ROOT 解析入門 II
～ アホの子でもわかる物理解析基礎 ～

H. Ito

2017 年 7 月 9 日

目次

1　はじめに

内容は素粒子実験物理学に絡めて、例題で ROOT 解析の手助けになればいいかなと思ってます。解析のコンセプトはスクリプトマクロ形式でコマンド root -l xxx.cxx だけで実効できるようにしています。重要なのはマクロの名前です。コードの中で void sample1() を定義していれば、sample1.cxx とファイル名にしないと ROOT から怒られますので注意してください。環境は ROOT ver. 5.34 を使用しています。2016 年ごろに ver.6 がリリースされましたが、未だにこちらを使っています。これ以降の話は C/C++ と Shell Script の内容が混じっている可能性がありますが、ちょいちょい説明し忘れることがあるので注意してください。インストールは別紙「アホの子でもできる n 次元解析」を参考にしてください。自分がアホでないと思っている方は是非閲覧しないでください。著者のアホ度がバレてしまうので。

2　相対論入門

粒子の運動学的な計算は学部生にとっては必修だと思います。
段階を経て理解できるように努めています。

2.1　TGraph で関数をプロットしてみよう

ここでは単純に数学の勉強です。サンプルを下に示します。

sample1.cxx

```
void sample1(){
      int n=100;
      double x[100],y[100];
      for(int i=0;i<100;i++){
            x[i]=i*0.01;
            y[i]=2*x[i];
      }
      graph=new TGraph(n,x,y);
      graph->SetTitle("tiltle");
      graph->GetXaxis()->SetTitle("x-axis");
      graph->GetYaxis()->SetTitle("y-axis");
      c1=new TCanvas("c1");
      graph->Draw("al");
      c1->Print("output.pdf");
      return;
}
```

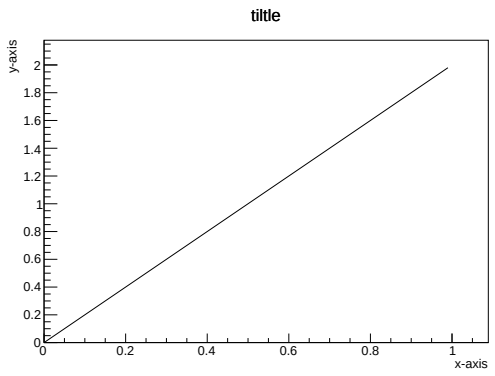


図 1　$y = 2x$ のグラフ

これは $y = 2x$ の曲線をプロットするプログラムです。100 回 for 文でループさせて xy の配列に格納させます。その後、TGraph を定義して、タイトル、x 軸 y 軸のラベルを導入します。TCanvas でキャンバスを定義してそこに graph を描写する流れです。

2.2　粒子の運動量と速度の関係

静止系 K から速度比 $\beta = v/c$ で動く系 K' へローレンツ変換した場合、粒子エネルギーと運動量の関係は

$$\begin{pmatrix} E' \\ p' \end{pmatrix} = \begin{pmatrix} \gamma & \gamma\beta \\ \gamma\beta & \gamma \end{pmatrix} \begin{pmatrix} E \\ p \end{pmatrix} \tag{1}$$

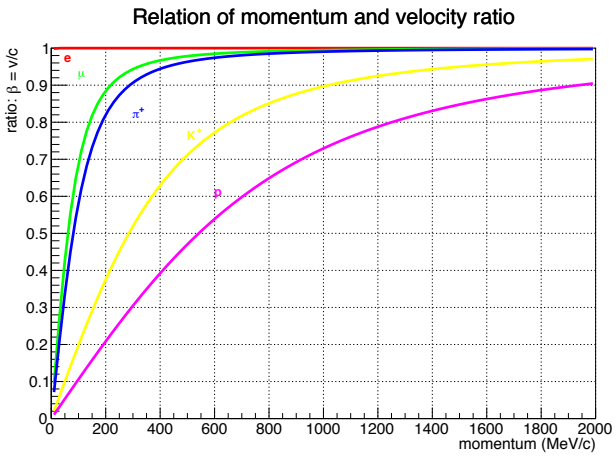
で表せます。粒子静止系 K: $p = (E, -\mathbf{p}) = (\mathbf{m}, \mathbf{0})$ とすると、話がラクになりますね。ここから $E = \gamma m$, $p = \gamma\beta m$ なので $\beta = 1/\sqrt{1 + (m/p)^2}$ が導き出せます。ここで大事なのが粒子の質量です。

- 電子質量 0.511 MeV/c^2,
- μ 粒子質量 105.6 MeV/c^2,
- π^+ 中間子質量 139.6 MeV/c^2,
- K^+ 中間子の質量 493.7 MeV/c^2,
- 陽子質量 938.3 MeV/c^2

を前提にしてサンプルソースを改良して運動量と速度比の関係をグラフにします。

sample2.cxx

```
void sample2(){
    int n=100;
    double x[100],y[100];
    double m[5];
    m[0]=0.511;
    m[1]=105.6;
    m[2]=139.6;
    m[3]=493.7;
    m[4]=938.3;

    graph[5]=new TGraph();
    for(int j=0;j<5;j++){
        for(int i=0;i<100;i++){
            x[i]=i*20+10;//p (MeV/c)
            y[i]=1./sqrt(1.+(m[j]/x[i])**2);//beta
        }
        graph[j]=new TGraph(n,x,y);
        graph[j]->SetLineColor(j+2);
        graph[j]->SetLineWidth(3);
    }
```



sample2.cxx の続き

```
    c1=new TCanvas("c1");
    c1->SetGridx();
    c1->SetGridy();
    graph[0]->SetTitle("Relation of momentum and velocity ratio");
    graph[0]->GetXaxis()->SetTitle("momentum (MeV/c)");
    graph[0]->GetYaxis()->SetTitle("ratio: #beta = v/c");
    graph[0]->GetHistogram()->SetMaximum(1);
    graph[0]->GetHistogram()->SetMinimum(0);
    graph[0]->GetXaxis()->SetLimits(0,2000);
    graph[0]->Draw("al");
    for(int j=1;j<5;j++){
       graph[j]->Draw("l");
    }

    text=new TLatex(50,0.96,"e");
    text->SetTextColor(2);
    text->SetTextSize(0.03);
    text->Draw("same");

    text=new TLatex(100,0.92,"#mu");
    text->SetTextColor(3);
    text->SetTextSize(0.03);
    text->Draw("same");

    text=new TLatex(300,0.81,"#pi^{+}");
    text->SetTextColor(4);
    text->SetTextSize(0.03);
    text->Draw("same");

    text=new TLatex(500,0.75,"K^{+}");
    text->SetTextColor(5);
    text->SetTextSize(0.03);
    text->Draw("same");

    text=new TLatex(600,0.6,"p");
    text->SetTextColor(6);
    text->SetTextSize(0.03);
    text->Draw("same");

    c1->Print("output.pdf");
    return;
}
```

ここで注目してもらいたいのは graph[5] が先に定義されていることです。本当は TGraph*graph[5] と定義するのでしょうが、ROOT さんはそこらへんが結構いい加減みたいです。すると graph[i] は i でループさせることができるので、粒子ごとにプロットができるようになります。そして、曲線の色は SetLineColor(int fColor) で引数 fColor の番号で色を指定します。TCanvas にグリッド線を入れてグラフが見やすくするのは趣味ですね。Log スケールにしたい場合は SetLogy() など挿入するといいでしょう。

TGraph の Draw() 関数の引数には"al"とありますが、"a"は軸を用意すると言う意味で新規でキャンバスに描写します。"l"はラインで折れ線グラフモード、他には"p"は点グラフになるので覚えておくと良いでしょう。
プロットするグラフの表示範囲を選択したい時に便利なコマンドを以下に示します。

```
graph->GetXaxis()->SetLimits(double min,double  max);
graph->GetHistogram()->SetMaximxum(double max);
graph->GetHistogram()->SetMinimum(double min);
```

これはコマンドの名前通りで、X 座標のとりうる範囲を選択します。Y 軸は GetYaxis でとってくないで、GetHistogram でとってくるところが要注意です。

キャンバス内にテキストを表示させる関数は TText() もしくは TLatex() がある。著者のオススメは TLatex で Tex の質感でギリシャ文字や特殊文字をキャンバスに代入できます。書き方の例は

```
text=new TLatex(100, 300, "test e #mu #nu #pi K p");
text->SetTextSize(0.03);
text->SetTextColor(2);
text->Darw("same");
```

です。TLatex の引数は x 座標、y 座標、そして文字の順番です。挿入する位置は文字 BOX をイメージすると左上の点に位置します。また、ポインタである text は本当なら、TLatex*text; とするところですし、何度も重複するのは良くないのですが、何度も言いますが、ROOT さんは適度にいい加減なので、何度同じポインタを使っても重ね書きしてくれます。

2.3 飛行時間測定法 (Time of flight)

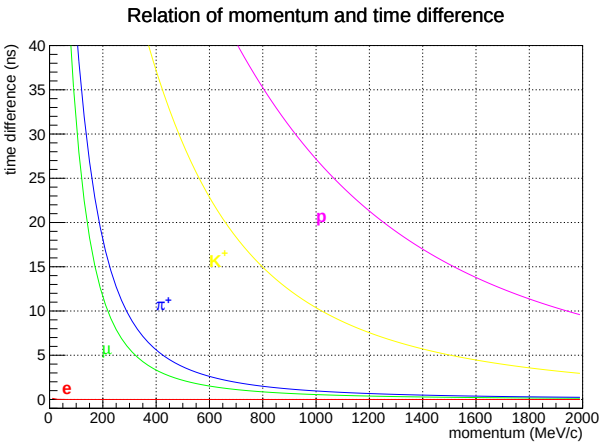
より実験的なグラフを書いてみましょう。粒子の速度が運動量に依存しているこということはわかりました。例えば、距離 $L = 10$m だけ離した 2 つの粒子検出器を用意し、片方をスタート、他方をストップで同じ運動量 p を持つ粒子を通過させます。運動量が同じで質量が違うので光速との時間差を観測することで粒子を特定することが可能です。この測定法は飛行時間測定法 (TOF: Time of flight) と呼ばれ粒子識別の一つとして現在の素粒子・原子核物理学実験で採用されている技術です。

sample3.cxx

```
void sample3(){
    int n=100;
    double x[100],y[100];
    double m[5]={0.511,105.6,139.6,493.7,938.3};
    double c=2.99e8;
    double L=30;//difference distance
```

sample3.cxx の続き

```
graph[5]=new TGraph();
for(int j=0;j<5;j++){
    for(int i=0;i<100;i++){
        x[i]=i*20+10;//p (MeV/c)
        y[i]=(1.-
         1./sqrt(1.+(m[j]/x[i])**2))
         /c*L*1e9;//dT(ns)
    }
    graph[j]=new TGraph(n,x,y);
    graph[j]->SetLineColor(j+2);
}
```




```
c1=new TCanvas("c1");
c1->SetGridx();
c1->SetGridy();
graph[0]->SetTitle("Relation of momentum and time difference");
graph[0]->GetXaxis()->SetTitle("momentum (MeV/c)");
graph[0]->GetYaxis()->SetTitle("time difference (ns)");
graph[0]->GetHistogram()->SetMaximum(40);
graph[0]->GetHistogram()->SetMinimum(-1);
graph[0]->GetXaxis()->SetLimits(0,2000);
graph[0]->Draw("al");
for(int j=1;j<5;j++){
    graph[j]->Draw("l");}

text=new TLatex(50,0.5,"e");
text->SetTextColor(2);
text->SetTextSize(0.04);
text->Draw("same");

text=new TLatex(200,5,"#mu");
text->SetTextColor(3);
text->SetTextSize(0.04);
text->Draw("same");

text=new TLatex(400,10,"#pi^{+}");
text->SetTextColor(4);
text->SetTextSize(0.04);
text->Draw("same");
```

sample3.cxx の続き

```
    text=new TLatex(600,15,"K^{+}");
    text->SetTextColor(5);
    text->SetTextSize(0.04);
    text->Draw("same");

    text=new TLatex(1000,20,"p");
    text->SetTextColor(6);
    text->SetTextSize(0.04);
    text->Draw("same");

    c1->Print("output.pdf");
    return;
}
```

