<mom<<" MeV/c; ";
38
```

L32では、0 or 1がランダムに出るようにして
0なら π^+
1なら K^+
の名前と質量を入れてますね。

```
39     // - pion
40     if(sw==0){
41         sprintf(particle,"pi+");
42         m=m_pion;
43     }
44     // - kaon
45     else if(sw==1){
46         sprintf(particle,"kaon+");
47         m=m_kaon;
48     }
49     beta = 1./sqrt(1. + pow(m/mom_true,2));
50     t0 = L/c;
51     t1 = L / beta / c;
52     dt = t1 - t0;
53     dt = gRandom->Gaus(dt, time_resolution);
54     dt *=1e9;
55     if(verbose)
56         cout<<"pion: "<<dt<<" ns; ";
57
58     tree->Fill();
59 }
60
61 return;
62 }
63
```

あとの計算は共通で、例題2と同じことをします。

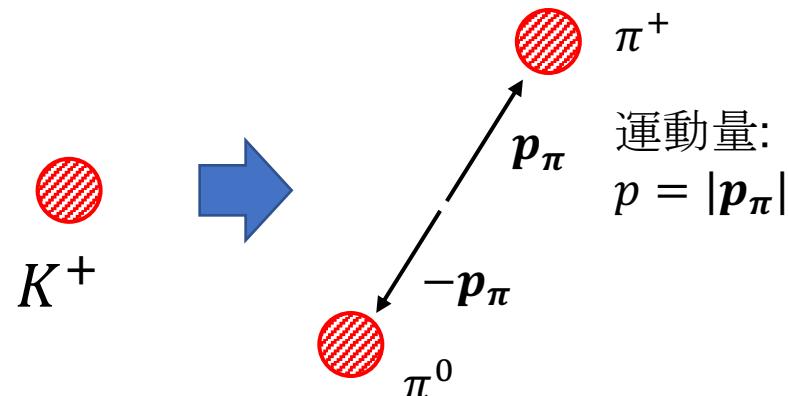
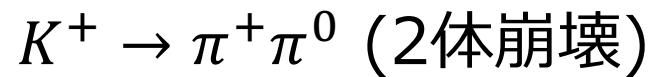
ここまでで習得したこと

- Branchに格納されているパラメータを、ヒストグラムで確認する。
- 簡易的なシミュレーションで生成したパラメータ群をTTreeに保存する。

Rootファイルを見つけたら、`root -l **.root`でひらいて、
TTreeがあれば、`Print()`で中身を確認しましょう。

次は、実践的な課題を通して、TTreeに慣れ親しみましょう。

例題4) 粒子の崩壊を疑似的に再現してみよう



K^+ 静止系を考えましょう。

K^+ の質量 $m_{K^+} = 493.7 \text{ MeV}/c^2$

π^+ の質量 $m_{\pi^+} = 139.6 \text{ MeV}/c^2$

π^0 の質量 $m_{\pi^0} = 134.9 \text{ MeV}/c^2$

エネルギー保存則と運動量保存則より

$$m_{K^+} = E_{\pi^+} + E_{\pi^0} \quad \cdots \textcircled{1}$$

$$\mathbf{p}_{K^+} = \mathbf{p}_{\pi^+} + \mathbf{p}_{\pi^0} = \mathbf{0} \quad \cdots \textcircled{2}$$

$$\textcircled{2} \text{ より、 } p = |\mathbf{p}_{\pi^+}| = |\mathbf{p}_{\pi^0}|$$

$$\textcircled{1} \text{ より、 } m_{K^+} = \sqrt{p^2 + m_{\pi^+}^2} + \sqrt{p^2 + m_{\pi^0}^2}$$

これを計算進めていくと、

$$\begin{aligned} p &= \sqrt{\left(\frac{m_{K^+}^2 - m_{\pi^+}^2 + m_{\pi^0}^2}{2m_{K^+}}\right)^2 - m_{\pi^0}^2} \\ &= 205.4 \text{ MeV}/c \end{aligned}$$

例題4)粒子の崩壊を疑似的に再現してみよう

```
$root -l src/exam4_0.cxx
```

```
root [0]
Processing src/exam4_0.cxx...
K->pi+ pi0 decay: p_pi+=205.417 MeV/c, p_pi0=205.417 MeV/c
```

```
1 #include "../include/twobodydecay.hh"
2 void exam4_0(){
3     double mK=493.677, mpi=139.6,mpi0=134.1;
4     double pK=0, pe=0, pnu=0, pmu=0, ppi=0, ppi0;
5
6     twobodydecay(mK, mpi, ppi, mpi0, ppi0);
7     cout<<"K->pi+ pi0 decay: p_pi+="<<ppi<<" MeV/c, p_pi0="<<ppi0<<" MeV/c"<<endl;
8     return;
9 }
```

2体崩壊の計算

$$p = \sqrt{\left(\frac{m_{K^+}^2 - m_{\pi^+}^2 + m_{\pi^0}^2}{2m_{K^+}}\right)^2 - m_{\pi^0}^2}$$

のコードが入っているソースファイル。

2体崩壊の計算をしている。

- mK, mpi, mpi0がインプット
- ppi, ppi0がアウトプット

src/exam4_0.cxx

K^+ 静止系でppi+, ppi0どちらも、205.4 MeV/cと正しく計算されていることが確認できました。

例題4)粒子の崩壊を疑似的に再現してみよう

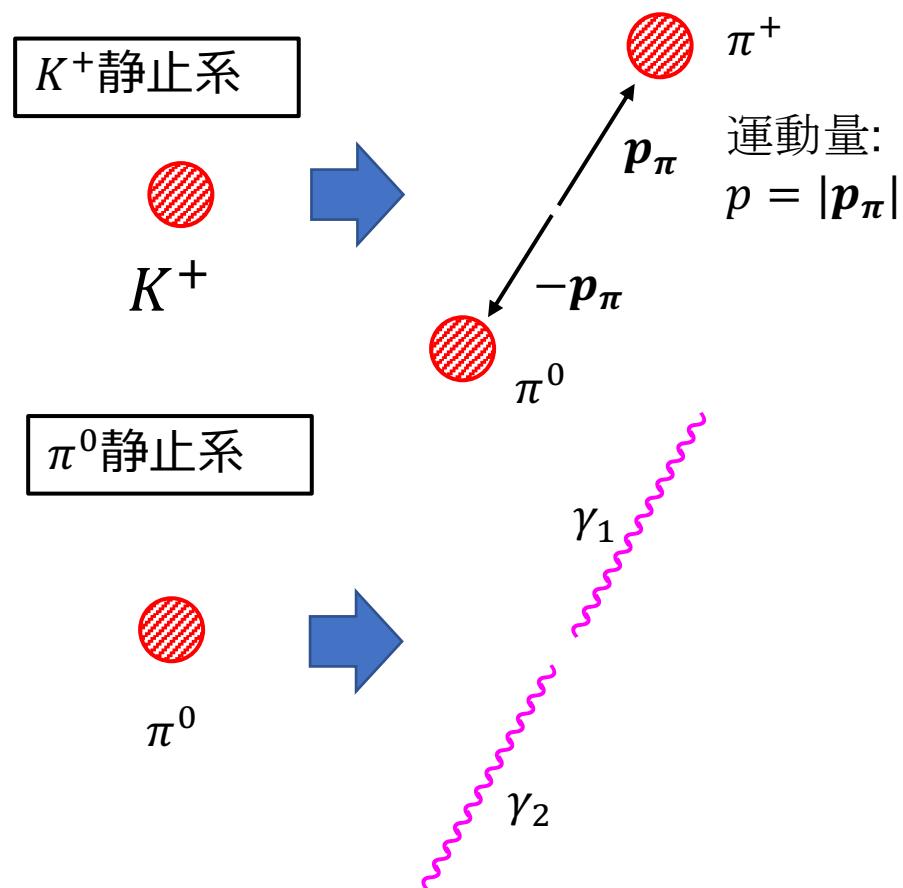
親粒子のmass

