μ-PIC による高速中性子線イメージング

栗本 真志　　松岡 佳大

2010 年 3 月 26 日

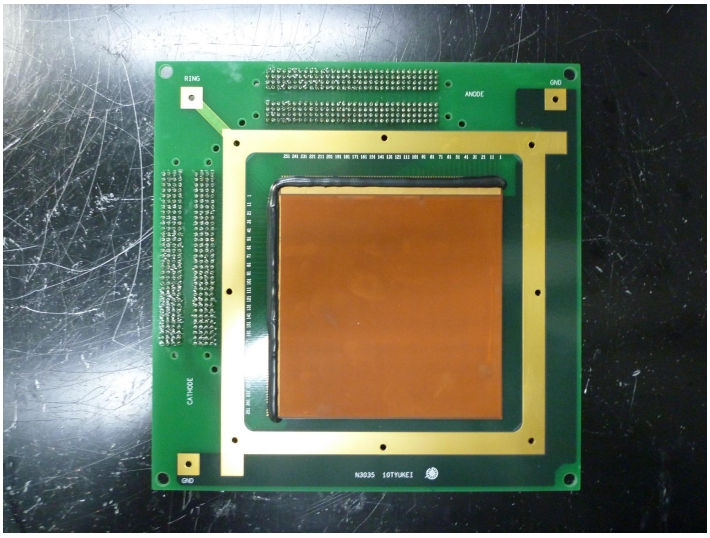
概要

非荷電粒子である中性子は、陽子と衝突しやすく、その際の散乱方向は前方に偏ることが一般的に知られている。今回、私たちは 2 次元・3 次元イメージングが可能なμ-PIC を用いて、中性子-陽子散乱を捉え、そのことを確かめた。
最初にμ-PIC に慣れながら性能調査を行い、次に 2 次元イメージングを用いて中性子-陽子散乱を捉えるところまで行った。

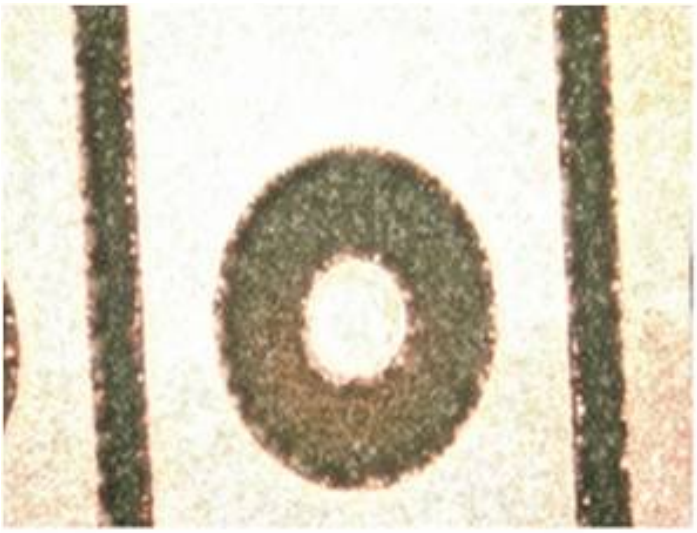
目次

1　μ-PIC について

今回使用したμ-PIC は、京都大学宇宙線研究室で開発されたガス検出器である。位置分解能が高く、放電に対して安定である、という特徴を持つ。比例計数管をピクセル状に 0〜255 の 256 ｃｈ並べており、2 次元イメージングを行うことができる。その構造は図 1 のようになっている。



(a)



(b)

図 1　μ-PI C
(a) の茶色い正方形の部分がμ-PIC である。10cm × 10cm の領域に
(ｂ) のような小型の比例計数管が 256 × 256 個並んでいる。

次に、μ-PIC による放射線の検出原理について説明する（図 2)。今回の実験においてはガス封じ切り容器を用いた (図 3)。中にはアルゴン＋エタンガス（エタン 7.74 %) が封入されており、ドリフト電圧として 3220V がかかっている。ガス容器内に放射線が入射してくると、光電吸収により、陽イオン・電子雲対が形成される。電子雲がドリフト電圧によってμ-PIC 検出面のアノードへと動いて行き、アノード・カソード間にかけられた高電圧により電子の雪崩増幅が起きる。アノードでは電子の負の信号、カソードでは正の信号がそれぞれ読みだされる。また、アノード・カソードについて同時刻にきた信号から、二次元位置情報を得ることができる。