出力結果の図を見ると、電子は数百 MeV でほぼ光速と等しく、粒子の質量が重たくなるにつれて時間差が大きくなります (当たり前でしょうが)。高エネルギー素粒子物理の領域では数 GeV/c の運動量領域で実験が行われることが多く、となると、時間差は電子と π 中間子で 1 ns も時間差が生まれないので、TOF で粒子識別することがほぼ無理。運動量が大きくなればなるほど TOF による粒子識別が困難になることがわかると思います。

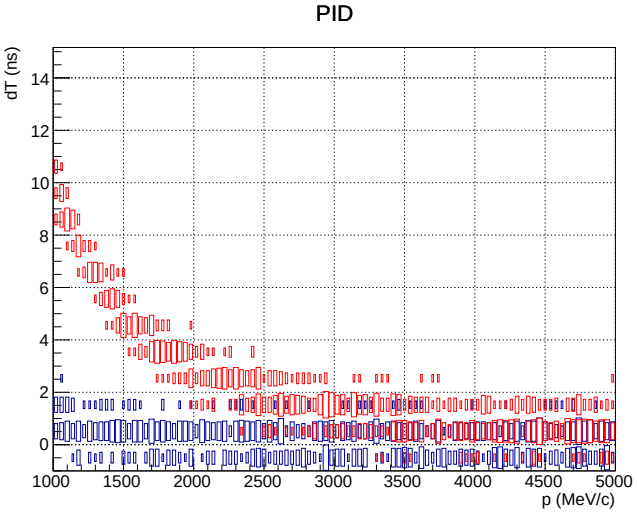
2.4 粒子識別 (PID)

質量から粒子を同定することなどを粒子識別 (PID: particle identification) と呼びます。TOF について理解したところで、例題：1 GeV/c における TOF での π/K 識別はどのようにして行うのか？ まずデータがないのはいただけませんので、仮想的に生成しましょう。

運動量分解能が標準偏差 5 MeV/c で、時間分解能が標準偏差で 200 ps であると仮定します。100-1000 MeV/c の領域で π と K 中間子を発生させます。

sample4.cxx

```
void sample4(){
  double mpi=139.6;
  double mK=493.7;
  double p;
  double dt;
  double c=3e8;//(m/s)
  double L=30;

  his1=new TH2F("his1","his1",
                1e2,1e3,5e3,1e2,-1,100);
  his2=new TH2F("his2","his2",
                1e2,1e3,5e3,1e2,-1,100);
```



sample4.cxx の続き

```
  ofstream ofs("data.dat");
  for(int i=0;i<100000;i++){
    p = gRandom->Uniform(1000,5000);
    dt=(1.-1./sqrt(1.+(mpi/p)**2))/
      c*L*1e9;
    p=gRandom->Gaus(p,5);
    dt=gRandom->Gaus(dt,0.2);
    ofs<<p<<" "<<dt<<endl;
    his1->Fill(p,dt);

    dt=(1.-1./sqrt(1.+(mK/p)**2))/
      c*L*1e9;
    p=gRandom->Gaus(p,5);
    dt=gRandom->Gaus(dt,0.2);
    ofs<<p<<" "<<dt<<endl;
    his2->Fill(p,dt);
  }

  gStyle->SetOptStat(0);
  c1=new TCanvas("c1","",500,400);
  c1->SetGridx();
  c1->SetGridy();

  his1->SetTitle("PID");
  his1->SetXTitle("p (MeV/c)");
  his1->SetYTitle("dT (ns)");
  his1->GetXaxis()->SetRangeUser(1000,5000);
  his1->GetYaxis()->SetRangeUser(-1,15);
  his1->Draw("box");

  his2->SetLineColor(2);
  his2->Draw("boxsame");

  ofs.close();
  c1->Print("output.pdf");

  return;
}
```

乱数は gRandom で生成することができます。
Uniform(start, stop) で start から stop までの領域を一様乱数で値を出力します。
例えば、

```
gRandom->Uniform(0,1);
```

では 0〜1 までの少数を一様に生成するのです。
一方、

```
gRandom->Gaus(mean, sigma)
```

は平均値 mean から標準偏差 sigma の正規分布の重みで乱数を生成します。

2 次元ヒストグラムの Draw オプションはいくつかあり、よく使用するのは box と colz, そして lego でしょう。重ね書きするときは same を加えると良いです。何もオプションをつけないと、ただの点としてプロットされます。そして紺色ヒストグラムが π で、赤ヒストグラムが K を示してます。

ofstream はファイル出力の C++ のクラスです。ofs() の引数に出力先を入力し、

```
ofs<<a<<" "<<b<<endl;
```

で 1 行に a と b を出力することができます。ここで endl は改行を意味します。
データは π と K を交互に出力され、配列は運動量 (MeV/c) と時間差が格納されています。

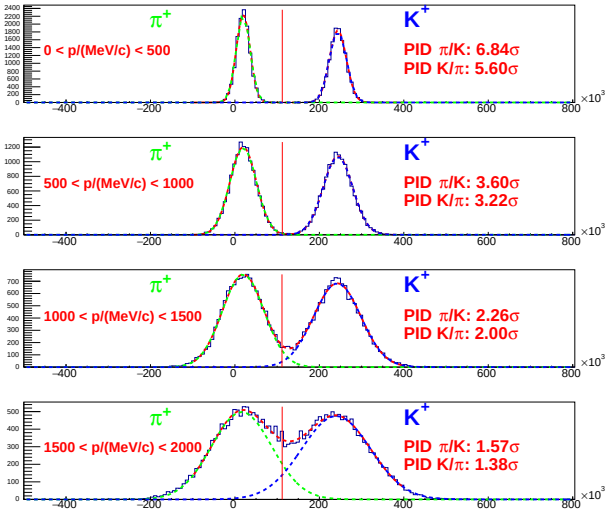
ここで生成したデータを使って質量分布を作っていきましょう。

system 関数は Linux 標準コマンドを実行してくれるので役立ちます。
wc コマンドは ファイルの行数を返すコマンドです。それにパイプをつないで、cat を命令します。cat は出力をそのまま返すのですが、> でつなぐとその後のファイルに出力出来ます。つまり、.lg というファイルに data.dat の行数つまり、イベント数が格納されることになります。こうすることで、for 文でループを回す最大値を知ることができます。

sample5 に解析コードを示します。

sample5.cxx

```
void sample5(){
  double m,m2;
  double p;
  double dt;
  double c=3e8;//(m/s)
  double L=30;
  int region;
  char name[100];

  system("wc -l data.dat|cat > .lg");
  ifstream lg(".lg");
  int loopmax=0;
  lg>>loopmax;
  lg.close();
  ifstream ifs("data.dat");
  TH1S*mass[10];
  for(int j=0;j<8;j++){
    sprintf(name,"mass%d",j);
    mass[j]=new TH1S(name,name,2e2,-5e5,1e6);
  }
  for(int i=0;i<loopmax;i++){
    ifs>>p>>dt;
    beta=1.-dt*1e-9*c/L;
    m2=p*p/beta/beta*(1+beta)*(1-beta);//MeV^2
    region=(p-1e3)/5e2;
    if(region>7) region =7;
    mass[region]->Fill(m2);
  }
    ifs.close();
```



sample5.cxx の続き

```
c1=new TCanvas("c1","",500,400);
 c1->Divide(1,4);
 gStyle->SetOptStat(0);
 for(int j=0;j<4;j++){
   c1->cd(j+1)->SetGridx();
   c1->cd(j+1)->SetGridy();
   mass[j]->GetXaxis()
    ->SetRangeUser(-5e5,8e5);
   mass[j]->Draw();
   func=new TF1("func","gaus(0)+gaus(3)");
   func->SetLineStyle(2);
   func->SetParameters(
    2e3,1.5e4,5e4,2e3,2.4e5,5e4);
   mass[j]->Fit("func","Q","",-1e5,4e5);

   func1=new TF1("func1","gaus",-5e5,1e6);
   func1->SetLineColor(3);
   func1->SetLineStyle(2);
   func1->SetParameters(
       func->GetParameter(0),
       func->GetParameter(1),
       func->GetParameter(2)
     );
   func1->Draw("same");
   func2=new TF1("func2",
    "gaus",-5e5,1e6);
   func2->SetLineColor(4);
   func2->SetLineStyle(2);
   func2->SetParameters(
           func->GetParameter(3),
           func->GetParameter(4),
           func->GetParameter(5)
       );
   func2->Draw("same");

     mass[j]->SetTitle("");
     mass[j]->GetXaxis()
      ->SetLabelSize(0.1);
     mass[j]->GetYaxis()
      ->SetLabelSize(0.06);

   sprintf(name,
      "%d < p/(MeV/c) < %d",
      j*500,(j+1)*500);
   text=new TLatex(-450e3,
      mass[j]->GetMaximum()*0.5,name);
   text->SetTextSize(0.12);
   text->SetTextColor(2);
   text->Draw("same");

   text=new TLatex(
      -200e3,
      mass[j]->GetMaximum()*0.8,
      "#pi^{+}");
   text->SetTextSize(0.2);
   text->SetTextColor(3);
   text->Draw("same");

   text=new TLatex(
      400e3,
      mass[j]->GetMaximum()*0.8,
      "K^{+}");
   text->SetTextSize(0.2);
   text->SetTextColor(4);
   text->Draw("same");

   double sigma=
      (func2->GetParameter(1) -
      func1->GetParameter(1))/
      func1->GetParameter(2)/2;
   sprintf(name,
      "PID #pi/K: %0.2f#sigma",sigma);
   text=new TLatex(400e3,
      mass[j]->GetMaximum()*0.5,name);
   text->SetTextSize(0.15);
   text->SetTextColor(2);
   text->Draw("same");
```

sample5.cxx の続き

```
sigma =
   (func2->GetParameter(1) -
   func1->GetParameter(1))/
   func2->GetParameter(2)/2;
sprintf(name,
   "PID K/#pi: %0.2f#sigma",sigma);
text=new TLatex(400e3,
   mass[j]->GetMaximum()*0.3,name);
text->SetTextSize(0.15);
text->SetTextColor(2);
text->Draw("same");

line=new TLine((
                                    func2->GetParameter(1)-
                                      func1->GetParameter(1))/2,
                                    0,
                                    (func2->GetParameter(1)-
                                      func1->GetParameter(1))/2,
                                    mass[j]->GetMaximum());
                                line->SetLineColor(2);
                                line->Draw("same");
                            }
                            c1->Print("output.pdf");
                            return;
                        }
```

事前に mass という配列ポインタに TH1S のヒストグラムのクラスを付与しておきます。
TF1S*mass[10];
その上で、for 文で new TH1S() でヒストグラムを定義します。
そして、ifstream を使ってデータを読み出します。m2 は質量の 2 乗を意味しています。mass は運動量ごとに Fill することで識別能力が運動量ごとにどう変わっていくか見ることができます。