```
1 void twobodydecay(double parentM, ←  
2           double &daughterM1, double &daughterP1,  
3           double &daughterM2, double &daughterP2  
4 {   娘粒子のmass1,2      娘粒子の運動量1,2  
5     daughterP1 = sqrt( pow( (pow(parentM, 2)  
6                           - pow(daughterM1, 2)  
7                           + pow(daughterM2, 2))  
8                           /(2*parentM), 2)  
9                           -pow(daughterM2, 2));  
10    daughterP2 = daughterP1;  
11    return;  
12 }
```

include/twobodydecay.hh

例題4) 粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。



π^0 は205.4 MeV/cで走っているので、崩壊した $\gamma\gamma$ もその方向にローレンツブーストされている。

1. まず、 π^0 の速度比を計算しよう。
2. β を使って、ローレンツ変換して $\gamma\gamma$ の運動量に作用させます。
3. $\pi^+\pi^0$ と $\gamma\gamma$ の放射角度を一様乱数で生成します。

例題4)粒子の崩壊を疑似的に再現してみよう

src/exam4_1.cxx

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

1. π^0 の速度比 β を計算しよう。
z方向に π^0 が進んだと仮定します。

\$root -l src/exam4_1.cxx

```
root [0]
Processing src/exam4_1.cxx...
beta = 0.837363
gamma = 1.82934
```

```
1 #include "../include/twobodydecay.hh"
2 void exam4_1(){
3     double mK=493.677, mpi=139.6,mpi0=134.1;
4     double pK=0, pe=0,pnu=0, pmu=0, ppi=0, ppi0=0, p_pi[3]={0}, p_pi0[3]={0};
5     double Epi, Eg1, pg1, Eg2, pg2, p_g1[3]={0}, p_g2[3]={0};
6     double beta, gamma;
7
8     // kaon decay to 2 pions at kaon rest system
9     twobodydecay(mK, mpi, ppi, mpi0, ppi0);
10    Epi=sqrt(pow(mpi0,2) + pow(ppi0,2));
11
12    // -- set pi0 momentum
13    p_pi0[0]=0;
14    p_pi0[1]=0;
15    p_pi0[2]=ppi; // <= assumption p_z = 205.4 MeV/c;
16
17    beta = ppi / Epi;
18    gamma=1./sqrt(1.-pow(beta,2));
19    cout<<"beta = "<<beta<<endl;
20    cout<<"gamma = "<<gamma<<endl;
21
22    return;
23 }
```

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

2. β を使って、ローレンツ変換し
て $\gamma\gamma$ の運動量に作用させます。

```
$root -l src/exam4_2.cxx
```

```
root [0]
Processing src/exam4_2.cxx...
gamma 1:
    Energy = 109.509 MeV
    momentum vec = (61.0428, 26.3782, 87.0071) NeV/c
gamma 2:
    Energy = 135.805 MeV
    momentum vec = (-61.0428, -26.3782, 118.41) NeV/c
```

Z方向にローレンツブーストされ
ていることがわかりますね。

```
28 // randomize emission direction
29 costheta=gRandom->Uniform(-1.,1.);
30 phi=2*TMath::Pi()*gRandom->Uniform(0,1);
31 sintheta=sqrt(1.-pow(costheta,2));
```

```
33 p_g1[0]=pg1*sintheta*cos(phi);
34 p_g1[1]=pg1*sintheta*sin(phi);
35 p_g1[2]=pg1*cosheta;
36
37 p_g2[0]=-pg2*sintheta*cos(phi);
38 p_g2[1]=-pg2*sintheta*sin(phi);
39 p_g2[2]=-pg2*cosheta;
40
41 // Lorentz boost
42 p_g1[2] = gamma*beta*Eg1 + gamma*p_g1[2];
43 p_g2[2] = gamma*beta*Eg2 + gamma*p_g2[2];
44
```

```
45 Eg1 = 0;
46 for(int j=0;j<3;j++) Eg1+=pow(p_g1[j],2);
47 Eg1=sqrt(Eg1);
48 pg1=Eg1;
49 Eg2 = 0;
50 for(int j=0;j<3;j++) Eg2+=pow(p_g2[j],2);
51 Eg2=sqrt(Eg2);
52 pg2=Eg2;
53
54 cout<<"gamma 1:"<<endl
55     <<"\t Energy = "<<Eg1<<" MeV"<<endl
56     <<"\t momentum vec = ("<<p_g1[0]<<", "<<p_g1[1]<<", "<<p_g1[2]<<") NeV/c"<<endl;
57
58 cout<<"gamma 2:"<<endl
59     <<"\t Energy = "<<Eg2<<" MeV"<<endl
60     <<"\t momentum vec = ("<<p_g2[0]<<", "<<p_g2[1]<<", "<<p_g2[2]<<") NeV/c"<<endl;
61
62 | return;
63 }
```

立体角一様乱数を
生成します

src/exam4_2.cxx

$$\begin{pmatrix} E' \\ p' \end{pmatrix} = \begin{pmatrix} \gamma & \gamma\beta \\ \gamma\beta & \gamma \end{pmatrix} \begin{pmatrix} E \\ p \end{pmatrix}$$

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

3. $\pi^+\pi^0$ と $\gamma\gamma$ の放射角度を一様乱数で生成します。

solidangleuniform()

立体角で一様に乱数を生成する

include/solidangleuniform.hh

```
1 void solidangleuniform(double &costheta,
2                         double &phi,
3                         double &sinttheta){
4     costheta=gRandom->Uniform(-1.,1.);
5     phi=2*TMath::Pi()*gRandom->Uniform(0,1);
6     sinttheta=sqrt(1.-pow(costheta,2));
7     return;
8 }
```