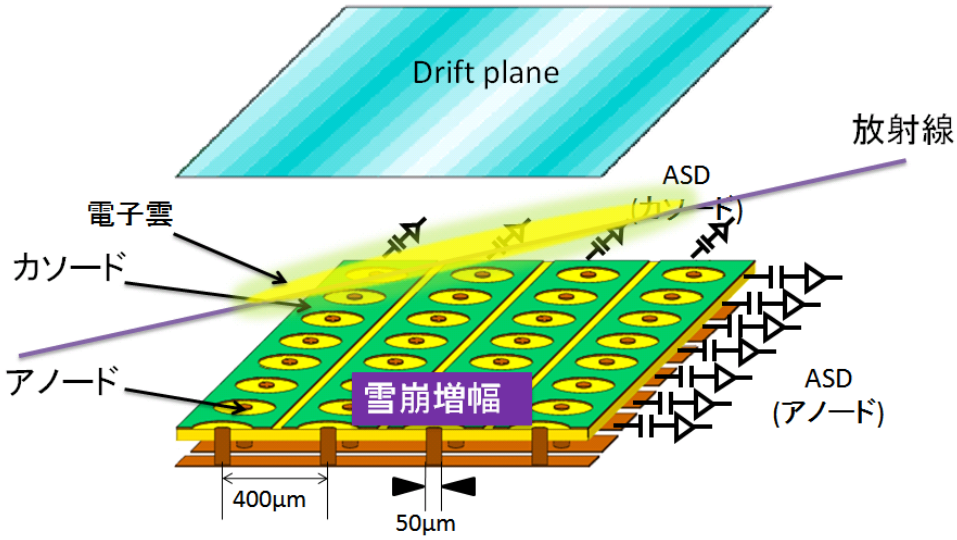


図 2　検出原理

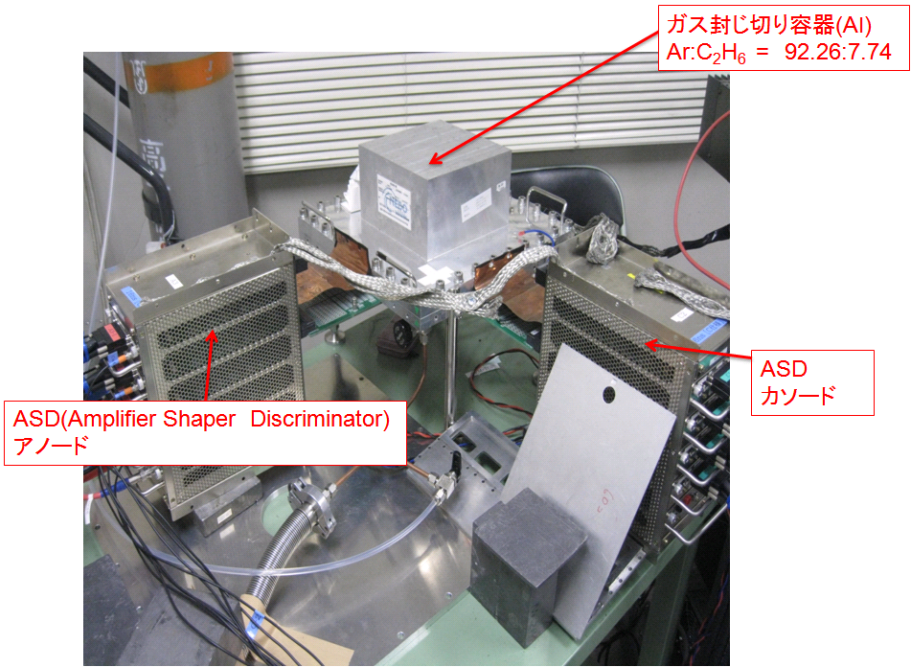
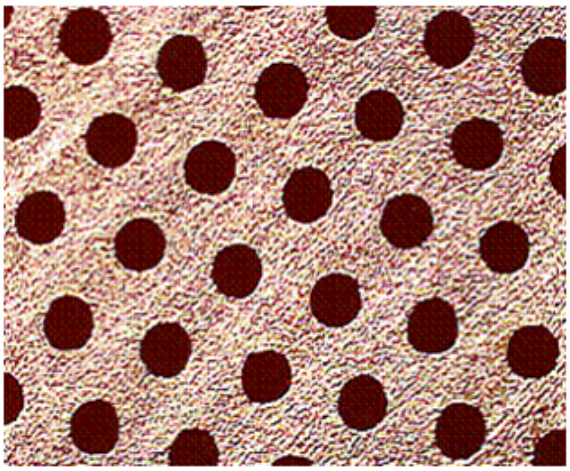


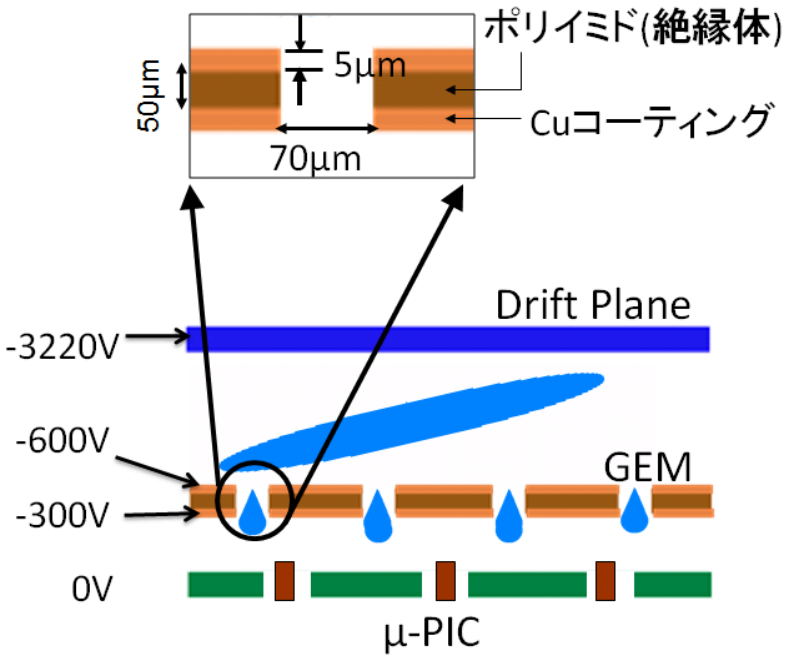
図 3　ガス封じ切り容器
ASD（Amplifier Shaper Discriminator）はプリアンプとしての役割を果たす（増幅率 160 倍）

※ GEM について

ガス封じ切り容器内には、GEM(Gas Electron Multiplier) という補助的な増幅装置がある（図4)。ポリイミド（絶縁体）を両面に薄い銅でコーティングしたもので、(a) のように 140 μ m 間隔で穴があいている。容器内でのその配置は (b) のようになっており、互いの面に電位差をあたえることで、電子雲が通過する際に雪崩増幅を引き起こす。



(a)



(b)

図 4　GEM

2　エネルギー較正・増幅率

まず最初にμ-PIC に慣れる段階として、エネルギー較正を行い、増幅率を調べた。

2.1　実験原理

線源は ^{109}Cd（22.2keV）と ^{133}Ba(31.0keV) を使用した。アノードの ASD において 256ch を 64ch ずつ、0ch〜3ch の 4 つに束ねて測定した。

スペクトルの取得には FADC(Flash ADC) を用いた。その回路及び概念図は図 5, 図 6 のようになっている。Discriminator の閾値を超えた信号が入ってきたとき約 8000nsec Delay をかけ、STOP させる。そこから　8000nsec 前の約 4800nsec のデータが読みだされる。ここから信号に対する電荷量を求めていく。最初の 1600nsec の平均をとることで Base Line を決定する。これで積分したのでは線源からの信号とは関係ないノイズも計算に含まれてしまうため、ノイズをカットする方法を考える必要がある。まず、既に決定した Base Line から約 7mV(FADC の 1.9channnel 分) のところにプログラムで Threshold をかける。さらに、欲しいエリアの部分は連続してその Threshold を超えたものであるとして、3 点以上連続して超えたもののみをプログラムで抽出するという方法をとった。

ここで、電荷量は、この Area の面積が FADC の総 channnel 数になることから、

$$Q = Area \times 3.9mV(FADC1channel) \times 16ns(FDAC1clock)/50\ \Omega \tag{1}$$