フィッティングは 2 つの正規分布でフィットを行いますが、その前に 2 つのガウスを持つ関数を定義します。
func=new TF1(char name, char function);
で定義します。一つ目の引数は関数名、2 つ目は関数を char 型で書きます。ROOT の中には gaus という関数が事前に登録されていて、正規分布を示します。括弧の中身は自由パラメータが 0 から始まり、3 つ使用するので、
$gaus(0) = [0] * exp(-((x - [1]/2/pi/[2])^2))$
を示します。そして function の初期パラメータを設定します。最後にフィットをしますが、mass-¿Fit(char functionname, char option1, char option2, double start, double stop); 最初の引数はフィットする関数名、option1 に Q を入れると、標準出力で情報を出さないようになります。基本的に option1, 2 はダブルクォーテーションでくくって、中身なしにするといいと思います。start と stop はフィット領域の最初と最後を示します。

粒子質量が正規分布で測定できたとして、ちょうど π と K のピークの半分を π/K 識別の境目にします。π の山のピークと境目が標準偏差の何倍かということで粒子識別能力を評価します。0-500 MeV/c の運動量領域では 6.84σ
500-1000 MeV/c の運動量領域では 3.60σ
1000-1500 MeV/c の運動量領域では 2.26σ
1500-2000 MeV/c の運動量領域では 1.57σ
この値が大きいと粒子識別能力が高いことを意味します。統計学でよく言われる標準偏差では 1 σ が 68%,2 σ が 95%,3 σ が 99.7% をカバーすることが知られています。これを指標に 3σ 以上あれば十分な識別能力があると判断するなら、TOF による粒子識別の限界は 1000 MeV/c あたりにあることがわかります。

3　粒子衝突と崩壊 ∼ 静止 K^+ 崩壊を中心に ∼

おなじみの K^+ です。
これは $u\bar{s}$ で結合している中間子 (meson) ですよね。
質量は $m_K = 493.677$ MeV.
寿命は $\tau = 12.38\ ns$.
主な崩壊チャンネルは表 1 に示します。

表 1　Main K^+ decay channel

Decay channel	Branching ratio
$K^+ \to e^+\nu_e(K_{e2})$	1.58×10^{-5}
$K^+ \to \mu^+\nu_\mu(K_{\mu 2})$	63.55%
$K^+ \to e^+\pi^0\nu_e(K_{e3})$	5.07%
$K^+ \to \mu^+\pi^0\nu_\mu(K_{\mu 3})$	3.35%
$K^+ \to \pi^+\pi^0(K_{\pi 2})$	20.66%
$K^+ \to \pi^+\pi^0\pi^0$	1.76%
$K^+ \to \pi^+\pi^+\pi^-$	5.59%

3.1　2 体崩壊

ターゲット内に K^+ を止めて 2 対崩壊した場合、2 粒子は Back to Back で放射するため、理解が簡単でしょう。K_{e2} と $K_{\mu 2}$ はヘリシティ抑制によって前者が抑制されている典型的な例です。分岐幅は $\Gamma = g_l \frac{G^2}{8\pi} f_K^2 m_K m_l^2 (1 - (m_l/m_K)^2)^2$ で与えられるため、同じセミレプトニック崩壊過程である K_{e2} と $K_{\mu 2}$ の崩壊分岐比は質量の関係から計算できます。

$$\begin{aligned} R_K = \frac{\Gamma(K_{e2})}{\Gamma(K_{\mu 2})} &= \frac{{m_e}^2}{{m_\mu}^2} \left(\frac{{m_K}^2 - {m_e}^2}{{m_K}^2 - {m_\mu}^2} \right)^2 \\ &= 2.57 \times 10^{-5} \end{aligned}$$

実際には制動放射による補正が入り、標準模型では ${R_K}^{SM} = (2.477 \pm 0.001) \times 10^{-5}$ と計算されます。実験では ${R_K}^{exp} = (2.488 \pm 0.009) \times 10^{-5}$ が報告されています。

$K_{\pi 2}$ は単色の π^0 が生成されるため、$\pi^0 \to 2\gamma$ 事象から 2 つの γ 線のエネルギーとそれぞれの運動量を観測するとピーク構造を得られます。そのため電磁カロリメータの較正測定によく使用されます。

2 体崩壊はエネルギーと運動量保存則、そして娘粒子 2 つの運動量絶対値が等しいことから、K_{e2}, $K_{\mu 2}$, $K_{\pi 2}$ のそれぞれ放射される荷電粒子のエネルギーと運動量が計算できます。

$$\begin{aligned} {m_K}^2 &= (E_l + E_\nu)^2 \\ m_K &= \sqrt{p^2 + {m_1}^2} + \sqrt{p^2 + {m_2}^2} \end{aligned}$$

ここで、娘粒子の 1 つが ν なら質量 $m_2 = 0$ と近似して運動量を決定します。

$$\begin{aligned} m_K &= \sqrt{p^2 + {m_1}^2} + p \\ p &= \frac{{m_K}^2 - {m_1}^2}{2m_K} \end{aligned}$$

例えば、K_{e2} の場合単色運動量 $p_e = 246.8$ MeV/c の e^+ が、$K_{\mu 2}$ の場合 $p_\mu = 235.5$ MeV/c の μ^+ である。一方。$K_{\pi 2}$ の場合は $m_2 = 0$ 近似できないので、ちゃんと計算しましょう。

$$
\begin{aligned}
m_K &= \sqrt{p^2 + {m_1}^2} + \sqrt{p^2 + {m_2}^2} \\
{m_K}^2 - 2m_K\sqrt{p^2 + {m_2}^2} + p^2 + {m_2}^2 &= p^2 + {m_1}^2 \\
2m_K\sqrt{p^2 + {m_2}^2} &= m_K^2 - m_1^2 + m_2^2 \\
p^2 + {m_2}^2 &= \left(\frac{m_K^2 - m_1^2 + m_2^2}{2m_K}\right)^2 \\
p &= \sqrt{\left(\frac{{m_K}^2 - {m_1}^2 + {m_2}^2}{2m_K}\right)^2 - {m_2}^2}
\end{aligned}
$$

つまり、$p_\pi = 205.4$ MeV/c の π^+ が放出されるみたいな。$m_2 = 0$ で前の式と一致してますね。

sample6.cxx

```
void sample6(){
    double mK=493.677, me=0.511, mmu=105.6, mpi=139.6,mpi0=134.1;
    double pK=0, pe=0,pnu=0, pmu=0, ppi=0, ppi0;
    twobodydecay(mK, me, pe, 0, pnu);
    twobodydecay(mK, mmu, pmu, 0, pnu);
    twobodydecay(mK, mpi, ppi, mpi0, ppi0);
    cout<<pe<<" "<<pmu<<" "<<ppi<<" "<<endl;
    return;
}

void twobodydecay(double parentM,
          double &daughterM1, double &daughterP1,
          double &daughterM2, double &daughterP2
          ){
    daughterP1 = sqrt(((parentM**2 - daughterM1**2 + daughterM2**2)/
                       (2*parentM))**2-daughterM2**2);
    daughterP2 = daughterP1;
    return;
}
```

このプログラムは 2 体崩壊関数 twobodydecay() を使用して、娘粒子の運動量を出力します。引数に親粒子の質量、そして娘粒子の質量と運動量が返ってくる。sample6.cxx ソースの中身で最初の twobodydecay() は、K_{e2} で e^+ と ν_e の 2 粒子の運動量を計算しています。引数の変数の前に & をつけると関数内で計算した値を、戻り値で返すことができるので便利ですよ。

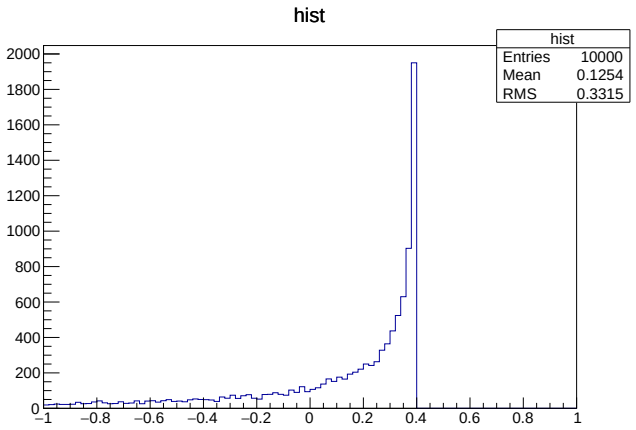
親粒子が飛んでいる際意中に崩壊する事象を in-flight 崩壊と言うのはご存知かと。せっかくなのでローレンツ変換を使って、親粒子が in-flight 崩壊した時に放射される娘粒子の運動量ベクトルを決定していきましょう。π^0 は寿命が 8.5×10^{-17} s なので、25 nm しか進まず、2 本の γ 線を放出します。$K_{\pi 2}$ で生成された単色 π^0 が放出する 2γ を考えていきましょう。

sample7.cxx

```
void sample7(){
  double mpi=134.9, mg1=0,mg2=0;
  double ppi[3]={0,0,205.4},mom;
  double Eg1, pg1,Eg2, pg2,p[2][3]={0};
  double costheta,phi,sintheta,costhetagg;
  double beta[3],gamma[3];
```

sample7.cxx の続き

```
  gSystem->Load("sample6.cxx");
  mom=0;
  for(int i=0;i<3;i++) mom+=ppi[i];
  for(int i=0;i<3;i++){
    beta[i]=pK[i]/sqrt(mK**2 + momK**2);
    gamma[i]=1./sqrt(1.-beta[i]**2);
  }

  hist=new TH1F("hist","hist",100,-1,1);

  twobodydecay(mpi, mg1, pg1, mg2, pg2);
  Eg1=sqrt(mg1**2+pg1**2);
  Eg2=sqrt(mg2**2+pg2**2);;

  for(int i=0;i<10000;i++){
    costheta=gRandom->Uniform(-1,1);
    phi=2*3.141592*gRandom->Uniform(0,1);
    sintheta=sqrt(1-costheta**2);
    p[0][0]=pg1*sintheta*sin(phi);
    p[0][1]=pg1*sintheta*cos(phi);
    p[0][2]=pg1*costheta;

    p[1][0]=-pg2*sintheta*sin(phi);
    p[1][1]=-pg2*sintheta*cos(phi);
    p[1][2]=-pg2*costheta;

    for(int j=0;j<3;j++){
      p[0][j] = gamma[j]*beta[j]*Emu + gamma[j]*p[0][j];
      p[1][j] = gamma[j]*beta[j]*Enu + gamma[j]*p[1][j];
    }
    costhetagg = (p[0][0]*p[1][0]+p[0][1]*p[1][1]+p[0][2]*p[1][2])
                          /sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2)
                          /sqrt(p[0][0]**2+p[0][1]**2+p[0][2]**2);
    hist->Fill(costhetagg);
  }

  c1=new TCanvas("c1");
  hist->Draw();
  c1->Print("output.pdf");
  return;
}
```