```
1 #include "../include/twobodydecay.hh"
2 #include "../include/solidangleuniform.hh"
3 #include "../include/lorentz.hh"
4 void exam4_3(){
5     double mK=493.677, mpi=139.6, mpi0=134.1, mg1=0., mg2=0.;
6     double pK=0, pe=0, pnu=0, pmu=0, ppi=0, ppi0=0, p_pi[3]={0}, p_pi0[3]={0};
7     double Epi, Eg1, pg1, Eg2, pg2, p_g1[3]={0}, p_g2[3]={0};
8     double beta[3];
9     double costheta1, sinttheta1, phi1;
10    double costheta2, sinttheta2, phi2;
11    double total_mom[3]={0};
12
13    for(int event = 0; event < 10; event++){
14
15        cout<<"======"<<endl;
16        cout<<"Event "<<event<<endl;
17
18        // kaon decay to 2 pions at kaon rest system
19        twobodydecay(mK, mpi, ppi, mpi0, ppi0);
20        solidangleuniform(costheta1, phi1, sinttheta1);
21
22        p_pi[0]=ppi*sinttheta1*cos(phi1);
23        p_pi[1]=ppi*sinttheta1*cos(phi1);
24        p_pi[2]=ppi*cos(theta1);
25
26        Epi=sqrt(pow(mpi,2)+pow(p_pi[0],2)+pow(p_pi[1],2)+pow(p_pi[2],2));
27        cout<<"pi+"<<endl
28        <<"\t Energy = "<<Epi<<" MeV"<<endl
29        <<"\t Momentum = "<<ppi<<" MeV/c"<<endl
30        <<"\t Momentum vec = ("<<p_pi[0]<<, "<<p_pi[1]<<, "<<p_pi[2]<<") NeV/c"<<endl;
31
32        p_pi0[0]=-ppi0*sinttheta1*cos(phi1);
33        p_pi0[1]=-ppi0*sinttheta1*sin(phi1);
34        p_pi0[2]=-ppi0*cos(theta1);
35
36        //beta = ppi / Epi;
37        for(int i=0;i<3;i++)
38            beta[i] = p_pi[i] / Epi;
```

src/exam4_3.cxx

例題4) 粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

3. $\pi^+\pi^0$ と $\gamma\gamma$ の放射角度を一様乱数で生成します。

solidangleuniform()

立体角で一様に乱数を生成する

lorentz()

ローレンツ変換を実施

引数1: エネルギー

引数2,3,4: 運動量ベクトル

引数5,6,7: 速度比ベクトル

このコマンドで $Eg1$, p_g1 が、書き変わる。

```
1 #include "../include/twobodydecay.hh"
2 #include "../include/solidangleuniform.hh"
3 #include "../include/lorentz.hh"
4 void exam4_3(){
5     double mk=493.677, mpi=139.6, mpi0=134.1, mg1=0., mg2=0.;
6     double pK=0, pe=0, pnu=0, pmu=0, ppi=0, ppi0=0, p_pi[3]={0}, p_pi0[3]={0};
```

ローレンツ変換するコード

```
53     p_g2[2]=-pg2*cosheta2;
54
55     // Lorentz boost
56     lorentz(Eg1, p_g1[0],p_g1[1],p_g1[2], beta[0],beta[1],beta[2]);
57     lorentz(Eg2, p_g2[0],p_g2[1],p_g2[2], beta[0],beta[1],beta[2]);
58
59     Eg1 = 0;
60     for(int j=0;j<3;j++) Eg1+=pow(p_g1[j],2);
61     Eg1=sqrt(Eg1);
62     pg1=Eg1;
63     Eg2 = 0;
64     for(int j=0;j<3;j++) Eg2+=pow(p_g2[j],2);
65     Eg2=sqrt(Eg2);
66     pg2=Eg2;
67
68     cout<<"gamma 1:"<<endl
69     <<"\t Energy = "<<Eg1<<" MeV"<<endl
70     <<"\t Momentum = "<<pg1<<" MeV"<<endl
71     <<"\t Momentum = vec ("<<p_g1[0]<<", "<<p_g1[1]<<", "<<p_g1[2]<<") MeV/c"<<endl;
72
73     cout<<"gamma 2:"<<endl
74     <<"\t Energy = "<<Eg2<<" MeV"<<endl
75     <<"\t Momentum = "<<pg2<<" MeV"<<endl
76     <<"\t Momentum = vec ("<<p_g2[0]<<", "<<p_g2[1]<<", "<<p_g2[2]<<") MeV/c"<<endl;
77
78 }
79
80 return;
81 }
```

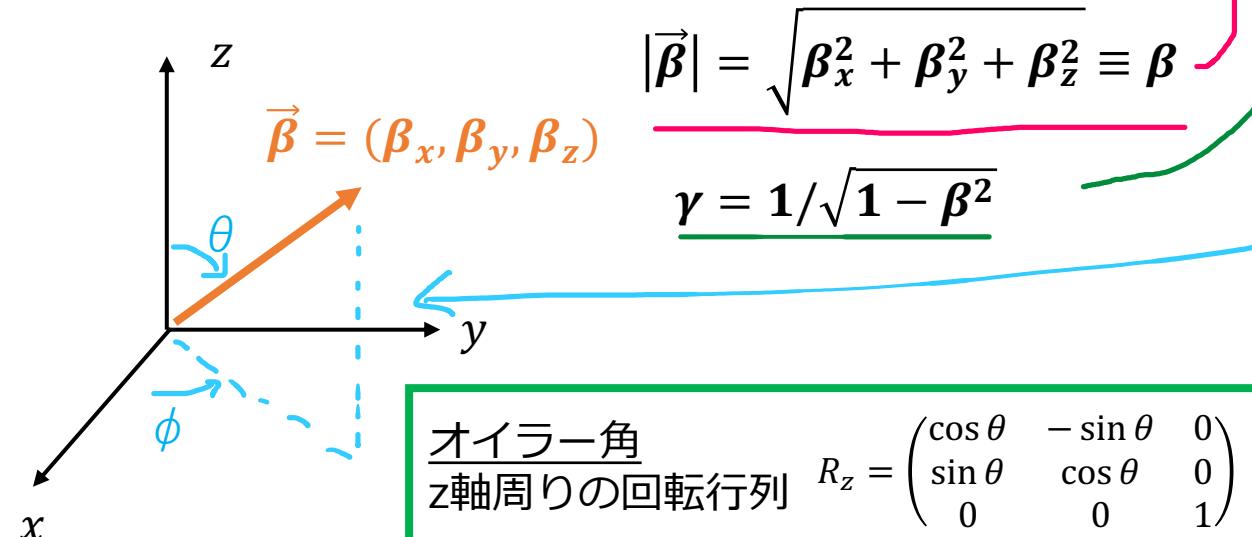
src/exam4_3.cxx

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

include/lorentz.hh

$$\begin{pmatrix} E' \\ p' \end{pmatrix} = \begin{pmatrix} \gamma & \gamma\beta \\ \gamma\beta & \gamma \end{pmatrix} \begin{pmatrix} E \\ p \end{pmatrix}$$