で求めることができる。

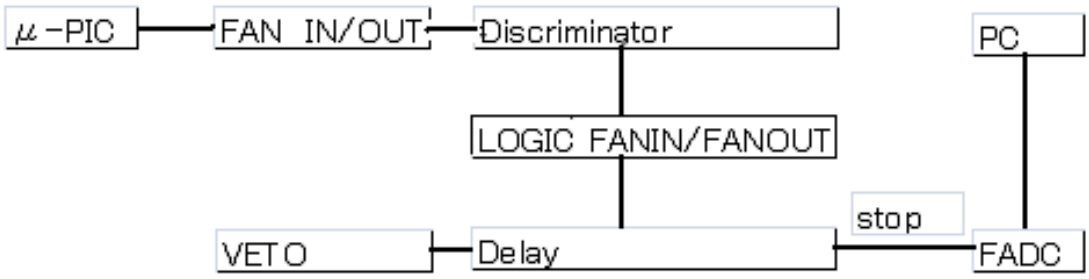


図 5　スペクトル取得回路

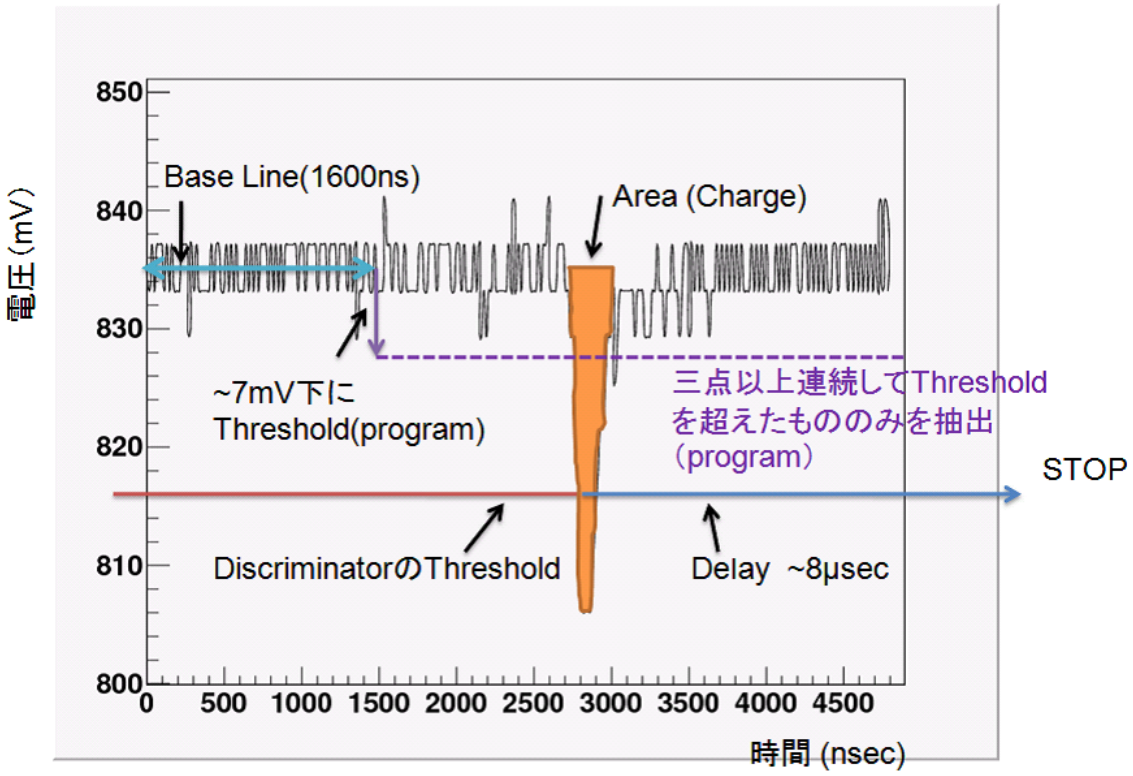
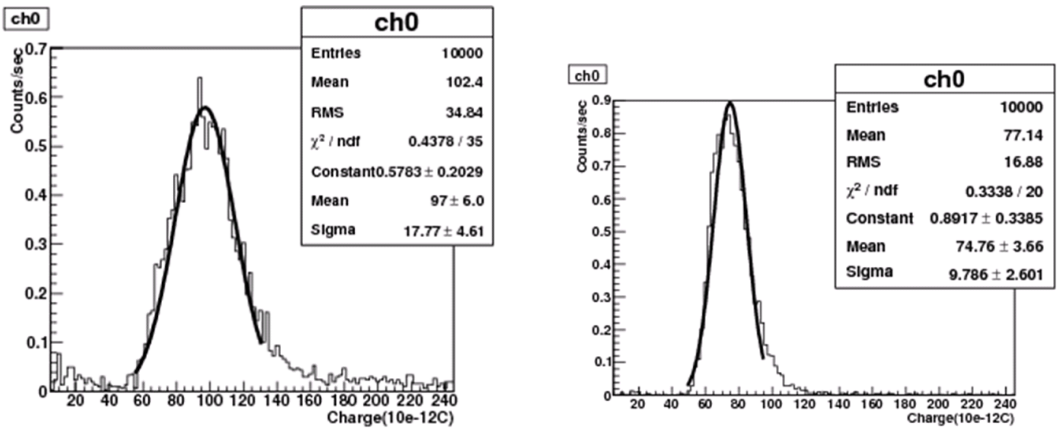


図 6　スペクトル取得概念図

2.2　エネルギー較正

実際に、アノードが 415V、ch0 において得られたスペクトルは図 7 のようになった。



(a) ^{133}Ba　　(b) ^{109}Cd

図 7　415V、ch0 のときのスペクトル

これら 2 つの線源によってエネルギー較正を行った (図 8)。原理的には 0channel で 0pC となるので、これを 2 つのデータと合わせて 3 点で行っている。もう少し増幅率がよければもっと多くのデータを得られたのだが（^{55}Fe 等)、今回は 3 点で行った。

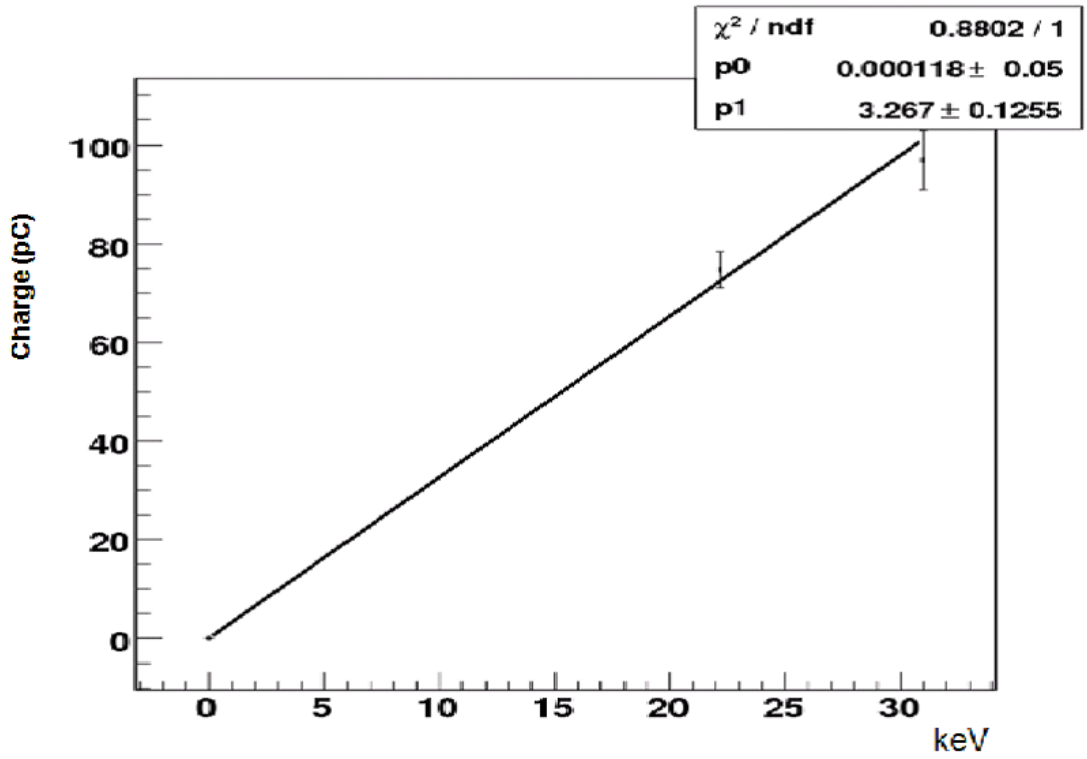


図 8　エネルギー較正

2.3 増幅率

ここまででエネルギー較正を行うことができたので、μ-PIC + GEM の増幅率を求める。ここでもアノード 415V、ｃｈ 0 について考える。アルゴンとエタンの混合ガスの W 値が 23.6eV、図 8 の傾きが 3.27pC/keV、素電荷量を 1.60×10^{-7}pC とすると、全体の増幅率は

$$3.27\mathrm{pC/keV} \times \frac{23.6 \times 10^{-3} keV}{1{,}60 \times 10^{-7} pC} = 4.82 \times 10^{5}$$