このことからも、π^0 から放出された γ 線の opening 角度はローレンツブーストによって前方方向に伸ばされていることがわかります。`gSystem->Load()` を実行すると、sample6.cxx の関数を読み込むことができます。$\cos\theta = 0.4$ は角度でいうと 66.4 度くらい。-1 までテイルをひくのは π^0 静止系で等方的に γ 線が放射する中で丁度 π^0 の進行方向と平行な事象が存在するから。0.4 にピークを持つのは、γ 線の向きと π^0 の向きが垂直の時の角度相関 $\cos(2\tan^{-1}(134.9/205.4)) = 0.397$ が最大であるからで、等方的に放射してもこれ以上角度が狭くなることが理論的にないので、絶壁を持った連続分布を生成します。

3.2　3 体崩壊

主な 3 体崩壊は K_{e3}, $K_{\mu 3}$, $K^+ \to \pi^+\pi^+\pi^-$, $K^+ \to \pi^+\pi^0\pi^0$ などの π を伴います。また、厄介なのはそれぞれの娘粒子の運動量は連続的に分布することです。エネルギー・運動量保存則を満たすように幾何学的な要請とその中での崩壊密度分布から、娘粒子の運動量が決定されます。この分布を Dalitz plot といいます。

3.2.1　$K \to 3\pi$

まずは $K \to 3\pi$ 過程から見ていきましょう。$\pi-$ 静止系では π^+ の 2 体崩壊と同等。その時の π^+ の運動量は

$$\begin{aligned} m_K &= 2\sqrt{{m_\pi}^2 + p^2} + m_\pi \\ m_K - m_\pi &= 2\sqrt{{m_\pi}^2 + p^2} \\ \frac{m_K - m_\pi}{2} &= {m_\pi}^2 + p^2 \\ p &= \sqrt{\left(\frac{m_K - m_\pi}{2}\right)^2 - {m_\pi}^2} \end{aligned} \tag{2}$$

で表せる。3 体崩壊は連続的な運動量を持ち、π^+ の最大の運動量は 110.15 MeV であると計算できました。$K \to 3\pi$ の Dalitz plot 分布は以下の散乱振幅から表されます。

$$\begin{aligned} |M|^2 \propto 1 &+ g\frac{s_3 - s_0}{m_{\pi^+}^2} + h\left\{\frac{s_3 - s_0}{m_{\pi^+}^2}\right\}^2 \\ &+ j\frac{s_2 - s_1}{m_{\pi^+}^2} + k\left\{\frac{s_2 - s_1}{m_{\pi^+}^2}\right\}^2 + ... \end{aligned} \tag{3}$$

ここで、m_{π^+} は π^+ の質量、

$$\begin{aligned} s_i &= (P_K - P_i)^2 \\ &= (m_K - m_i)^2 - 2m_K T_i,\ (i = 1, 2, 3), \end{aligned} \tag{4}$$

$$\begin{aligned} s_0 &= \frac{1}{3}\sum_i s_i \\ &= \frac{1}{3}({m_K}^2 + {m_1}^2 + {m_2}^2 + {m_3}^2) \end{aligned} \tag{5}$$

ここで、P_i と m_i, T_i はそれぞれ崩壊後の娘核 i の 4 元運動量と質量、運動エネルギーを示す。パラメータ g, h, k は最新の研究結果を採用している。CP 対称性が破れていないと仮定しているため $j = 0$. 粒子 3 は奇数パイオンを示しています。$K^\pm \to \pi^\pm\pi^\pm\pi^\mp$ において

$$g = -0.21134 \pm 0.00017, \tag{6}$$

$$h = (1.848 \pm 0.040) \times 10^{-2}, \tag{7}$$

$$j = 0, \tag{8}$$

$$k = (-4.63 \pm 0.14) \times 10^{-3} \tag{9}$$

です。この時の Dalitz plot を描いて見ましょう。

sample8.cxx

```
void sample8(){
    gStyle->SetOptStat(0);
    double mK=493.677, mpi=139;
    double p[3][3]={0};
    dalitz23=new TH2F("dalitz23","dalitz23",120,0,120,120,0,120);
    dalitz31=new TH2F("dalitz31","dalitz31",120,0,120,120,0,120);
    dalitz12=new TH2F("dalitz12","dalitz12",120,0,120,120,0,120);

    for(int i=0;i<100000;i++){
        DalitzdecayKto3piPlus(mK,
                              mpi,p[0][0],p[0][1],p[0][2],
                              mpi,p[1][0],p[1][1],p[1][2],
                              mpi,p[2][0],p[2][1],p[2][2]);
        dalitz23->Fill(sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2),
                       sqrt(p[2][0]**2+p[2][1]**2+p[2][2]**2));
        dalitz31->Fill(sqrt(p[2][0]**2+p[2][1]**2+p[2][2]**2),
                       sqrt(p[0][0]**2+p[0][1]**2+p[0][2]**2));
        dalitz12->Fill(sqrt(p[0][0]**2+p[0][1]**2+p[0][2]**2),
                       sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2));
    }
    c1=new TCanvas("c1","",800,300);
    c1->Divide(3,1);
    c1->cd(1);
    dalitz23->SetXTitle("p_{#pi2} (MeV/c)");
    dalitz23->GetXaxis()->CenterTitle();
    dalitz23->SetYTitle("p_{#pi3} (MeV/c)");
    dalitz23->GetYaxis()->CenterTitle();
    dalitz23->Draw("colz");
    c1->cd(2);
    dalitz31->SetXTitle("p_{#pi3} (MeV/c)");
    dalitz31->GetXaxis()->CenterTitle();
    dalitz31->SetYTitle("p_{#pi1} (MeV/c)");
    dalitz31->GetYaxis()->CenterTitle();
    dalitz31->Draw("colz");
    c1->cd(3);
    dalitz12->SetXTitle("p_{#pi1} (MeV/c)");
    dalitz12->GetXaxis()->CenterTitle();
    dalitz12->SetYTitle("p_{#pi2} (MeV/c)");
    dalitz12->GetYaxis()->CenterTitle();
    dalitz12->Draw("colz");
    c1->Print("output.pdf");
    return;
}
```

sample8.cxx の続き

```
void DalitzdecayKto3piPlus(
            double parentM,
            double daughter1M,  double &daughter1P1,double &daughter1P2,
            double &daughter1P3,double daughter2M,  double &daughter2P1,
            double &daughter2P2,double &daughter2P3,double daughter3M,
            double &daughter3P1,double &daughter3P2,double &daughter3P3){
    double en_p[3]={0},en_E[3]={0},s[4]={0},en_r;
    double pmax=sqrt(((parentM - daughter1M)/2)**2 - daughter1M**2);
    double Tmax=(parentM - 3*daughter1M)/2;
    double Emax=(parentM - daughter1M)/2;
    double g=-0.21134;
    double h=1.848e-2;
    double k=-4.63e-3;
    double M, M2;
    double costheta, sintheta, phi, sintheta3decay, phin, costheta3decay;

    do{
        en_E[0] = gRandom->Uniform(daughter1M,Tmax+daughter1M);
        en_E[1] = gRandom->Uniform(daughter1M,Tmax+daughter1M);
        en_E[2] = parentM - en_E[0] - en_E[1];
        for(int i=0;i<3;i++) en_p[i]=sqrt(en_E[i]**2 - daughter1M**2);
        s[0]=(parentM**2 + 3*daughter1M**2)/3;
        for(int i=1;i<4;i++){
            s[i]=(parentM-daughter1M)**2-
            2*parentM*(en_E[i-1]-daughter1M);
        }
        M = en_E[0]+en_E[1]+en_E[2];
        M2 = 1. + g*(s[3]-s[0])/daughter1M**2
            + h*(s[3]-s[0])**2/(daughter1M**4)
            + k*(s[2]-s[1])**2/(daughter1M**4);
        M2*=1e-1;

            }while(
           en_E[0] > Tmax+daughter1M ||
           en_E[1] > Tmax+daughter1M ||
           en_E[2] > Tmax+daughter1M ||
           en_E[0] < daughter1M ||
           en_E[1] < daughter1M ||
           en_E[2] < daughter1M ||
           gRandom->Uniform(0.00,1.00) > M2
    );
```

sample8.cxx の続き

```
        costheta3decay =
        (en_E[2]**2 - en_E[0]**2 - en_E[1]**2 + daughter1M**2)/
        (2*en_p[0]*en_p[1]);

        // pion 1
        costheta = -1. +2.*gRandom->Uniform(0,1);
        sintheta = sqrt((1.-costheta)*(1.+costheta));
        phi = 2.*3.141592*gRandom->Uniform(0,1);
        daughter1P1 = en_p[0]*sintheta*cos(phi);
        daughter1P2 = en_p[0]*sintheta*sin(phi);
        daughter1P3 = en_p[0]*costheta;

        // pion 2
        sintheta3decay=sqrt((1.+costheta3decay)*(1.-costheta3decay));
        phin=2.*3.141592*gRandom->Uniform(0,1);
        daughter2P1 = en_p[1]*
        (sintheta3decay*cos(phin)*costheta*cos(phi)
         - sintheta3decay*sin(phin)*sin(phi)
         + costheta3decay*sintheta*cos(phi));
        daughter2P2 = en_p[1]*
        (sintheta3decay*cos(phin)*costheta*sin(phi)
         + sintheta3decay*sin(phin)*cos(phi)
         + costheta3decay*sintheta*sin(phi));
        daughter2P3 = en_p[1]*
        (- sintheta3decay*cos(phin)*sintheta
         + costheta3decay*costheta);

        // pion 3
        daughter3P1 = - daughter1P1 - daughter2P1;
        daughter3P2 = - daughter1P2 - daughter2P2;
        daughter3P3 = - daughter1P3 - daughter2P3;

    return;
}
```

DalitzdecayKto3piPlus() 関数の中では運動学的に 3 つの π のエネルギーを与えて、散乱振幅式 (3) から崩壊強度を決定しています。 2 粒子の運動量がわかると、その間の角度を計算できる。

$$\begin{aligned}\cos\Theta &= \frac{\mathbf{p_1}\cdot\mathbf{p_2}}{|p_1|\,|p_2|}\\ &= \frac{(\mathbf{p_1}+\mathbf{p_2})^{\mathbf{2}}-\mathbf{p_1}^{\mathbf{2}}-\mathbf{p_2}^{\mathbf{2}}}{2\,|p_1|\,|p_2|}\\ &= \frac{(\mathbf{p_3})^{\mathbf{2}}-(\mathbf{E_1}^{\mathbf{2}}-\mathbf{m_1}^{\mathbf{2}})-(\mathbf{E_2}^{\mathbf{2}}-\mathbf{m_2}^{\mathbf{2}})}{2\,|p_1|\,|p_2|}\\ &= \frac{({E_3}^2-{m_3}^2)-({E_1}^2-{m_1}^2)-({E_2}^2-{m_2}^2)}{2\,|p_1|\,|p_2|}\end{aligned} \tag{10}$$

ここで $m_1 = m_2 = m_3 = m_\pi$ なのでキャンセルして、

$$\cos\Theta = \frac{{E_3}^2-{E_1}^2-{E_2}^2+{m_\pi}^2}{2\,|p_1|\,|p_2|} \tag{11}$$

と計算できます。次に 3 粒子の運動量の成分を決定しましょう。粒子 1 は θ, ϕ の一様乱数を使って

$$\mathbf{p_1} = |\mathbf{p_1}|\cdot\begin{pmatrix}\sin\theta\cos\phi\\ \sin\theta\sin\phi\\ \cos\theta\end{pmatrix} \tag{12}$$