```

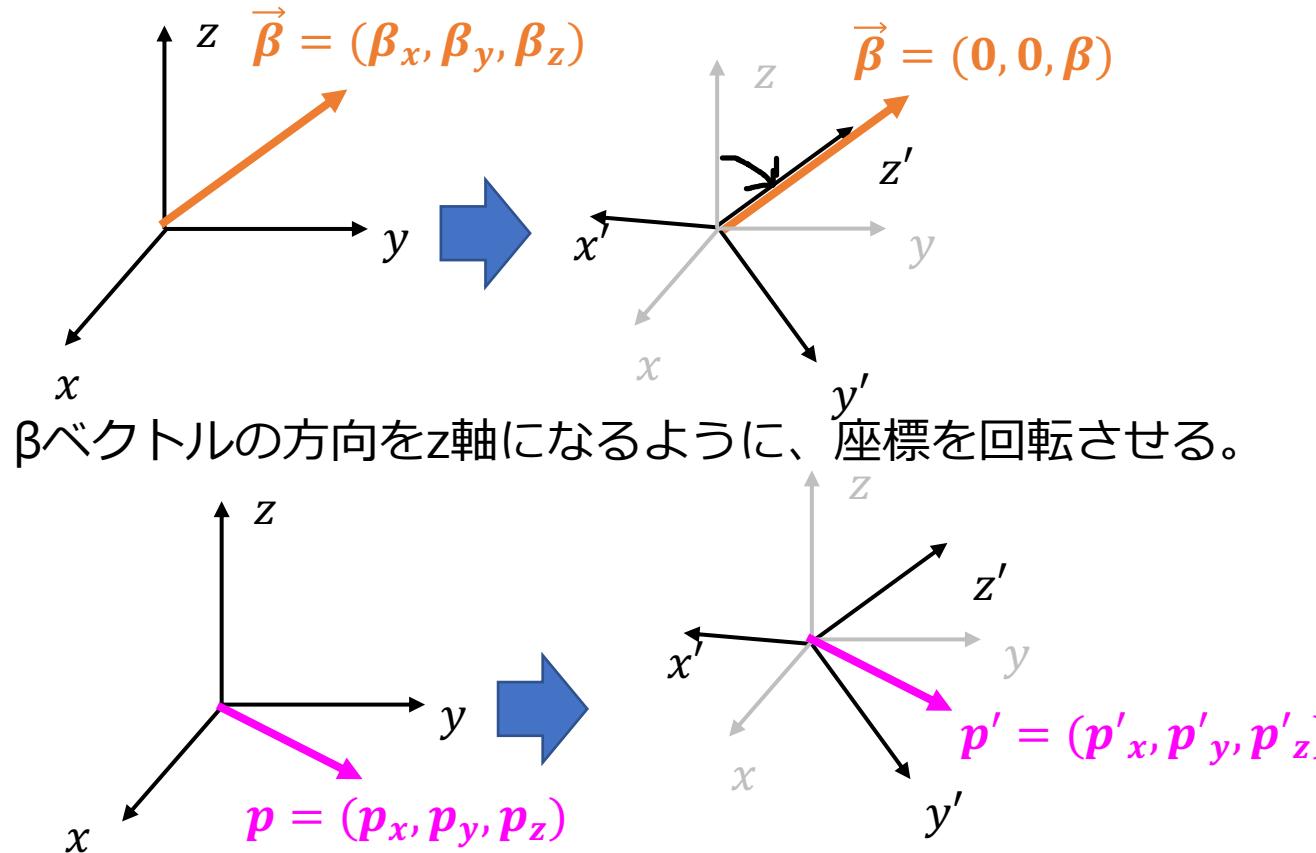
9 // -- calc absolute beta from beta vector
10 double absbeta = 0;
11 for(int i=0;i<3;i++)
12     absbeta += pow(beta[i],2);
13 absbeta = sqrt(absbeta);
14
15 // -- calc gamma from beta
16 double gamma = 1. / sqrt(1. - absbeta*absbeta);
17
18 // -- boost vector (solid angle) calculate
19 double costheta, sintheta, cosphi, sinphi, phi, theta;
20 costheta = beta[2] / absbeta;
21 sintheta = sqrt(1.-pow(costheta,2));
22 if(sintheta>=0) theta = acos(costheta);
23 else theta = -acos(costheta) + 2*TMath::Pi();
24
25 if(costheta==1){
26     cosphi=1; sinphi=0;
27     phi=0;
28 }
29 else{
30     cosphi = beta[0]/absbeta/sintheta;
31     sinphi = beta[1]/absbeta/sintheta;
32     if(sinphi>=0) phi = acos(cosphi);
33     else phi = -acos(cosphi) + 2*TMath::Pi();
34 }
35
36 // -- Euler angle matrix definition
37 double Ry[3][3] = {{ costheta,          0,   sintheta},
38                     {          0,          1,          0},
39                     { -sintheta,         0,  costheta}};
40 double invRy[3][3] = {{costheta,          0, -sintheta},
41                     {          0,          1,          0},
42                     {sintheta,         0,  costheta}};
43 double Rz[3][3] = {{cosphi, -sinphi, 0},
44                     {sinphi,  cosphi, 0},
45                     {          0,          0,  1}};
46 double invRz[3][3] = {{cosphi, sinphi, 0},
47                     {-sinphi, cosphi, 0},
48                     {          0,          0,  1}};
49

```

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+ \pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+ \gamma_1 \gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

include/lorentz.hh



```

50 // transformation
51 // S' -> S: (x) = Ry^-1 Rz^-1 (x')
52 double dmy[3]={0}, mom_[3]={0};
53 for(int i=0;i<3;i++){
54     for(int j=0;j<3;j++){
55         dmy[i] += invRz[i][j] * mom[j];
56     }
57 }
58 for(int i=0;i<3;i++){
59     for(int j=0;j<3;j++){
60         mom_[i] += invRy[i][j] * dmy[j];
61     }
62 }

63 // boost at z-direction in beta-direction-system
64 mom_[2] = gamma*absbeta*Ene + gamma*mom_[2];

65 // S -> S': (x') = Rz Ry (x)
66 for(int i=0;i<3;i++){
67     dmy[i]=0;
68     mom[i]=0;
69 }
70 for(int i=0;i<3;i++){
71     for(int j=0;j<3;j++){
72         dmy[i] += Ry[i][j] * mom_[j];
73     }
74 }
75 for(int i=0;i<3;i++){
76     for(int j=0;j<3;j++){
77         mom[i] += Rz[i][j] * dmy[j];
78     }
79 }
80 }
81 }

82

```

include/lorentz.hh

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+ \pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+ \gamma_1 \gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

include/lorentz.hh

1. β ベクトル方向をz軸になるように座標回転

$$(x') = (R_y)^{-1} (R_z)^{-1} (x)$$

2. その回転座標系におけるガンマ線方向のz成分をローレンツブースト

$$p''_z = \gamma \beta p'_z + \gamma E'$$

3. その後、元の座標系に戻す。

$$(x) = R_z R_y (x')$$

The diagram illustrates the flow of code execution. It starts with a green arrow pointing from the first section of the code (lines 50-62) down to the second section (lines 64-82). From there, another green arrow points left to the third section (line 84). Finally, a green arrow points left again to the fourth section (line 86), which is enclosed in a blue box.