となる。ここで ASD の増幅率を 160（既知）とすると、μ-PIC + GEM の増幅率は

$$4.82 \times 10^{5} / 160 = 3.01 \times 10^{3}$$

となる。それぞれのｃｈについてアノード電圧ごとに増幅率を求めると、図 9 のようになる。縦軸は増幅率の対数をとっている。電圧に対して増幅率が指数関数的に大きくなっているのがわかる。

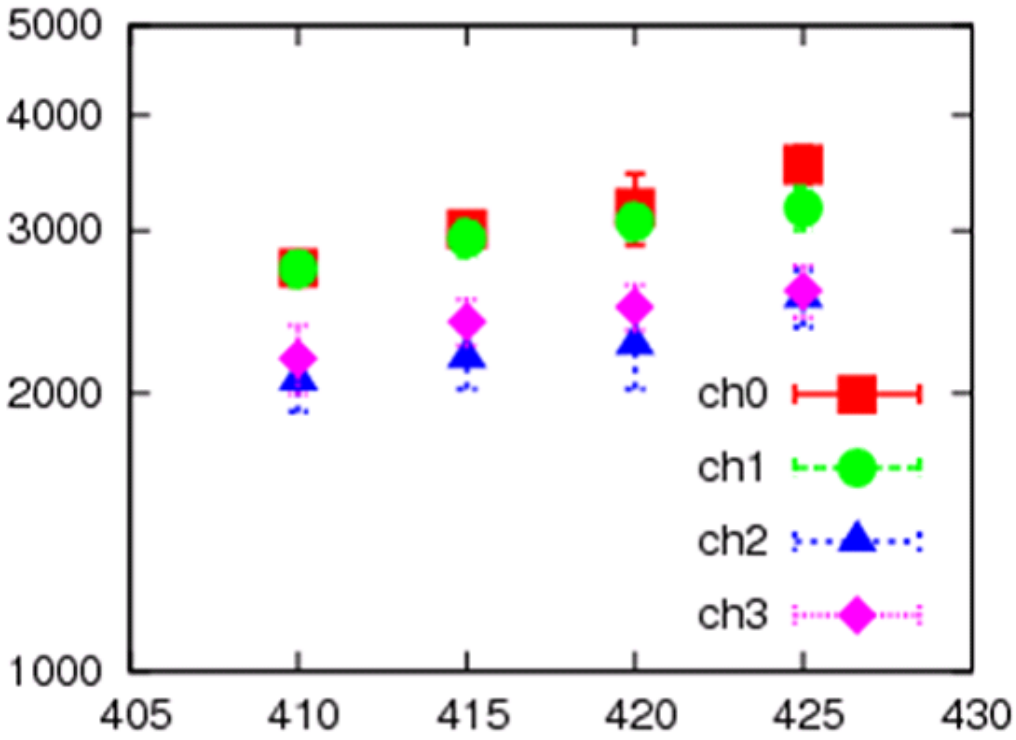
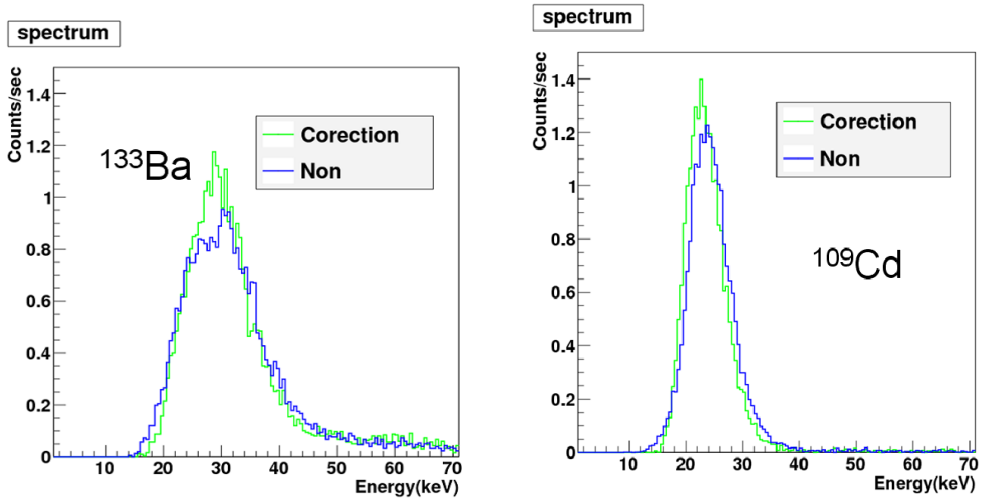


図 9　ｃｈごとの増幅率

2.4 ゲイン補正

1 イベントにおいて、1 つの ch ですべてエネルギーを落とすことなく 2 つ以上のｃｈにまたがって信号が見えている場合がある（全体の 8% くらい)。図 9 を見ると、ch0-ch3 まででそれぞれ様々な増幅率をもっていることがわかる。よってそれぞれの増幅率を加味したスペクトルを得る必要がある。 これをゲイン補正と呼び、ゲイン補正を行う前後でエネルギースペクトルは図 10 のようになった。



(a) ^{133}Ba　　(b) ^{109}Cd

図 10　ゲイン補正前後のスペクトル

ゲイン補正を行った結果、エネルギー分解能 (FWHM) は ^{133}Ba で 52.6% から 47.7% に、^{109}Cd で 32.8% から 28.6% に改善することができた。

3　2 次元イメージング

これまででμ-PIC のガス検出器としての性能を調べることができたので、μ-PIC の特徴である 2 次元イメージングについて考える。

3.1　channel と位置の対応

まず、2 次元イメージングを行うための位置情報取得の方法について述べる（図 11）。2 次元イメージングを行うためには、新たに Encoder Board と Memory Board が必要となる。アノードとカソードの ASD からそれぞれ threshold を超えた信号が 256ch ずつ Encoder Board に送られる。この実験では 0 から 255ch の信号は束ねられることなくそれぞれ情報をもって送られている。このうち、アノードとカソードが同期したものが Memory Board を介し、PC へ送られる。

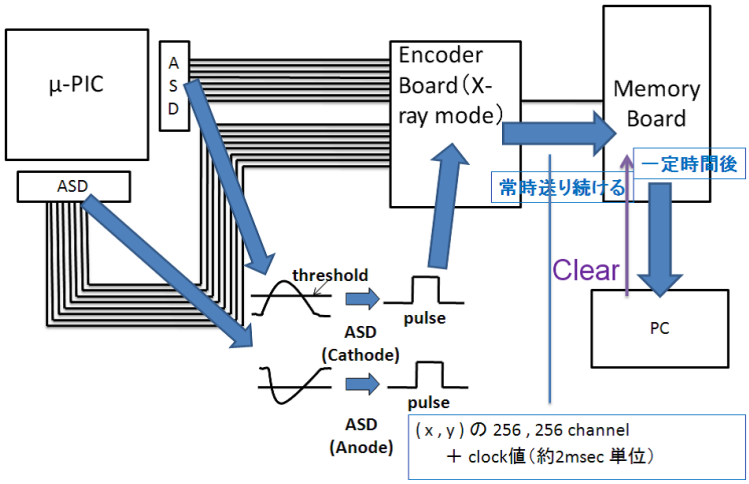


図 11　2 次元位置情報取得の概念図

この回路で ^{133}Ba をガス容器の上部中央において全面照射したときの 2 次元ヒストグラムは図 12 のようになった。ストリップが抜けているところは、放電が見られた箇所のジャンパーを外しているためである。場所により検出効率にかなり差があることが見られた。