とわかる。粒子 2 は粒子 1 の周りを角度 Θ で回転する自由度を持つ。一様乱数 Φ を用いて、

$$\mathbf{p_2} = |\mathbf{p_2}|\cdot\begin{pmatrix}\sin\Theta\cos\Phi\cos\theta\cos\phi-\sin\Theta\sin\Phi\sin\phi+\cos\Theta\sin\Theta\cos\phi\\ \sin\Theta\cos\Phi\cos\theta\sin\phi+\sin\Theta\sin\Phi\cos\phi+\cos\Theta\sin\Theta\sin\phi\\ -\sin\Theta\cos\Phi\sin\theta+\cos\Theta\cos\theta\end{pmatrix} \tag{13}$$

と計算することができる。粒子 3 は粒子 1, 2 と親核の運動量保存則から

$$\mathbf{p_3} = -\mathbf{p_1}-\mathbf{p_2} \tag{14}$$

と計算することができる。

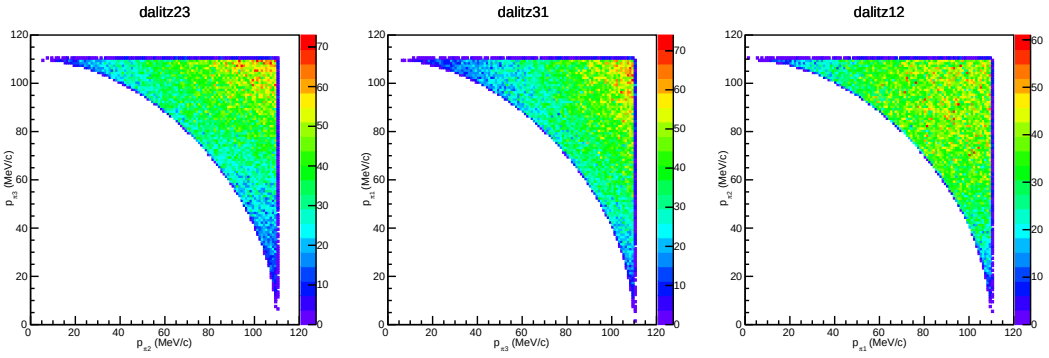


図 2　$K^+ \to \pi^+\pi^+\pi^-$ における π の運動量分布. 粒子 1, 2 は π^+, 粒子 3 は π^- を示す.

$K^\pm \to \pi^\pm\pi^0\pi^0$ において

$$g_0 = 0.626 \pm 0.007, \tag{15}$$
$$h_0 = 0.052 \pm 0.008, \tag{16}$$
$$j_0 = 0, \tag{17}$$
$$k_0 = 0.0054 \pm 0.0035 \tag{18}$$

のパラメータで同様に計算できる。

Dalitzdecayto3piZero.cxx

```
void DalitzdecayKto3piZero(
            double parentM,
            double daughter1M,  double &daughter1P1,double &daughter1P2,
            double &daughter1P3,double daughter2M,  double &daughter2P1,
            double &daughter2P2,double &daughter2P3,double daughter3M,
            double &daughter3P1,double &daughter3P2,double &daughter3P3){
    double en_p[3]={0},en_E[3]={0},s[4]={0},en_r;
    double Tmax=(parentM - daughter1M - daughter2M - daughter3M)/2;
    double g=0.626;
    double h=0.052;
    double k=0.0054;
    double M, M2;
    double costheta, sintheta, phi, sintheta3decay, phin, costheta3decay;

    do{
        en_E[0] = gRandom->Uniform(daughter1M,Tmax+daughter1M);
        en_E[1] = gRandom->Uniform(daughter2M,Tmax+daughter2M);
        en_E[2] = parentM - en_E[0] - en_E[1];

        en_p[0]=sqrt(en_E[0]**2 - daughter1M**2);
        en_p[1]=sqrt(en_E[1]**2 - daughter2M**2);
        en_p[2]=sqrt(en_E[2]**2 - daughter3M**2);

        s[0]=(parentM**2 + daughter1M**2+daughter2M**2+daughter3M**2)/3;
        s[1]=(parentM-daughter1M)**2-2*parentM*(en_E[0]-daughter1M);
        s[2]=(parentM-daughter2M)**2-2*parentM*(en_E[1]-daughter2M);
        s[3]=(parentM-daughter3M)**2-2*parentM*(en_E[2]-daughter3M);
        M = en_E[0]+en_E[1]+en_E[2];
        M2 = 1. + g*(s[3]-s[0])/daughter3M**2
            + h*(s[3]-s[0])**2/(daughter3M**4)
            + k*(s[2]-s[1])**2/(daughter3M**4);
        M2*=1e-1;

    }while(
           en_E[0] > Tmax+daughter1M ||
           en_E[1] > Tmax+daughter2M ||
           en_E[2] > Tmax+daughter3M ||
           en_E[0] < daughter1M ||
           en_E[1] < daughter2M ||
           en_E[2] < daughter3M ||
           gRandom->Uniform(0.00,1.00) > M2
    );
```

Dalitzdecayto3piZero.cxx の続き

```
    costheta3decay =
    (en_E[2]**2 - daughter3M**2 - en_E[0]**2 + daughter1M**2 - en_E[1]**2
    + daughter2M**2)/(2*en_p[0]*en_p[1]);

    costheta = -1. +2.*gRandom->Uniform(0,1);     // pion 1
    sintheta = sqrt((1.-costheta)*(1.+costheta));
    phi = 2.*3.141592*gRandom->Uniform(0,1);
    daughter1P1 = en_p[0]*sintheta*cos(phi);
    daughter1P2 = en_p[0]*sintheta*sin(phi);
    daughter1P3 = en_p[0]*costheta;

    sintheta3decay=sqrt((1.+costheta3decay)*(1.-costheta3decay));     // pion 2
    phin=2.*3.141592*gRandom->Uniform(0,1);
    daughter2P1 = en_p[1]*
    (sintheta3decay*cos(phin)*costheta*cos(phi)
     - sintheta3decay*sin(phin)*sin(phi)
     + costheta3decay*sintheta*cos(phi));
    daughter2P2 = en_p[1]*
    (sintheta3decay*cos(phin)*costheta*sin(phi)
     + sintheta3decay*sin(phin)*cos(phi)
     + costheta3decay*sintheta*sin(phi));
    daughter2P3 = en_p[1]*
    (- sintheta3decay*cos(phin)*sintheta
     + costheta3decay*costheta);

    daughter3P1 = - daughter1P1 - daughter2P1;     // pion 3
    daughter3P2 = - daughter1P2 - daughter2P2;
    daughter3P3 = - daughter1P3 - daughter2P3;
    return;
}
```

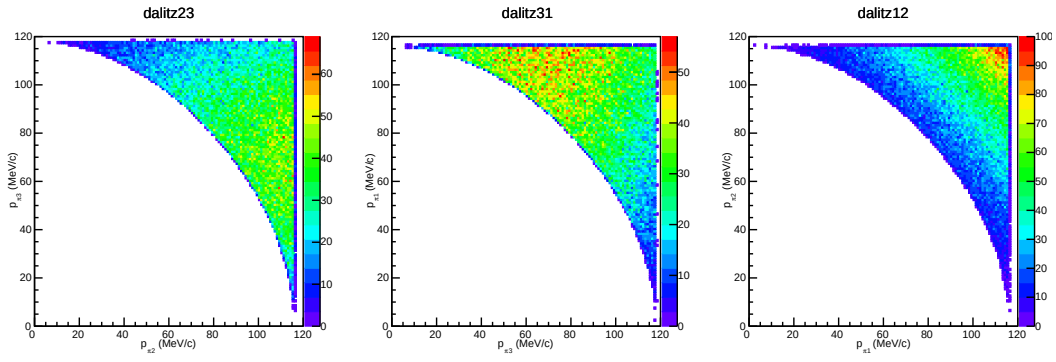


図 3　$K^+ \to \pi^0\pi^0\pi^+$ における π の運動量分布. 粒子 1, 2 は π^0, 粒子 3 は π^+ を示す.

3.2.2　$K \to \pi l \nu$

ν 静止系において $K \to \pi l \nu$ の娘粒子である l と π^0 の運動量は等しく、この運動量が最大値をとる。$m_\nu = 0$ を仮定して、K_{e3} の場合、

$$\begin{aligned} m_K &= \sqrt{{m_e}^2 + p^2} + \sqrt{{m_{\pi^0}}^2 + p^2} \\ m_K^2 - 2m_k\sqrt{{m_{\pi^0}}^2 + p^2} + {m_{\pi^0}}^2 + p^2 &= {m_e}^2 + p^2 \\ {m_{\pi^0}}^2 + p^2 &= \frac{m_k^2 - {m_{\pi^0}}^2 + {m_e}^2}{2m_k} \\ p &= \sqrt{\frac{m_k^2 - {m_{\pi^0}}^2 + {m_e}^2}{2m_k} - {m_{\pi^0}}^2} \\ &\sim 228.7 \text{ MeV}/c. \end{aligned} \tag{19}$$

$K_{\mu 3}$ の場合、

$$p \sim 215.5 \text{ MeV}/c. \tag{20}$$

３体崩壊を解くにおいて運動量の取りうる範囲をまず制限します。その次に確率密度から、崩壊頻度を決定します。$K \to \pi l \nu$ ($K_{e3},\ K_{\mu 3}$) の確率密度は以下で表せます。

$$\rho(t) \propto f^2(t) \left\{ A + B\,\xi(t) + C\,\xi^2(t) \right\} \tag{21}$$

ここで、使用されているパラメータは

$$t = {m_K}^2 + {m_{\pi^0}}^2 - 2m_K E_\pi, \tag{22}$$

$$\xi(t) = \xi_0 \left(1 + \lambda \frac{t}{{m_{\pi^0}}^2} \right), \tag{23}$$

$$f(t) = 1 + \lambda \frac{t}{{m_{\pi^0}}^2}, \tag{24}$$

$$E = \frac{{m_K}^2 + m_{\pi^0}^2 - m_l}{2m_K} - E_{\pi^0} \tag{25}$$

$$A = m_K\,(2E_l E_\nu - m_K E) + {m_l}^2 \left(\frac{E}{4} - E_\nu \right) \tag{26}$$

$$B = {m_l}^2 \left(E_\nu - \frac{E}{2} \right) \tag{27}$$

$$C = \frac{{m_l}^2 E}{4} \tag{28}$$

です。

ここで $f(t)$ を Form Factor (形状因子) という。

K_{e3} の場合、初期値として $\lambda = 0.0286,\ \xi_0 = -0.35$.