```
50 // transformation
51 // S' -> S: (x) = Ry^-1 Rz^-1 (x')
52 double dmy[3]={0}, mom_[3]={0};
53 for(int i=0;i<3;i++){
54     for(int j=0;j<3;j++){
55         dmy[i] += invRz[i][j] * mom[j];
56     }
57 }
58 for(int i=0;i<3;i++){
59     for(int j=0;j<3;j++){
60         mom_[i] += invRy[i][j] * dmy[j];
61     }
62 }
63
64 // boost at z-direction in beta-direction-system
65 mom_[2] = gamma*absbeta*Ene + gamma*mom_[2];
66
67 // S -> S': (x') = Rz Ry (x)
68 for(int i=0;i<3;i++){
69     dmy[i]=0;
70     mom[i]=0;
71 }
72 for(int i=0;i<3;i++){
73     for(int j=0;j<3;j++){
74         dmy[i] += Ry[i][j] * mom_[j];
75     }
76 }
77 for(int i=0;i<3;i++){
78     for(int j=0;j<3;j++){
79         mom[i] += Rz[i][j] * dmy[j];
80     }
81 }
82 }
```

include/lorentz.hh

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+\gamma_1\gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

3. $\pi^+\pi^0$ と $\gamma\gamma$ の放射角度を一様乱
数で生成します。

```
$root -l src/exam4_3.cxx
```

10イベント擬似サンプルが
計算されています。

```
root [0]
Processing src/exam4_3.cxx...
=====
Event 0
pi+:
    Energy = 248.363 MeV
    Momentum = 205.417 MeV/c
    Momentum vec = (5.63742, 3.43468, 205.311) NeV/c
gamma 1:
    Energy = 75.5738 MeV
    Momentum = 75.5738 MeV
    Momentum = vec (-17.2403, 57.1672, 46.3258) MeV/c
gamma 2:
    Energy = 163.004 MeV
    Momentum = 163.004 MeV
    Momentum = vec (23.5783, -56.9634, 150.896) MeV/c
=====
Event 1
pi+:
    Energy = 248.363 MeV
    Momentum = 205.417 MeV/c
    Momentum vec = (16.3401, -172.555, -110.245) NeV/c
gamma 1:
    Energy = 87.8466 MeV
    Momentum = 87.8466 MeV
    Momentum = vec (-64.181, 57.4232, 17.3319) MeV/c
gamma 2:
    Energy = 150.731 MeV
    Momentum = 150.731 MeV
    Momentum = vec (-25.1764, 83.0648, -123.233) MeV/c
=====
Event 2
pi+:
    Energy = 248.363 MeV
    Momentum = 205.417 MeV/c
    Momentum vec = (-204.349, -12.9172, 16.4513) NeV/c
gamma 1:
    Energy = 72.2732 MeV
    Momentum = 72.2732 MeV
    Momentum = vec (-42.2065, 42.1917, 40.7664) MeV/c
gamma 2:
    Energy = 166.305 MeV
    Momentum = 166.305 MeV
    Momentum = vec (57.9589, 153.866, -24.9632) MeV/c
=====
Event 3
```

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+ \pi^0; \pi^0 \rightarrow \gamma\gamma$ を再現して、
終状態: $\pi^+ \gamma_1 \gamma_2$ のエネルギー、運動量ベクトル
の擬似データを生成してみよう。

さて、このデータを TTree に
格納して保存しましょう。

```
$root -l src/exam4_4.cxx
```

```
=====
Event 9999
pi+:
    Energy = 248.363 MeV
    Momentum = 205.417 MeV/c
    Momentum vec = (115.977, -169.536, 1.67199) NeV/c
gamma 1:
    Energy = 157.215 MeV
    Momentum = 157.215 MeV
    Momentum = vec (-51.2331, 145.694, -29.4095) NeV/c
gamma 2:
    Energy = 39.3217 MeV
    Momentum = 39.3217 MeV
    Momentum = vec (-20.7139, -17.468, 28.4955) NeV/c
[root [1] .ls
TFile**      exam4.root
TFile*       exam4.root
OBJ: TTree   tree   tree : 0 at: 0x7f860902c6a0
KEY: TTree   tree;1 tree
```

exam4.rootができましたね。