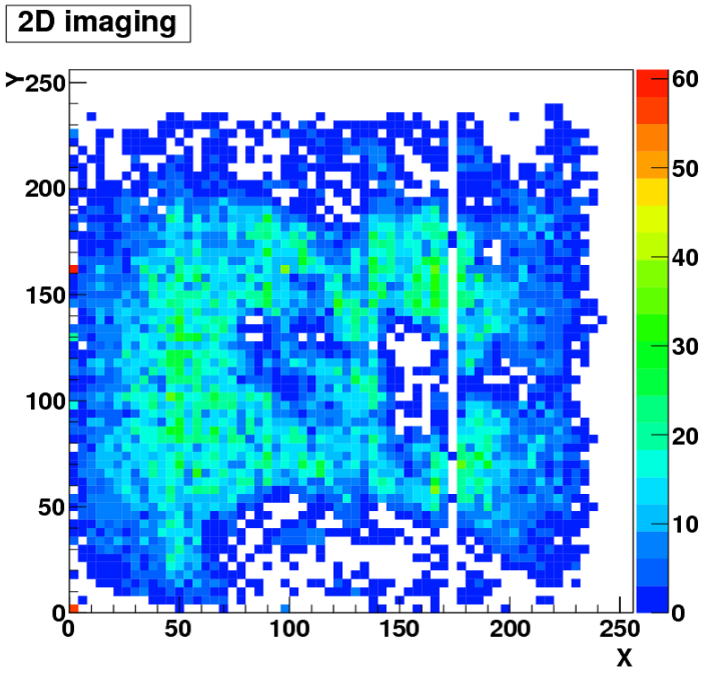


図 12　^{133}Ba による全面照射

さて、これから 2 次元イメージングや 3 次元トラッキングについて考えるためにはガス容器の中で μ-PIC がどの位置にあるのかを知る必要がある。ここで、今回次のような実験を行った。

^{133}Ba をガス容器の上に図 13、図 14(a) のように鉛で入射放射線を絞った状態で置いた。

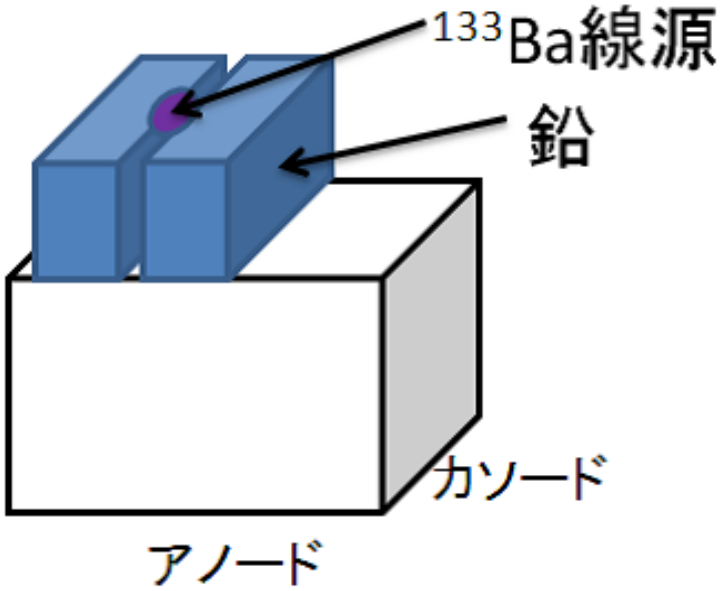
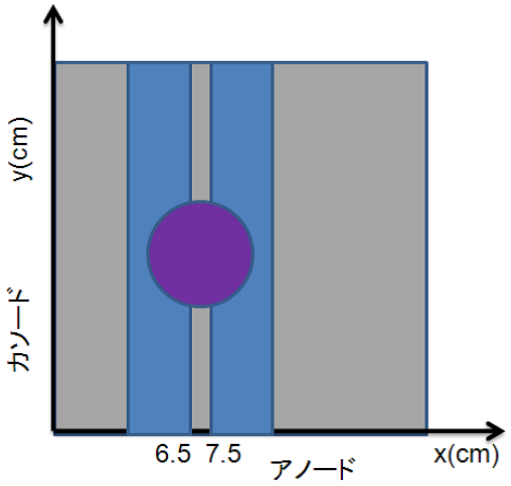
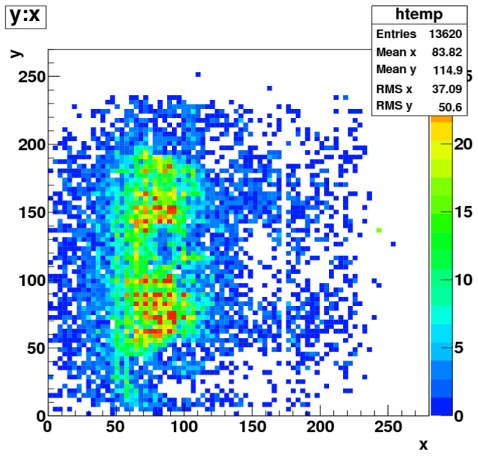


図 13

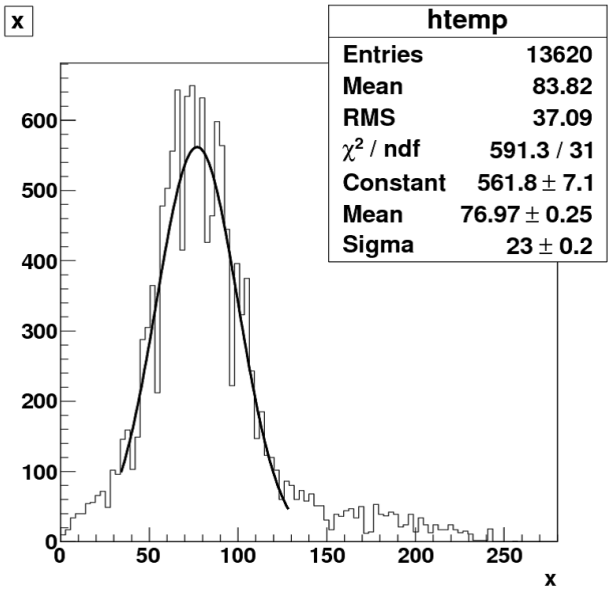
このときの 2 次元ヒストグラムは図 14(b) のようになる。また、このときのアノード方向のヒストグラムは図 14(c) になった。このことから、6.5-7.5cm に対応するのは 77 ± 23channnel であることがわかる。