$K_{\mu 3}$ の場合、$\lambda = 0.033,\ \xi_0 = -0.35$.

sample9.cxx

```
void sample9(){
    gStyle->SetOptStat(0);
    double mK=493.677, mmu=105.6, mpi=139,mpi0=134,me=0.511,mnu=0;
    double pK[3]={0};

    dalitz23=new TH2F("dalitz23","dalitz23",100,0,250,100,0,300);
    dalitz31=new TH2F("dalitz31","dalitz31",100,0,250,120,0,300);

    for(int i=0;i<100000;i++){
        DalitzdecayKl3(mK,
                       mpi0,p[0][0],p[0][1],p[0][2],
                       me,p[1][0],p[1][1],p[1][2],
                       mnu,p[2][0],p[2][1],p[2][2]);
        dalitz23->Fill(sqrt(me**2 + p[1][0]**2+p[1][1]**2+p[1][2]**2),
                       sqrt(mpi0**2+ p[0][0]**2+p[0][1]**2+p[0][2]**2));

        DalitzdecayKl3(mK,
                       mpi0,p[0][0],p[0][1],p[0][2],
                       mmu,p[1][0],p[1][1],p[1][2],
                       mnu,p[2][0],p[2][1],p[2][2]);
        dalitz31->Fill(sqrt(mmu**2+ p[1][0]**2+p[1][1]**2+p[1][2]**2),
                       sqrt(mpi0**2+ p[0][0]**2+p[0][1]**2+p[0][2]**2));

        if(i%100==0)cout<<"."<<flush;
    }

    c1=new TCanvas("c1","",700,300);
    c1->Divide(2,1);

    c1->cd(1);
    dalitz23->SetTitle("K^{+}_{e3}");
    dalitz23->SetXTitle("Electron Energy (MeV)");
    dalitz23->GetXaxis()->CenterTitle();
    dalitz23->SetYTitle("Pion Energy (MeV)");
    dalitz23->GetYaxis()->CenterTitle();
    dalitz23->Draw("colz");
```

sample9.cxx の続き

```
  c1->cd(2);
    dalitz31->SetTitle("K^{+}_{#mu3}");
    dalitz31->SetXTitle("Muon Energy (MeV)");
    dalitz31->GetXaxis()->CenterTitle();
    dalitz31->SetYTitle("Pion Energy (MeV)");
    dalitz31->GetYaxis()->CenterTitle();
    dalitz31->Draw("colz");

    c1->Print("output.pdf");
    return;
}

void DalitzdecayKl3(
        double parentM,
        double daughter1M,double &daughter1P1,double &daughter1P2,double &daughter1P3,
        double daughter2M,double &daughter2P1,double &daughter2P2,double &daughter2P3,
        double daughter3M,double &daughter3P1,double &daughter3P2,double &daughter3P3
    ){
    double en_p[3]={0},en_E[3]={0};
    double pmax=
         sqrt(((parentM**2 + daughter1M**2 -
                  daughter2M**2)/2/parentM)**2 - daughter1M**2);
    double M;
    double costheta, sintheta, phi, sintheta3decay, phin, costheta3decay;
    double Epi,Epi_max,E,q2,F,Fmax,Xi,coeffA,coeffB,coeffC,RhoMax,Rho,Epi,El,Enu;

    double massK  = parentM;
    double massPi = daughter1M;
    double massL  = daughter2M;
    double massNu = daughter3M;

    double pLambda, pXi0;
    if(massL==0.511){ pLambda = 0.0286; pXi0 = -0.35;}//0.0297,-0.3
    else if (massL=105.6){pLambda = 0.033;pXi0 = -0.35;}//0.0297,0.17

    do{
        en_E[0] = gRandom->Uniform(daughter1M,sqrt(daughter1M**2+pmax**2));
        en_E[1] = gRandom->Uniform(daughter2M,sqrt(daughter2M**2+pmax**2));
        en_E[2] = parentM - en_E[0] - en_E[1];//nu

        en_p[0]=sqrt(en_E[0]**2 - daughter1M**2);
        en_p[1]=sqrt(en_E[1]**2 - daughter2M**2);
        en_p[2]=sqrt(en_E[2]**2 - daughter3M**2);
```

sample9.cxx の続き

```
        Epi=en_E[0];
        El =en_E[1];
        Enu=en_E[2];
        Epi_max = (massK*massK+massPi*massPi-massL*massL)/2.0/massK;
        E  = Epi_max - Epi;
        M = en_E[0]+en_E[1]+en_E[2];
        q2 = massK*massK + massPi*massPi - 2.0*massK*Epi;
        F = 1.0 + pLambda*q2/massPi/massPi;
        Fmax = 1.0;
        if (pLambda >0.0) Fmax = (1.0 + pLambda*(massK*massK/massPi/massPi+1.0));
        Xi = pXi0*(1.0 + pLambda*q2/massPi/massPi);
        coeffA = massK*(2.0*El*Enu-massK*E)+massL*massL*(E/4.0-Enu);
        coeffB = massL*massL*(Enu-E/2.0);
        coeffC = massL*massL*E/4.0;
        RhoMax = (Fmax*Fmax)*(massK*massK*massK/8.0);
        Rho = (F*F)*(coeffA + coeffB*Xi + coeffC*Xi*Xi);

        M=en_E[0]+en_E[1]+en_E[2];

        costheta3decay =
                 (en_E[2]**2 - daughter3M**2 - en_E[0]**2
                 + daughter1M**2 - en_E[1]**2 + daughter2M**2)/(2*en_p[0]*en_p[1]);

    }while(
           en_E[0] > sqrt(daughter1M**2+pmax**2) ||
           en_E[1] > sqrt(daughter2M**2+pmax**2) ||
           en_E[2] > sqrt(daughter3M**2+pmax**2) ||
           en_E[0] < daughter1M ||
           en_E[1] < daughter2M ||
           en_E[2] < daughter3M ||

           costheta3decay>1 ||
           costheta3decay<-1||

           gRandom->Uniform(0.00,1.00) > Rho/RhoMax||
           Rho/RhoMax < 0
    );
```

sample9.cxx の続き

```
    costheta = -1. +2.*gRandom->Uniform(0,1);     // particle 1
    sintheta = sqrt((1.-costheta)*(1.+costheta));
    phi = 2.*3.141592*gRandom->Uniform(0,1);
    daughter1P1 = en_p[0]*sintheta*cos(phi);
    daughter1P2 = en_p[0]*sintheta*sin(phi);
    daughter1P3 = en_p[0]*costheta;


    sintheta3decay=sqrt((1.+costheta3decay)*(1.-costheta3decay));    // particle 2
    phin=2.*3.141592*gRandom->Uniform(0,1);
    daughter2P1 = en_p[1]*
    (sintheta3decay*cos(phin)*costheta*cos(phi)
     - sintheta3decay*sin(phin)*sin(phi)
     + costheta3decay*sintheta*cos(phi));
    daughter2P2 = en_p[1]*
    (sintheta3decay*cos(phin)*costheta*sin(phi)
     + sintheta3decay*sin(phin)*cos(phi)
     + costheta3decay*sintheta*sin(phi));
    daughter2P3 = en_p[1]*
    (- sintheta3decay*cos(phin)*sintheta
     + costheta3decay*costheta);

    daughter3P1 = - daughter1P1 - daughter2P1;    // particle 3
    daughter3P2 = - daughter1P2 - daughter2P2;
    daughter3P3 = - daughter1P3 - daughter2P3;

    return;
}
```

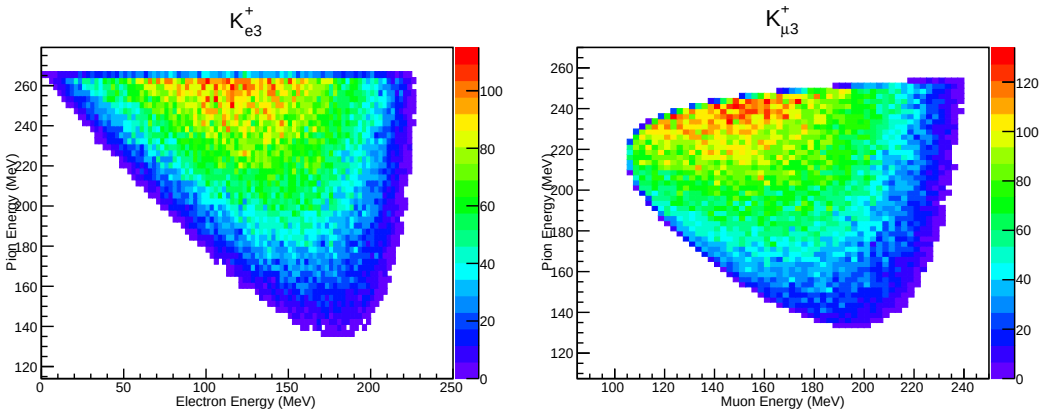


図 4　K_{l3} の Dalitz Plot. K_{e3} (左), $K_{\mu 3}$ (右)

3.3 静止 K 崩壊荷電粒子のスペクトラム

荷電 K^+ 中間子が崩壊した時のほとんどの崩壊過程をみて来ました。磁場を用いた実験で静止 K^+ から崩壊した荷電粒子は (1) プラスチックシンチレータ検出器でできた TOF start を通過し、(2) ドリフトチェンバーを通過し、磁場で曲がって (3) ドリフトチェンバーを通過し、(4) プラスチックシンチレータ検出器でできた TOF stop を通過するセットアップを考えます。ここでは、簡単のためトラッキングによる運動量分解能が 0.9%、TOF の時間分解能が 100 ps であると仮定して、磁場での軌道は 10 m に規格化されているとする。

K_{e2} と K_{e3} では e^+ が、$K_{\mu 2}$ と $K_{\mu 3}$ では μ^+ が、そして $K_{\pi 2}$ と $K^+ \to \pi^+\pi^+\pi^-$, $K^+ \to \pi^0\pi^0\pi^+$ では π^+ 放出される。これらの粒子の得られる運動量を描いてみましょう。

sample10.cxx

```
void sample10(){
    gStyle->SetOptStat(0);
    double mK=493.677, mmu=105.6, mpi=139,mpi0=134,me=0.511,mnu=0;
    double pmu,pnu,pe,ppi,ppi0,p[3][3]={0};
    double costheta,phi,sintheta,rndm;

    double Gamma1=1.58e-5; //Ke2
    double Gamma2=63.55e-2;//Kmu2
    double Gamma3=5.07e-2; //Ke3
    double Gamma4=3.35e-2; //Kmu3
    double Gamma5=20.66e-2;//Kpi2
    double Gamma6=1.76e-2; //Kpi3plus
    double Gamma7=5.59e-2; //Kpi3zero

    double dmy;
    double resoP=0.9e-2;

    total=new TH1F("total","total",600,0,300);
    Ke2=new TH1F("Ke2","Ke2",600,0,300);
    Kmu2=new TH1F("Kmu2","Kmu2",600,0,300);
    Ke3=new TH1F("Ke3","Ke3",600,0,300);
    Kmu3=new TH1F("Kmu3","Kmu3",600,0,300);
    Kpi2=new TH1F("Kpi2","Kpi2",600,0,300);
    Kpi3plus=new TH1F("Kpi3plus","Kpi3plus",600,0,300);
    Kpi3zero=new TH1F("Kpi3zero","Kpi3zero",600,0,300);

    gSystem->Load("sample6.cxx");
    gSystem->Load("sample8.cxx");
    gSystem->Load("DalitzdecayKto3piZero.cxx");
    gSystem->Load("sample9.cxx");
```

sample10.cxx の続き

```
    for(int i=0;i<10000000;i++){
        rndm=gRandom->Uniform(0,1);
        if(rndm<=Gamma1){
            twobodydecay(mK, me, pe, 0, pnu);
            dmy=gRandom->Gaus(pe,pe*resoP);
            total->Fill(dmy);
            Ke2->Fill(dmy);
        }
        else if(rndm>Gamma1 && rndm<=Gamma1+Gamma2){
            twobodydecay(mK, mmu, pmu, 0, pnu);
            dmy=gRandom->Gaus(pmu,pmu*resoP);
            total->Fill(dmy);
            Kmu2->Fill(dmy);
        }
        else if(rndm>Gamma1+Gamma2
                && rndm<=Gamma1+Gamma2+Gamma3){
            DalitzdecayKl3(mK,
                           mpi0,p[0][0],p[0][1],p[0][2],
                           me,p[1][0],p[1][1],p[1][2],
                           mnu,p[2][0],p[2][1],p[2][2]);
            pe=sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2);
            dmy=gRandom->Gaus(pe,pe*resoP);
            total->Fill(dmy);
            Ke3->Fill(dmy);
        }
        else if(rndm>Gamma1+Gamma2+Gamma3
                && rndm<=Gamma1+Gamma2+Gamma3+Gamma4){
            DalitzdecayKl3(mK,
                        mpi0,p[0][0],p[0][1],p[0][2],
                        mmu,p[1][0],p[1][1],p[1][2],
                        mnu,p[2][0],p[2][1],p[2][2]);
            pmu=sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2);
            dmy=gRandom->Gaus(pmu,pmu*resoP);
            total->Fill(dmy);
            Kmu3->Fill(dmy);
        }
        else if(rndm>Gamma1+Gamma2+Gamma3+Gamma4&&
                rndm<=Gamma1+Gamma2+Gamma3+Gamma4+Gamma5){
            twobodydecay(mK, mpi, ppi, mpi0, ppi0);
            dmy=gRandom->Gaus(ppi,ppi*resoP);
            total->Fill(dmy);
            Kpi2->Fill(dmy);
        }
```