```
1 #include "../include/twobodydecay.hh"
2 #include "../include/solidangleuniform.hh"
3
4 const int total_number = 10000; 統計1万イベント回してみましょうか
5 void exam4_4(){
6     double mK=493.677, mpi=139.6, mpi0=134.1, mg1=0., mg2=0.;
7     double pK=0, pe=0, pnu=0, pmu=0, ppi=0, ppi0=0, p_pi[3]={0}, p_pi0[3]={0};
8     double Epi, Eg1, pg1, Eg2, pg2, p_g1[3]={0}, p_g2[3]={0};
9     double beta[3]={0}, gamma[3]={0};
10    double ctheta, sintheta, phi;
11    int event=0;
12
13    auto file=new TFile("exam4.root","recreate"); TFileの作成
14    auto tree=new TTree("tree","tree");
15    tree->Branch("event",&event,"event/I");
16    tree->Branch("Epi",&Epi,"Epi/D");
17    tree->Branch("ppi",&ppi,"ppi/D");
18    tree->Branch("p_pi[3]",&p_pi,"p_pi[3]/D");
19    tree->Branch("Eg1",&Eg1,"Eg1/D");
20    tree->Branch("pg1",&pg1,"pg1/D");
21    tree->Branch("p_g1[3]",&p_g1,"p_g1[3]/D");
22    tree->Branch("Eg2",&Eg2,"Eg2/D");
23    tree->Branch("pg2",&pg2,"pg2/D");
24    tree->Branch("p_g2[3]",&p_g2,"p_g2[3]/D");
25
26    for(event = 0; event < total_number; event++){
27
28        tree->Fill(); ← 每eventのループでtreeに、
29        }                   变数を詰める
30
31        file->Write(); ← TFileの保存
32        return;
33    }
```

src/exam4_4.cxx

TTree定義
Branchの設定

46

例題4) 粒子の崩壊を疑似的に再現してみよう

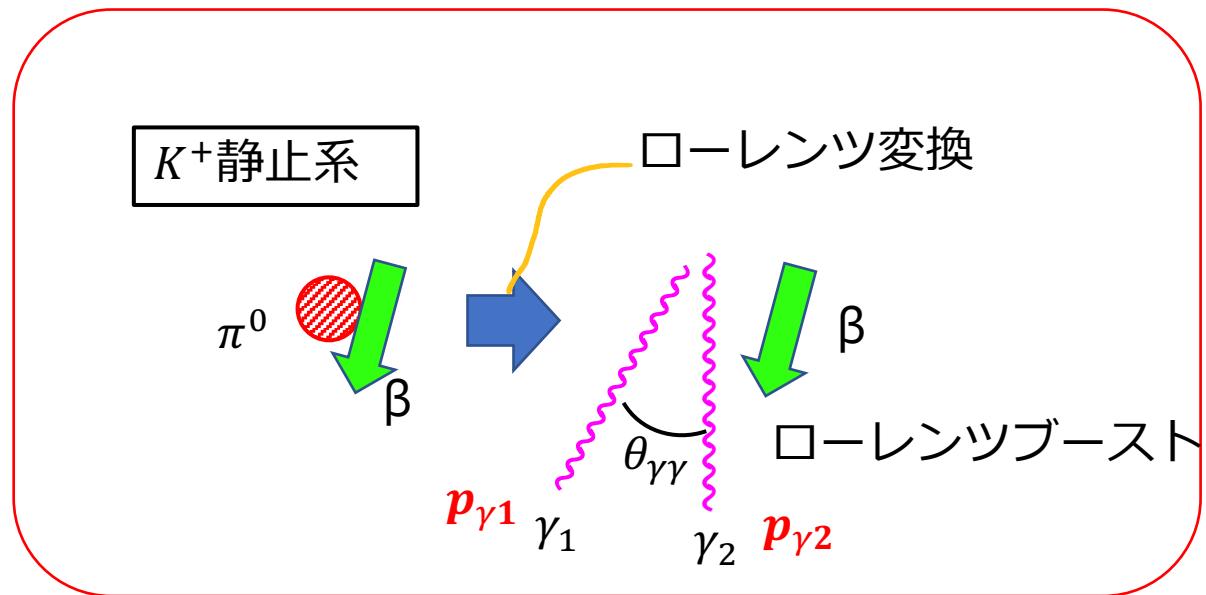
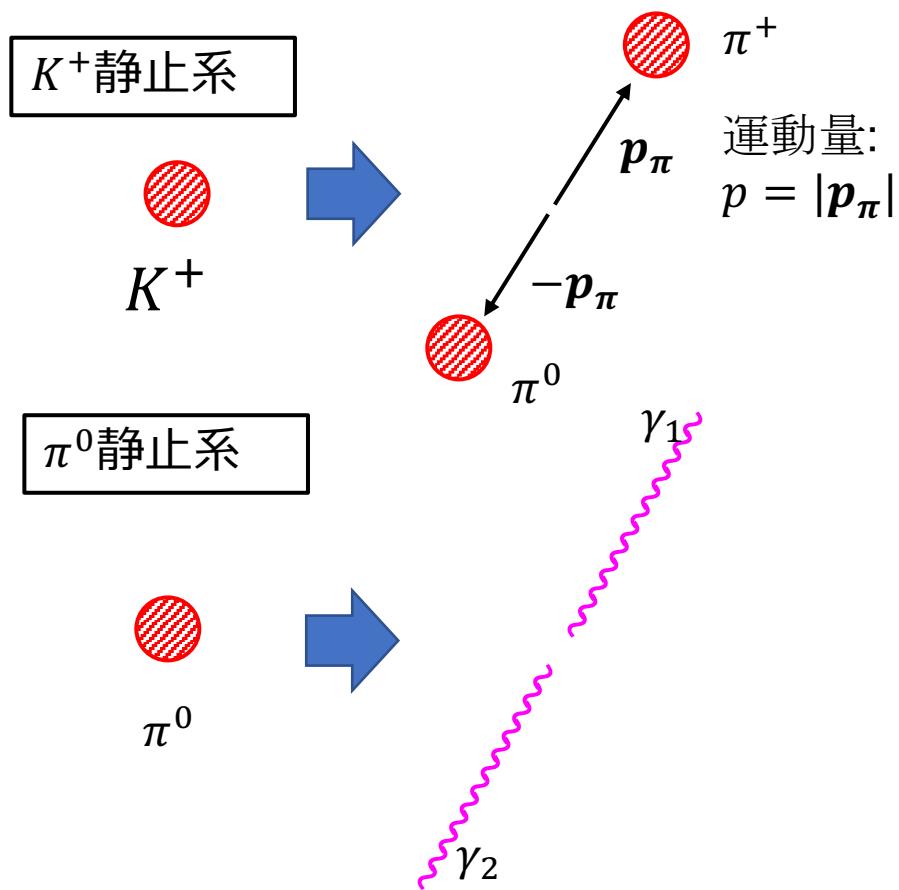
$K^+ \rightarrow \pi^+ \pi^0; \pi^0 \rightarrow \gamma\gamma$ の擬似データを解析しよう。

1. $\gamma_1\gamma_2$ の開口角 $\theta_{\gamma\gamma}$ の分布をつくってみよう。
2. π^0 のinvariant massをつくってみよう。

例題4) 粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+ \pi^0; \pi^0 \rightarrow \gamma\gamma$ の擬似データを解析しよう。

1. $\gamma_1 \gamma_2$ の開口角 $\theta_{\gamma\gamma}$ の分布をつくってみよう。



$$\cos \theta_{\gamma\gamma} = \frac{\mathbf{p}_{\gamma 1} \cdot \mathbf{p}_{\gamma 2}}{|\mathbf{p}_{\gamma 1}| |\mathbf{p}_{\gamma 2}|}$$

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ の擬似データを解析しよう。

1. $\gamma_1\gamma_2$ の開口角 $\theta_{\gamma\gamma}$ の分布をつくってみよう。

auto tree = (TTree*) file->Get("tree")

…TFileに保存されているtreeを呼び出し、ポインタ treeに登録する。

tree->SetBranchAddress()

…treeのBranchのパラメータを変数に格納できるようにする定義。

tree->GetEntry(i)

…このときに **SetBranchAddress()** で定義した変数に、各事象ごとに値が格納される。

開口角を計算している。

$$\cos \theta_{\gamma\gamma} = \frac{\mathbf{p}_{\gamma 1} \cdot \mathbf{p}_{\gamma 2}}{|\mathbf{p}_{\gamma 1}| |\mathbf{p}_{\gamma 2}|}$$

```

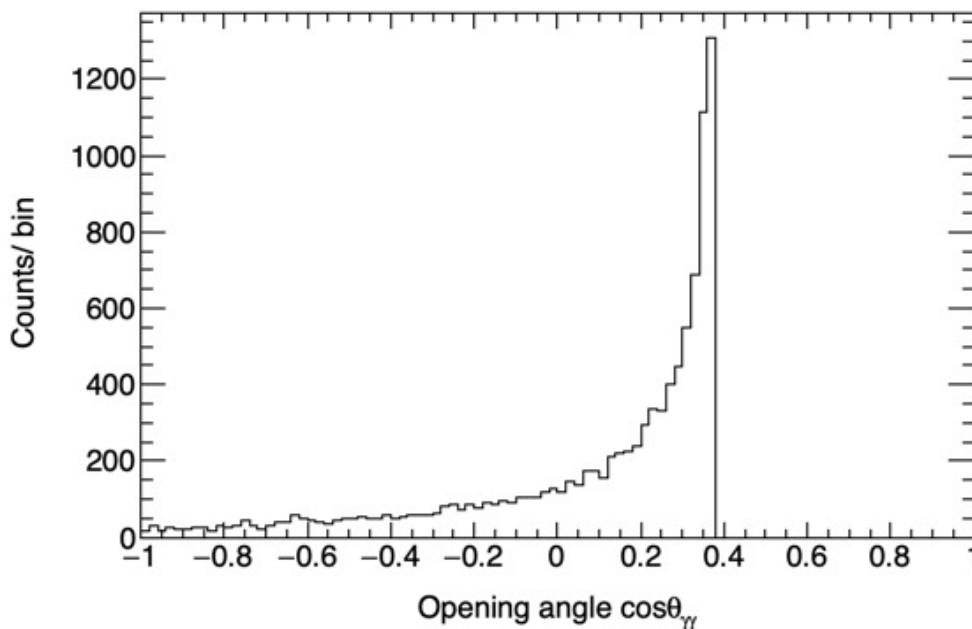
1 void exam4_5(){
2     int event;
3     double Epi, ppi, p_pi[3], Eg1, Eg2, pg1, pg2, p_g1[3], p_g2[3];
4     double costheta,inpro;
5
6
7     auto file = new TFile("exam4.root");
8     auto tree =(TTree*) file->Get("tree");
9     tree->SetBranchAddress("event",&event);
10    tree->SetBranchAddress("Epi",&Epi);
11    tree->SetBranchAddress("ppi",&ppi);
12    tree->SetBranchAddress("p_pi[3]",&p_pi);
13    tree->SetBranchAddress("Eg1",&Eg1);
14    tree->SetBranchAddress("pg1",&pg1);
15    tree->SetBranchAddress("p_g1[3]",&p_g1);
16    tree->SetBranchAddress("Eg2",&Eg2);
17    tree->SetBranchAddress("pg2",&pg2);
18    tree->SetBranchAddress("p_g2[3]",&p_g2);
19
20    gROOT->SetStyle("ATLAS");
21    auto hist=new TH1F("hist","hist",100,-1,1);
22    auto c1=new TCanvas("c1","",600,400);
23
24    for(int i=0;i<tree->GetEntries();i++){
25        tree->GetEntry(i);
26        inpro=0;
27        for(int j=0;j<3;j++)
28            inpro += p_g1[j]*p_g2[j];
29        costheta = inpro/pg1/pg2;
30        hist->Fill(costheta);
31    }
32    hist->GetXaxis()->CenterTitle();
33    hist->GetYaxis()->CenterTitle();
34    hist->SetTitle("Opening angle cos#theta_{#gamma#gamma}");
35    hist->SetTitle("Counts/ bin");
36    hist->Draw("hist");
37
38    return;
39 }
```

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ の擬似データを解析しよう。

1. $\gamma_1\gamma_2$ の開口角 $\theta_{\gamma\gamma}$ の分布をつくってみよう。

\$root -l src/exam4_5.cxx



Q. なぜこのような分布になるのか考えてみよう。

親粒子の質量によって変わるので？