(a) 鉛で遮蔽



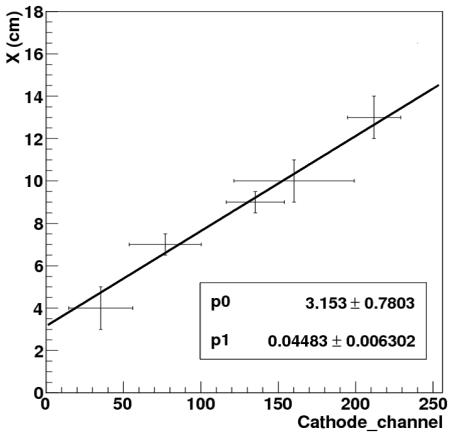
(b) 2 次元ヒストグラム



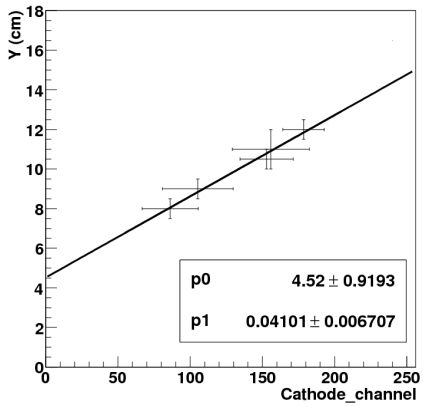
(c) アノード方向のヒストグラム

図 14　測定例（アノード方向に 6.5-7.5cm で鉛を置いたとき）

この測定をアノード (X)・カソード (Y) 方向それぞれ何箇所か測定し、channel と位置の対応付けを見た (図 15)。



(a) アノード (X 方向)



(b) カソード (Y 方向)

図 15

このことから、実際のガス容器内では図 16 のように少し左上の位置にあったことが分かった。

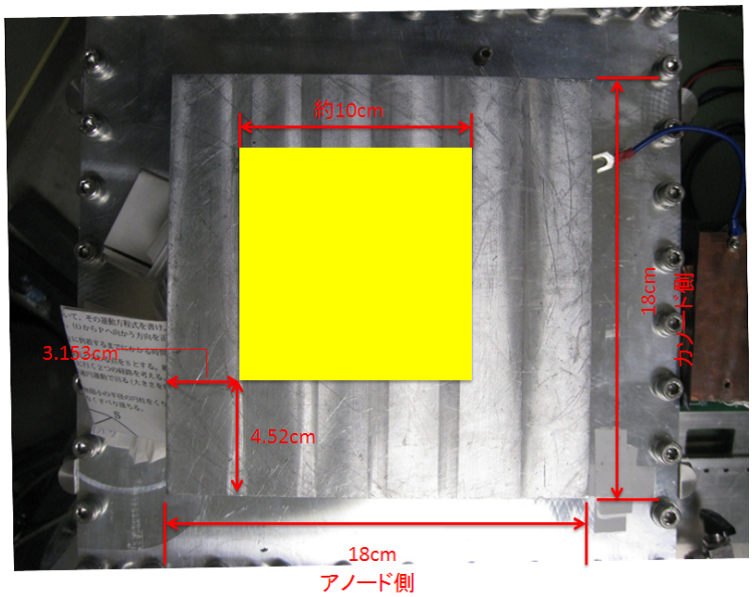


図 16　ガス容器内でのμ-PIC の位置

3.2 n-p 散乱の軌跡と散乱角

3.2.1 実験原理

μ-PIC では、非荷電粒子の中性子の検出は、主に、中性子によって散乱される陽子の奇跡として測定できる。

中性子-陽子の散乱反応は、中性子の非荷電性より、純粋な 2 体弾性衝突の問題に帰着でき、散乱角について、図 17 より、式 2 が成り立つ。

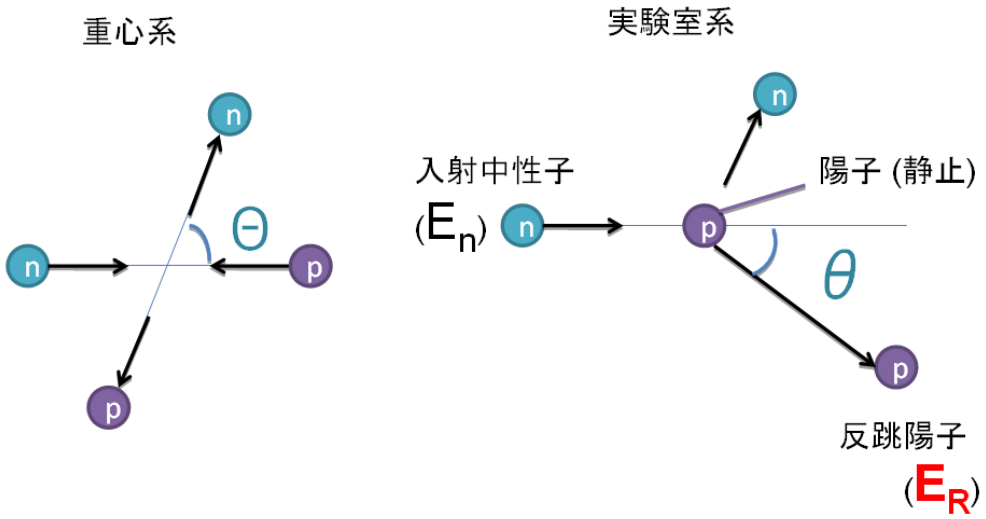


図 17　散乱角

$$E_R = \frac{1-\cos\Theta}{2} E_n = \cos^2\theta \times E_n \tag{2}$$

3.2.2 解析

測定では、^{252}Cf を中性子源として用い、ガス容器に対して X 方向に-5cm、Y 方向に 9cm、Z 方向 (検出面からの高さ) に 4cm の位置に置いた。

X-ray mode では、時間幅を 2msec と電子雲形成に対して十分長いので、測定データ中の 1 イベントでは clock 数が等しくなる。を考慮し、測定データから clock 数が 3 個以上連続して同じもののみを抽出した。

抽出後のデータの例として 15 個分が図 18 である。

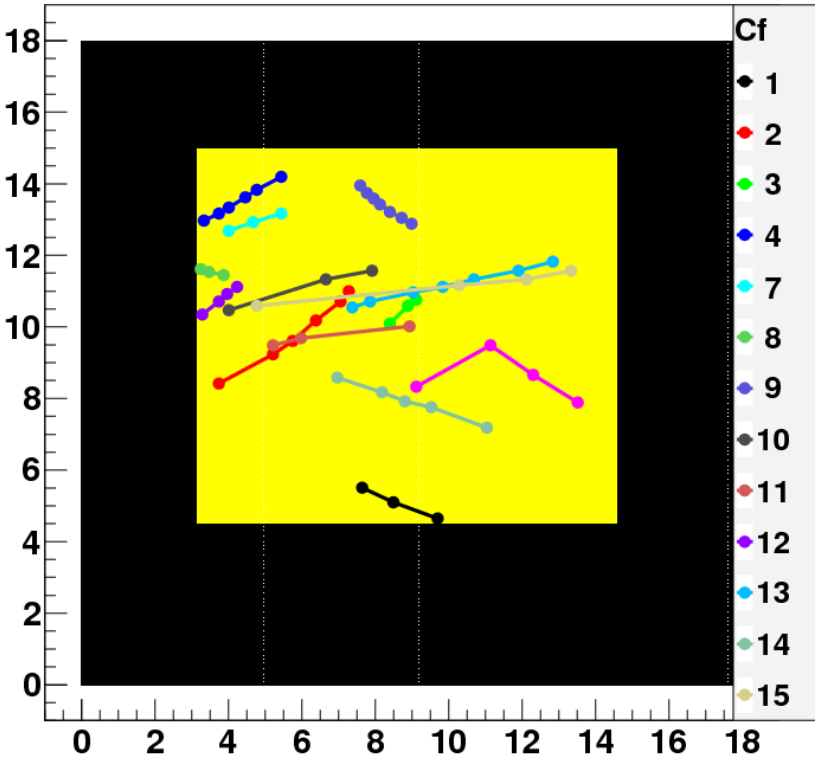


図 18　測定データ (15 個分)