sample10.cxx の続き

```
    else if(rndm>Gamma1+Gamma2+Gamma3+Gamma4+Gamma5
            && rndm<=Gamma1+Gamma2+Gamma3+Gamma4+Gamma5+Gamma6){
        DalitzdecayKto3piPlus(mK,
                      mpi,p[0][0],p[0][1],p[0][2],
                      mpi,p[1][0],p[1][1],p[1][2],
                      mpi,p[2][0],p[2][1],p[2][2]);
        ppi=sqrt(p[0][0]**2+p[0][1]**2+p[0][2]**2);
        dmy=gRandom->Gaus(ppi,ppi*resoP);
        total->Fill(dmy);
        Kpi3plus->Fill(dmy);
    }
    else if(rndm>Gamma1+Gamma2+Gamma3+Gamma4+Gamma5+Gamma6
            && rndm<=Gamma1+Gamma2+Gamma3+Gamma4+Gamma5+Gamma6+Gamma7){
        DalitzdecayKto3piZero(mK,
                      mpi0,p[0][0],p[0][1],p[0][2],
                      mpi0,p[1][0],p[1][1],p[1][2],
                      mpi,p[2][0],p[2][1],p[2][2]);
        ppi=sqrt(p[2][0]**2+p[2][1]**2+p[2][2]**2);
        dmy=gRandom->Gaus(ppi,ppi*resoP);
        total->Fill(dmy);
        Kpi3zero->Fill(dmy);
    }
    if(i%10000==0)cout<<"."<<flush;
}

c1=new TCanvas("c1","",500,300);
c1->SetLogy();
total->SetTitle("");
total->SetXTitle("Momenum p (MeV/c)");
total->GetXaxis()->CenterTitle();
total->SetLineColor(1);
total->SetFillColor(1);
total->SetFillStyle(3003);
total->Draw();
Ke2->SetLineColor(2);
Ke2->Draw("same");
Kmu2->SetLineColor(3);
Kmu2->Draw("same");
Ke3->SetLineColor(4);
Ke3->Draw("same");
Kmu3->SetLineColor(5);
Kmu3->Draw("same");
```

sample10.cxx の続き

```
    Kpi2->SetLineColor(6);
    Kpi2->Draw("same");
    Kpi3plus->SetLineColor(7);
    Kpi3plus->Draw("same");
    Kpi3zero->SetLineColor(8);
    Kpi3zero->Draw("same");

    c1->Print("output.pdf");
    return;
}
```

2 体崩壊における荷電粒子のスペクトラムはピーク構造なので、K_{e2}, $K_{\mu 2}$, $K_{\pi 2}$ はわかりやすいですね。3 体崩壊で放出される荷電粒子は連続的な運動量を持つことが視覚的にも理解できると思います。$K_{\mu 2}$ の裾に K_{e2} が隠れているため、運動量分布から K_{e2} を観測することは難しいことがわかります。e/μ の粒子識別によって K_{e2} ピークを見やすくできるはずです。

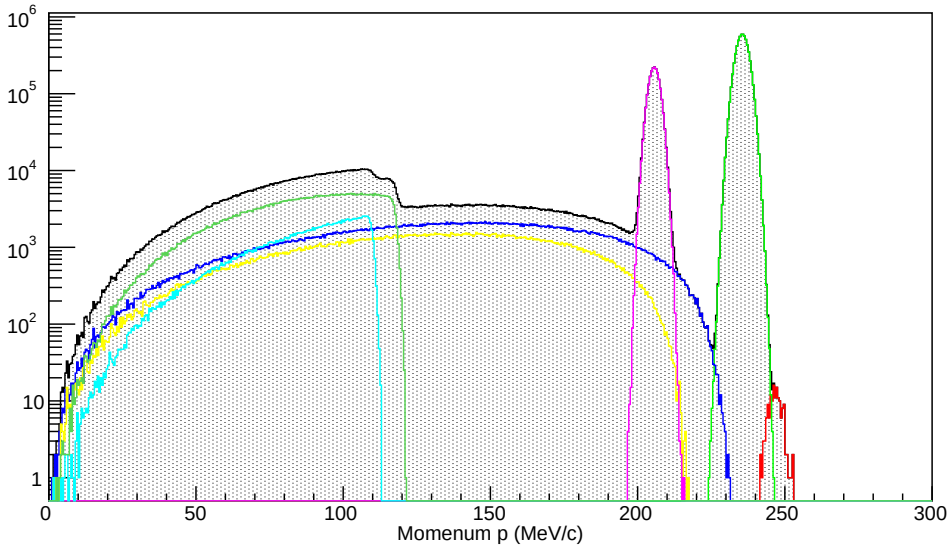


図 5　静止 K^+ 崩壊 運動量スペクトラム. K_{e2} (赤), $K_{\mu 2}$ (緑), K_{e3} (青), $K_{\mu 3}$ (黄), $K_{\pi 2}$ (マゼンタ), $K^+ \to \pi^+\pi^+\pi^-$ (暗緑), $K^+ \to \pi^0\pi^0\pi^+$ (シアン).

TOF を用いて $e^+/\mu^+/pi^+$ の識別をしていきます。運動量 p =200–250 MeV/c の速度比 β はそれぞれ

$$\beta_e = 1 - (2.089 \sim 3.264) \times 10^{-6}$$
$$\beta_\mu = 0.884 \sim 0.921$$
$$\beta_\pi = 0.821 \sim 0.874$$

なので、10 m 走ると $t = L/\beta$ だけ時間がかかる。このことから、運動量と時間差の分布を解析していきましょう。

sample11.cxx

```
void sample11(){
    gStyle->SetOptStat(0);
    double mK=493.677, mmu=105.6, mpi=139,mpi0=134,me=0.511,mnu=0;
    double pmu,pnu,pe,ppi,ppi0,p[3][3]={0},P;
    double costheta,phi,sintheta,rndm;
    double beta,c=3.0e8,dt;

    double Gamma1=1.58e-5; //Ke2
    double Gamma2=63.55e-2;//Kmu2
    double Gamma3=5.07e-2; //Ke3
    double Gamma4=3.35e-2; //Kmu3
    double Gamma5=20.66e-2;//Kpi2
    double Gamma6=1.76e-2; //Kpi3plus
    double Gamma7=5.59e-2; //Kpi3zero

    double dmy,dmy2;
    double resoP=0.9e-2;//%
    double resoT=0.1;//ns

    file=new TFile("kaon.root","recreate");
    total=new TH2F("total","total",300,0,300,300,20,120);

    kaon=new TTree("kaon","kaon");
    kaon->Branch("P",&P,"P/D");
    kaon->Branch("dt",&dt,"dt/D");

    gSystem->Load("sample6.cxx");
    gSystem->Load("sample8.cxx");
    gSystem->Load("DalitzdecayKto3piZero.cxx");
    gSystem->Load("sample9.cxx");

    for(int i=0;i<10000000;i++){
        rndm=gRandom->Uniform(0,1);
        if(rndm<=Gamma1){
            twobodydecay(mK, me, pe, 0, pnu);
            P=gRandom->Gaus(pe,pe*resoP);
            dmy2=10/c/(pe/sqrt(me**2 +pe**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }
```

sample11.cxx の続き

```
        else if(rndm>Gamma1 && rndm<=Gamma1+Gamma2){
            twobodydecay(mK, mmu, pmu, 0, pnu);
            P=gRandom->Gaus(pmu,pmu*resoP);
            dmy2=10/c/(pmu/sqrt(mmu**2 +pmu**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }

        else if(rndm>Gamma1+Gamma2
                && rndm<=Gamma1+Gamma2+Gamma3){
            DalitzdecayKl3(mK,
                           mpi0,p[0][0],p[0][1],p[0][2],
                           me,p[1][0],p[1][1],p[1][2],
                           mnu,p[2][0],p[2][1],p[2][2]);
            pe=sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2);
            P=gRandom->Gaus(pe,pe*resoP);
            dmy2=10/c/(pe/sqrt(me**2 +pe**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }
        else if(rndm>Gamma1+Gamma2+Gamma3
                && rndm<=Gamma1+Gamma2+Gamma3+Gamma4){
            DalitzdecayKl3(mK,
                       mpi0,p[0][0],p[0][1],p[0][2],
                       mmu,p[1][0],p[1][1],p[1][2],
                       mnu,p[2][0],p[2][1],p[2][2]);
            pmu=sqrt(p[1][0]**2+p[1][1]**2+p[1][2]**2);
            P=gRandom->Gaus(pmu,pmu*resoP);
            dmy2=10/c/(pmu/sqrt(mmu**2 +pmu**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }
        else if(rndm>Gamma1+Gamma2+Gamma3+Gamma4&&
                rndm<=Gamma1+Gamma2+Gamma3+Gamma4+Gamma5){
            twobodydecay(mK, mpi, ppi, mpi0, ppi0);
            P=gRandom->Gaus(ppi,ppi*resoP);
            dmy2=10/c/(ppi/sqrt(mpi**2 +ppi**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }
```

sample11.cxx の続き

```
        else if(rndm>Gamma1+Gamma2+Gamma3+Gamma4+Gamma5
                && rndm<=Gamma1+Gamma2+Gamma3+Gamma4+Gamma5+Gamma6){
            DalitzdecayKto3piPlus(mK,
                          mpi,p[0][0],p[0][1],p[0][2],
                          mpi,p[1][0],p[1][1],p[1][2],
                          mpi,p[2][0],p[2][1],p[2][2]);
            ppi=sqrt(p[0][0]**2+p[0][1]**2+p[0][2]**2);
            P=gRandom->Gaus(ppi,ppi*resoP);
            dmy2=10/c/(ppi/sqrt(mpi**2 +ppi**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }
        else if(rndm>Gamma1+Gamma2+Gamma3+Gamma4+Gamma5+Gamma6
                && rndm<=Gamma1+Gamma2+Gamma3+Gamma4+Gamma5+Gamma6+Gamma7){
            DalitzdecayKto3piZero(mK,
                          mpi0,p[0][0],p[0][1],p[0][2],
                          mpi0,p[1][0],p[1][1],p[1][2],
                          mpi,p[2][0],p[2][1],p[2][2]);
            ppi=sqrt(p[2][0]**2+p[2][1]**2+p[2][2]**2);
            P=gRandom->Gaus(ppi,ppi*resoP);
            dmy2=10/c/(ppi/sqrt(mpi**2 +ppi**2))*1e9;//ns
            dt=gRandom->Gaus(dmy2,dmy2*resoT);
            total->Fill(P,dt);
        }
        else continue;
        kaon->Fill();
        if(i%10000==0)cout<<"."<<flush;
    }

    c1=new TCanvas("c1","",500,300);
    total->SetTitle("");
    total->SetXTitle("Momenum p (MeV/c)");
    total->GetXaxis()->CenterTitle();
    total->SetYTitle("#Delta t (ns)");
    total->GetYaxis()->CenterTitle();
    total->Draw("colz");

    kaon->Write();
    c1->Print("output.pdf");
    return;
}
```