```

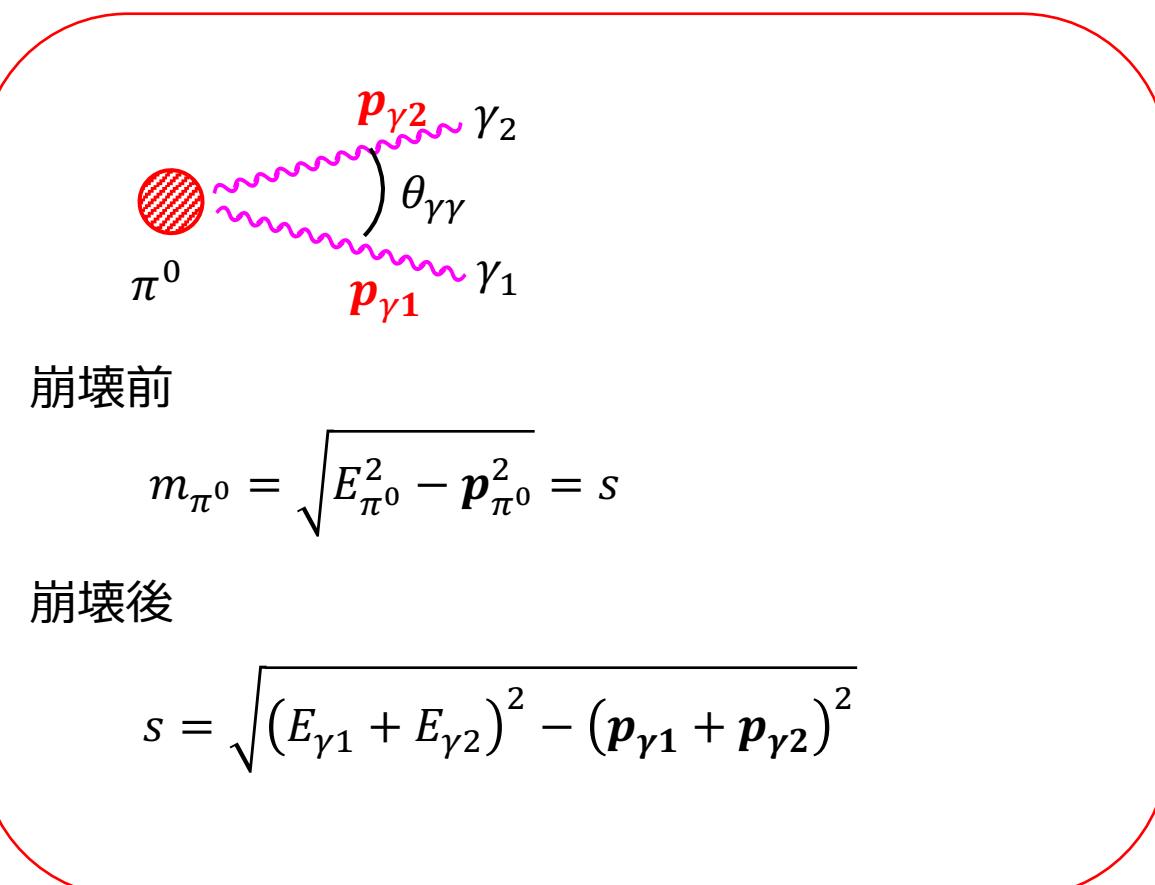
1 void exam4_5(){
2
3     int event;
4     double Epi, ppi, p_pi[3], Eg1, Eg2, pg1,pg2, p_g1[3],p_g2[3];
5     double costheta,inpro;
6
7     auto file = new TFile("exam4.root");
8     auto tree =(TTree*) file->Get("tree");
9     tree->SetBranchAddress("event",&event);
10    tree->SetBranchAddress("Epi",&Epi);
11    tree->SetBranchAddress("ppi",&ppi);
12    tree->SetBranchAddress("p_pi[3]",&p_pi);
13    tree->SetBranchAddress("Eg1",&Eg1);
14    tree->SetBranchAddress("pg1",&pg1);
15    tree->SetBranchAddress("p_g1[3]",&p_g1);
16    tree->SetBranchAddress("Eg2",&Eg2);
17    tree->SetBranchAddress("pg2",&pg2);
18    tree->SetBranchAddress("p_g2[3]",&p_g2);
19
20    gROOT->SetStyle("ATLAS");
21    auto hist=new TH1F("hist","hist",100,-1,1);
22    auto c1=new TCanvas("c1","",600,400);
23
24    for(int i=0;i<tree->GetEntries();i++){
25        tree->GetEntry(i);
26        inpro=0;
27        for(int j=0;j<3;j++)
28            inpro += p_g1[j]*p_g2[j];
29        costheta = inpro/pg1/pg2;
30        hist->Fill(costheta);
31    }
32    hist->GetXaxis()->CenterTitle();
33    hist->GetYaxis()->CenterTitle();
34    hist->SetTitle("Opening angle cos#theta_{#gamma#gamma}");
35    hist->SetTitle("Counts/ bin");
36    hist->Draw("hist");
37
38    return;
39 }
```

例題4) 粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+ \pi^0; \pi^0 \rightarrow \gamma\gamma$ の擬似データを解析しよう。

2. π^0 の invariant mass をつくってみよう。

invariant mass (不变質量)



ここで、

γ は massless なので

$$E_{\gamma_1} = |\mathbf{p}_{\gamma_1}| = p_{\gamma_1}, \quad E_{\gamma_2} = |\mathbf{p}_{\gamma_2}| = p_{\gamma_2}$$

注: $c=1$

$$\begin{aligned} (\mathbf{p}_{\gamma_1} + \mathbf{p}_{\gamma_2})^2 &= |\mathbf{p}_{\gamma_1}|^2 + |\mathbf{p}_{\gamma_2}|^2 + 2\mathbf{p}_{\gamma_1} \cdot \mathbf{p}_{\gamma_2} \\ &= E_{\gamma_1}^2 + E_{\gamma_2}^2 + 2\mathbf{p}_{\gamma_1} \cdot \mathbf{p}_{\gamma_2} \end{aligned}$$

$$\begin{aligned} (E_{\gamma_1} + E_{\gamma_2})^2 - (\mathbf{p}_{\gamma_1} + \mathbf{p}_{\gamma_2})^2 &= 2E_{\gamma_1}E_{\gamma_2} - 2\mathbf{p}_{\gamma_1}\mathbf{p}_{\gamma_2} \cos\theta_{\gamma\gamma} \\ &= 2p_{\gamma_1}p_{\gamma_2}(1 - \cos\theta_{\gamma\gamma}) \end{aligned}$$

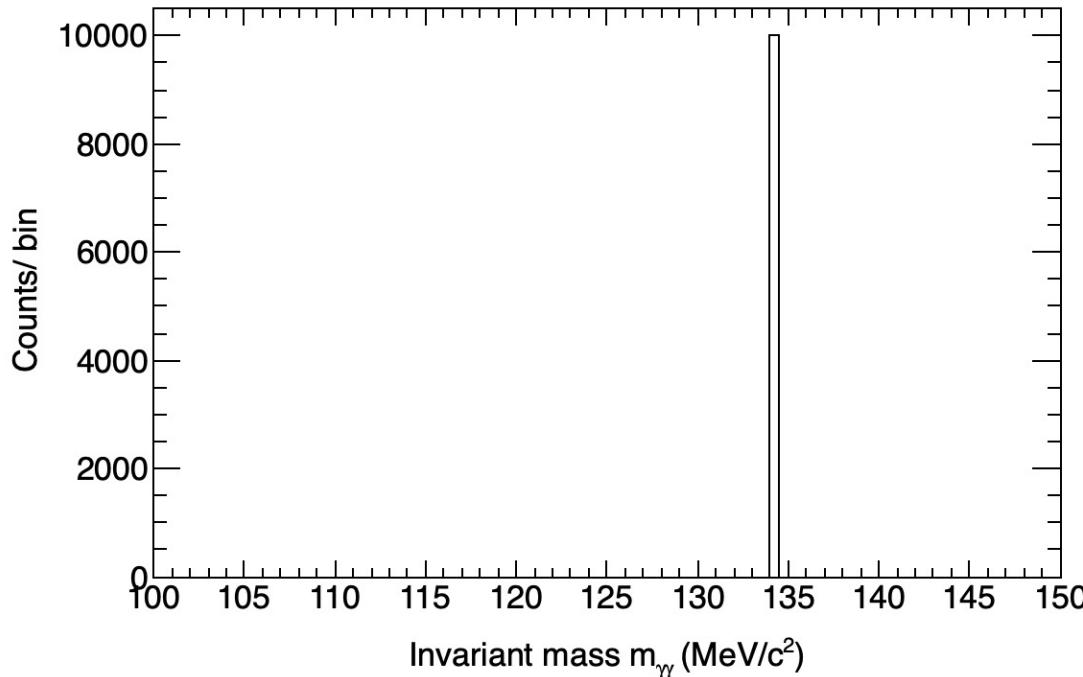
Invariant Mass

$$m_{\gamma\gamma} = \sqrt{2p_{\gamma_1}p_{\gamma_2}(1 - \cos\theta_{\gamma\gamma})}$$

例題4)粒子の崩壊を疑似的に再現してみよう

$K^+ \rightarrow \pi^+\pi^0; \pi^0 \rightarrow \gamma\gamma$ の擬似データを解析しよう。

2. π^0 のinvariant massをつくってみよう。



Invariant Mass

$$m_{\gamma\gamma} = \sqrt{2 p_{\gamma 1} p_{\gamma 2} (1 - \cos \theta_{\gamma\gamma})}$$

src/exam4_6.cxx