測定データにノイズなどの陽子の軌跡以外のデータが見られたので、それらを排除するため、データを 1 イベントごとに 1 次関数でフィッティングしてやり、得た直線からの距離の平均 (図 19) が 0.1 cm 以上のものを排除した。

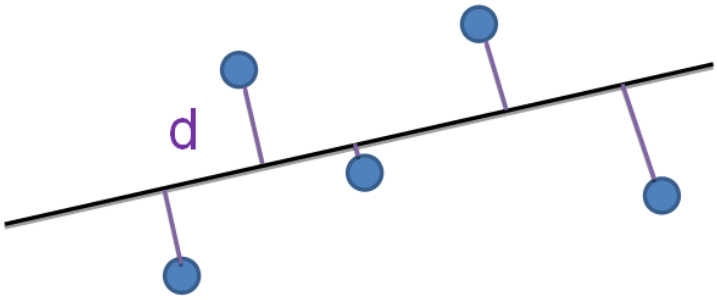


図 19　フィッティング直線と距離 d

先ほどの図 18 の中から排除されたデータが図 20 である。

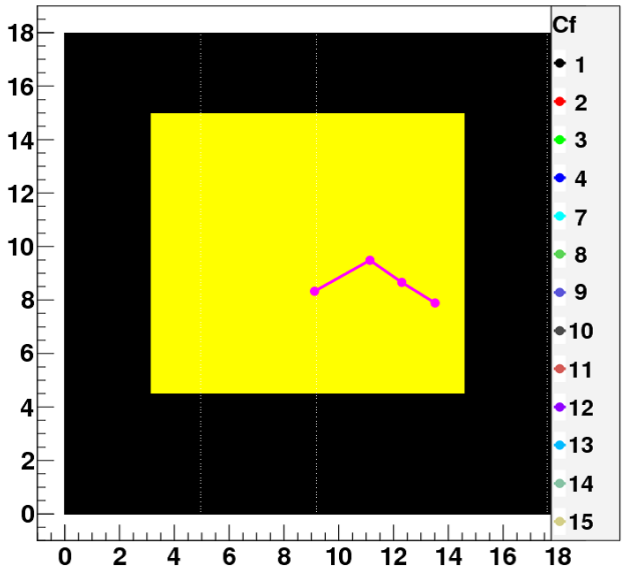


図 20　排除されたデータ

この抽出後のデータから、散乱角を求める。
中性子の最初の座標はセットアップから (-5cm、9cm) となり、散乱地点は測定データの中から一番 X の値が小さいものを選び、この 2 点から、入射中性子の角度を得て、それに対するフィッティング関数の傾きから、最終的な散乱角を求める (図 21)