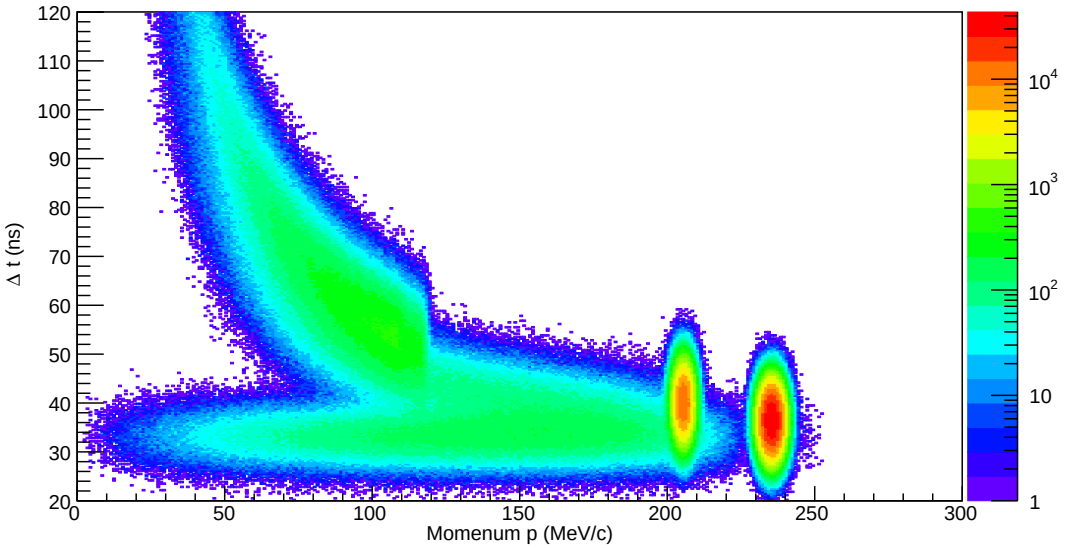


図 6　時間分解能 100 ps、運動量分解能 0.9% におけるスペクトラム分布.

図 6 に運動量と時間差のスキャッタ plot を示します。z 軸は Log スケールで表示しています。235 MeV/c あたりに存在する $K_{\mu 2}$ の右にちょこんと存在しているのが K_{e2} なのですが、これではわかりませんね。時間分解能を向上すれば見えてくることは明らかで、例えば、20 ps に設定してもう一度見て見ましょう。

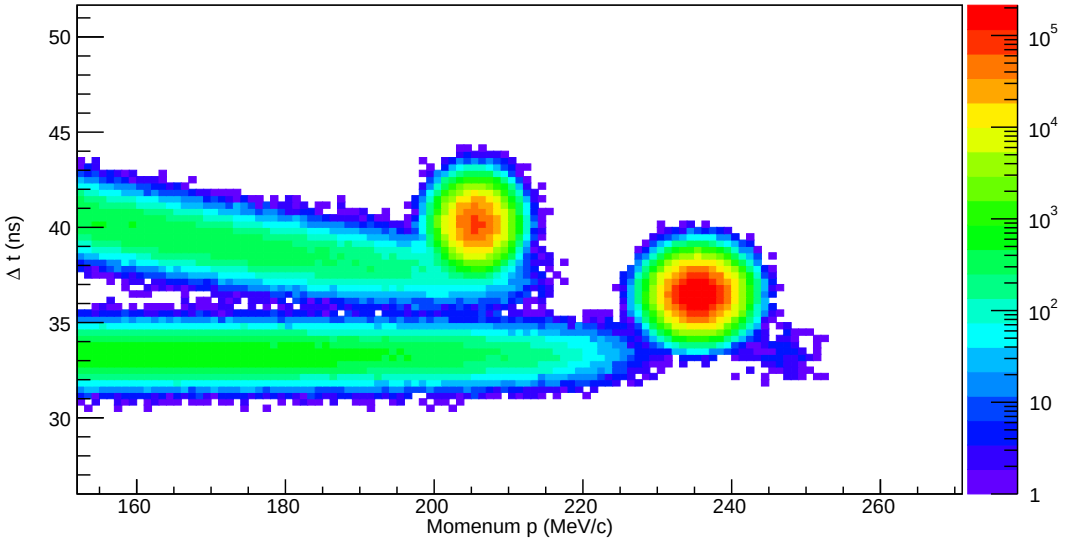


図 7　時間分解能 20 ps、運動量分解能 0.9% におけるスペクトラム分布.

図 7 は TOF 時間分解能 20 ps における運動量と時間差の分布です。こうなってくると 245 MeV/c あたりで K_{e2} が分離しました。さて、時間差閾値をどこに設定するかで $K_{\mu 2}$ によるなだれ込みの量が変わって来ます。ROOT file にデータを格納しましたので、実際に解析していきましょう。

sample12.cxx

```
void sample12(){
    gStyle->SetOptStat(0);
    double P,dt;
    double thre=33;
    char name[100];
    file=new TFile("kaon.root");
    kaon=(TTree*)file->Get("kaon");
    kaon->SetBranchAddress("P",&P);
    kaon->SetBranchAddress("dt",&dt);
    TH1F*mom[10];
    for(int i=0;i<8;i++){
        sprintf(name,"mom%d",i);
        mom[i]=new TH1F(name,name,300,0,300);
    }

    for(int i=0;i<kaon->GetEntries();i++){
        kaon->GetEntry(i);
        for(int j=0;j<8;j++){
            if(dt<thre+0.2*j) mom[j]->Fill(P);
        }
        if(i%100000==0)cout<<"."<<flush;
    }

    c1=new TCanvas("c1","",500,300);
    mom[0]->SetTitle("");
    mom[0]->SetXTitle("Momenum p (MeV/c)");
    mom[0]->GetXaxis()->CenterTitle();
    mom[0]->GetXaxis()->SetRangeUser(220,260);
    mom[0]->GetYaxis()->SetRangeUser(0,50);
    mom[0]->Draw();
    for(int i=1;i<4;i++){
        mom[i]->SetLineColor(i+1);
        mom[i]->Draw("same");
    }
```

sample12.cxx の続き

```
    text=new TLatex(247,20,"K_{e2}");
    text->SetTextSize(0.05);
    text->Draw("same");
    text=new TLatex(235,45,"K_{#mu2}");
    text->SetTextSize(0.05);
    text->Draw("same");
    text=new TLatex(225,5,"K_{e3}");
    text->SetTextSize(0.05);
    text->Draw("same");

    c1->Print("output.pdf");
    return;
}
```

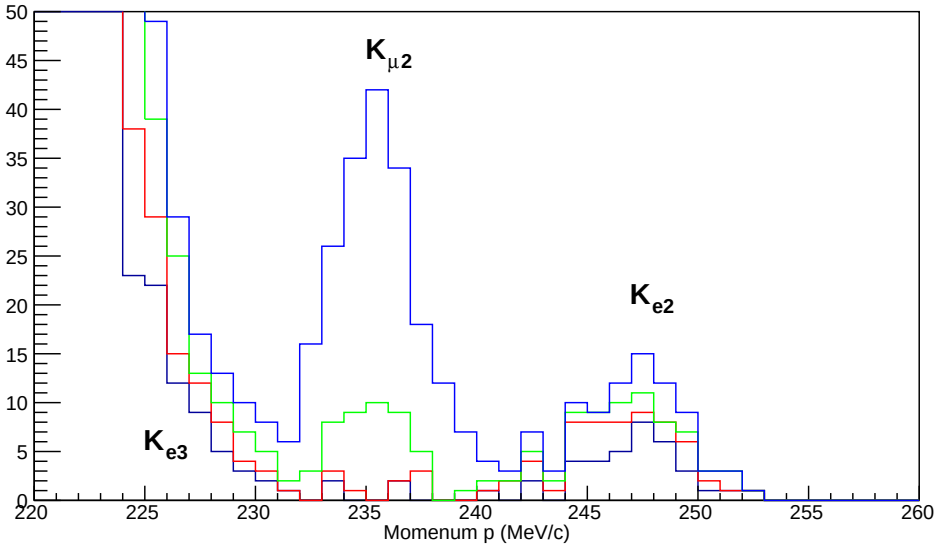


図 8　TOF 事象選択 $\Delta t < 33,\ 33.5,\ 34,\ 34.5$ ns における運動量分布.

TOF の時間差の事象選択を $\Delta t < 34.5,\ 34,\ 33.5,\ 33$ ns ときつくしていくと、次第に $K_{\mu 2}$ が減っていく。ちょうど $\Delta t < 33.5$ ns の時 (赤)、$K_{\mu 2}$ のなだれ込みを優位に抑制でき、K_{e3} と K_{e2} のスペクトラムのみが見えるようになりました。ここまですれば K_{e2} を観測し、Ke2 の事象数を数えることができます。つまり分岐比の測定ができることを意味します。分岐比 $\Gamma(K_{\mu 2})$ と $\Gamma(K_{e2})$ の比を測定することができます。

もしこの値が理論値より統計的優位に離れているなら、暗に新しい物理を示唆します。

4　まとめ

本稿では静止 K による崩壊を中心に模型作成とデータ解析を実施しました。時間分解能 100 ps では、運動量と時間差の分布で $K_{\mu 2}$ のテイルがまだまだ K_{e2} に雪崩込んで見えません。20 ps の TOF があれば粒子識別は TOF だけで K_{e2} を観測することができるが、ちょっと現実的ではありません。実際、$K_{\mu 2}$ のテイルから K_{e2} を抜き出すためには、さらに γ 線のデータを駆使する必要があります。なぜなら、e^+ が標的内もしくは磁場内で制動放射した場合、運動量が下がり $K_{\mu 2}$ の領域になだれ込み、さらに $K^+ \to e^+ \nu_e \gamma$ の 3 体崩壊が分岐比 9.5×10^{-6} で入り込むため、誤

認する可能性を持つからです。$K^+ \to \mu^+\nu_\mu\gamma$ は分岐比 1.33×10^{-5} なので γ 線を放出して TOF で事象選択すれば無視できる水準に抑制できる。e^+ の制動放射によるエネルギー損失はどうしようもないので、実際に実験をするときは 10^5 程度大きい分岐比を持つ $K_{\mu 2}$ を徹底的に排除するために TOF 以外に粒子識別を 2 重 3 重にかける必要があります。

あとがき

ROOT 入門 第 2 弾を書きましたが、第 1 弾が 2014 年なので 3 年越しなんです。今回は ROOT を勉強するというより、C/C++ を勉強した感が強い気がします。そして何かしらの物理を入れて入門らしくしたいな〜という気持ちもありましたので、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ました。執筆していてなんですが、自分にもいい勉強になった気がします。2 体崩壊は簡単なのに 3 体崩壊になると途端に難しくなりました。GEANT4 のソースを参考にして Phase Space や Dalitz Decay など組み込んでいます。分岐比は PDG を参考にしています。それでも物理が間違っているとか、お偉い方々に見られると指摘される箇所があるかもしれません。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）。