```
30  
31     invmass = 2*pg1*pg2*(1-costheta);  
32     invmass = sqrt(invmass);  
33     cout<<invmass<<endl;  
34     hist->Fill(invmass);  
35 }  
36 hist->GetXaxis()->CenterTitle();  
37 hist->GetYaxis()->CenterTitle();  
38 hist->SetXTitle("Invariant mass m_gamma#gamma (MeV/c2)");  
39 hist->SetYTitle("Counts/ bin");  
40 hist->Draw("hist");  
41  
42 return;  
43 }  
44 }
```

π^0 のinvariant massが、134 MeV/c^2 にピークが現れたので、これまでの一連のコードに、バグがないことが確認できた。

また、この変数を実際の測定で計算して、(1)事象選択のカット条件に使ったり、(2)ピーク幅から検出器性能を評価、(3)他のエネルギー領域での粒子探索に使ったりできる。

実習課題4) 例題のコードを改造して、陽子崩壊: $p \rightarrow e^+ \pi^0$; $\pi^0 \rightarrow \gamma\gamma$ の擬似データを生成して、 $\gamma_1\gamma_2$ の開口角 $\theta_{\gamma\gamma}$ 分布、 π^0 invariant mass 分布を計算してみましょう。

例題のコードを改造して、課題に取り組んでみましょう。

角度分布は、何度がpeakになるか？
計算できたらチャットに書いてみよう。

余裕があれば、他の崩壊チャネルについても応用可能か試してみましょう。

陽子崩壊: $p \rightarrow \mu^+ \pi^0$, $p \rightarrow \mu^+ K_s$, $p \rightarrow e^+ \eta$, $p \rightarrow \mu^+ \eta$, ... ニュートリノの反応へ展開: $\nu_\mu p \rightarrow \mu^+ n$ (CCQE反応)
 $\nu_\mu p \rightarrow \mu^- \Delta^{++}$; $\Delta^{++} \rightarrow p \pi^+$ (Δ 共鳴反応)

まとめ

- 復習も兼ねて、相対論、粒子の運動をTH1, Tgraphをつかって物理の数値計算をした。
- 亂数(gRandom)を使って検出器応答を模擬する分布を作成した。分解能の値を変えて、粒子識別(PID)の性能が要求を満たすための、時間分解能の条件を決定する練習をした。
- TTreeについて、実践的な実習を通して、読み込み、生成、解析手法の理解を深めた。#include使って、オブジェクト指向的なコーディングの理解を深めた。
- 特に、K中間子の崩壊の疑似データを生成し、終状態の粒子エネルギー、運動量ベクトルなどのパラメータをTTreeに格納し、 $\cos \theta_{\gamma\gamma}$, $m_{\gamma\gamma}$ を解析するところまで習得した。

あとがき

この内容は、「[アホの子でもわかる物理解析基礎](#)」をベースに、スライドを作成しました。第1弾が2014年、第2弾が2017年にPDFで残しています。この時は、ROOT ver.5でしたが、今回 ver.6で実行できるようにコードは修正します。

ROOTを勉強するにあたって、何かしらの物理を入れて入門らしくしたいなあ～という気持ちで、どこの実験とは言いませんが、最初のかじりをテーマにしています。

質量などの物理量は、PDGを参考にしています。説明を簡単にするために、実際の物理解析からすると、色々すっ飛びしている箇所もあり、お偉い方々に見られると指摘されるかもしれません。なので本稿を100%鵜呑みにしないでください。

最後に、本稿で取り扱ったサンプルソースは改造したりして有意義に使ってください(大したことはやっていないのですが)。