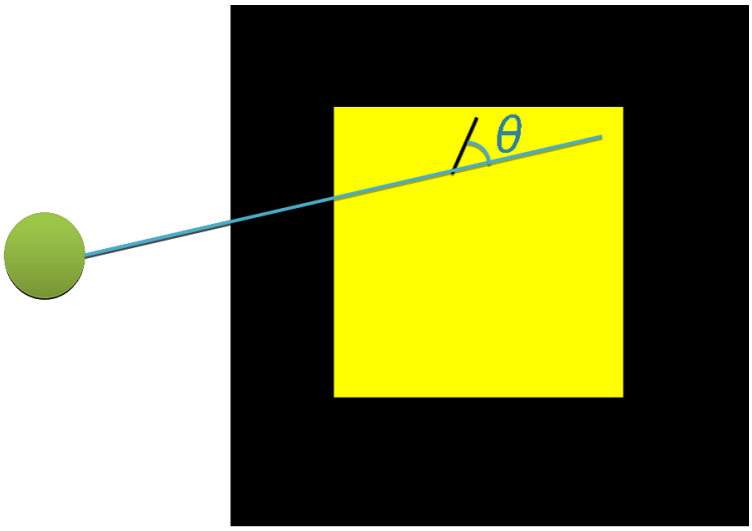


図 21　入射角に対する散乱角

このようにして得られた散乱角の $\cos\theta$ をヒストグラムにし、図 22 を得た。
散乱方向は、かなり前方に偏っているのが見られた。

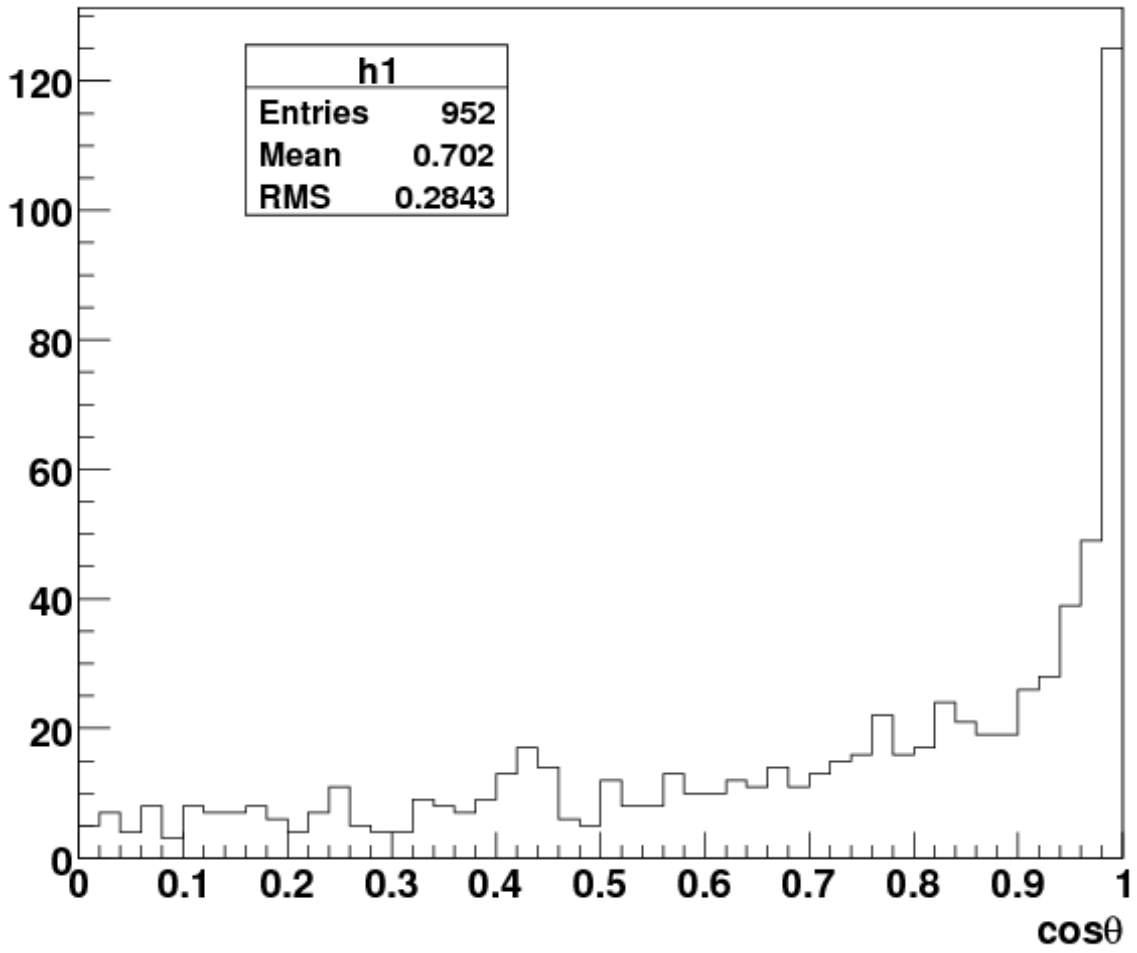


図 22　測定から得られた散乱角

3.2.3 シミュレーション

メルセンヌ・ツイストを用いて乱数を得て、中性子-陽子散乱のシミュレーションを行った。中性子-陽子散乱では、同一エネルギーの中性子 (E_n) から散乱がおこる場合、陽子のエネルギー分布は等確率になる。よって、式 2 の関係から $\cos\theta$ のヒストグラムは 1 次関数になり、シミュレーションでは図 23 を得た。

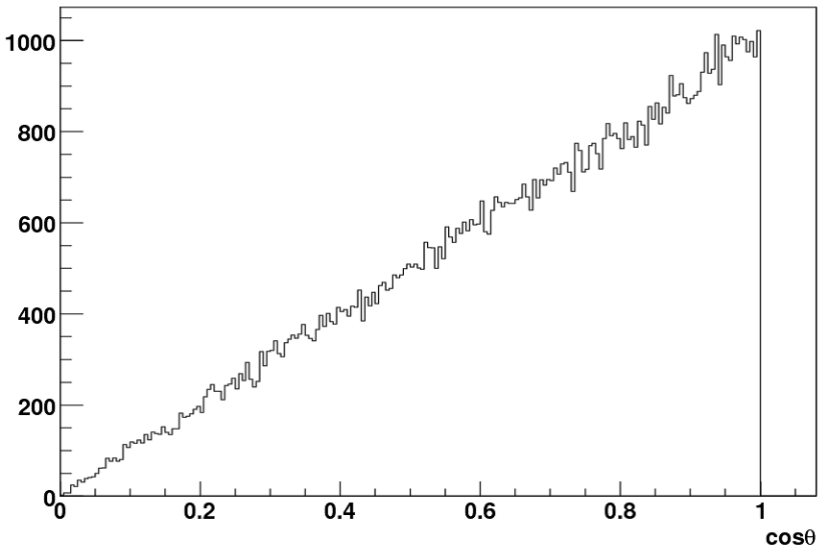


図 23　シミュレーションによる散乱角

ここから、陽子のエネルギーが小さいと検出されないことや、入射中性子にもエネルギー分布 (式 3) があることを考慮する。

$$\frac{dN}{dE_n} = E_n^{\frac{1}{2}} e^{-\frac{E_n}{T}} \quad (T = 1.42MeV) \tag{3}$$

つまり、式 3 に従いシミュレーションを行い、得られた散乱角分布を更に、E_R の大きさでカットしてやる。

実際、シミュレーションすると図 24 の結果を得た。E_R でカットしてやることで前方のピークが大きくなるのがわかる。

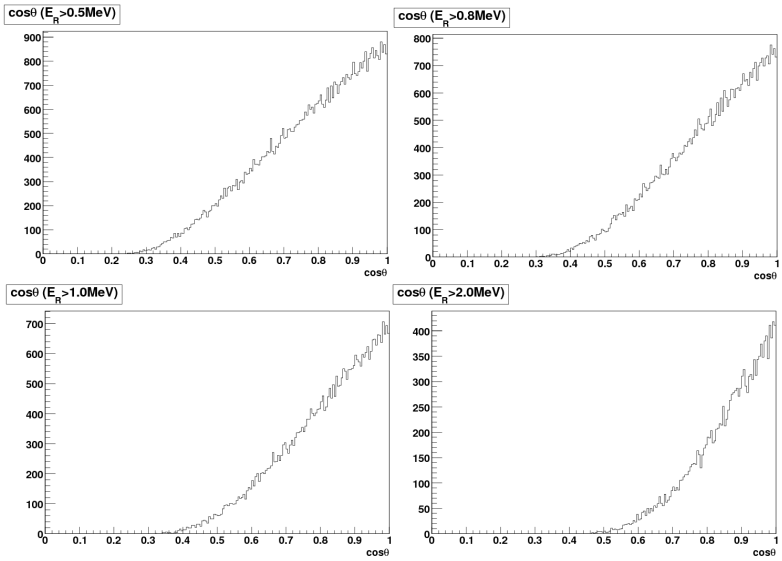


図 24　散乱角の分布　E_R でカット

測定データとシミュレーションで件数をそろえてヒストグラムで比較すると、図 25 になる。

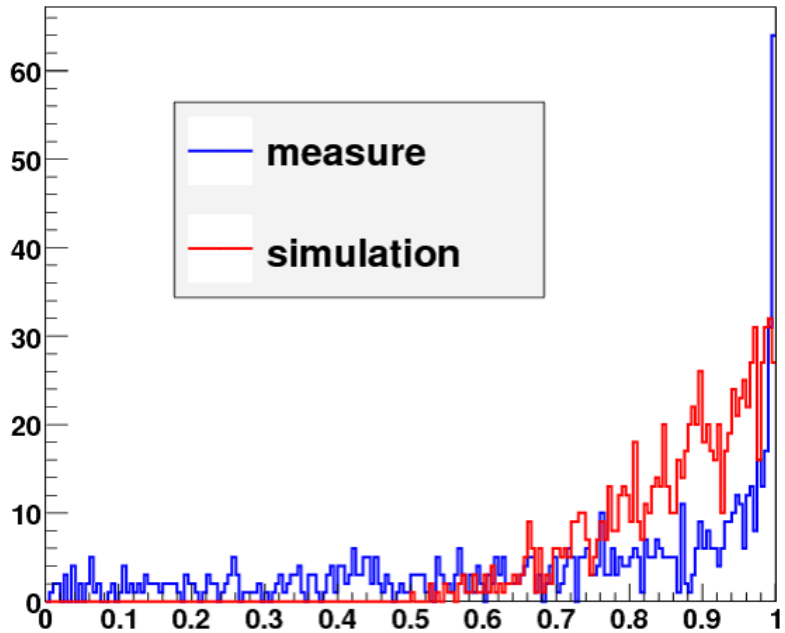


図 25　散乱角の分布　E_R でカット

正確には、シミュレーションの散乱角は入射方向に対する 3 次元の仰角であり、その角度を 2 次元平面に射影しなければならずピークは更に鋭くなるはずである。

4　三次元トラッキング

Xray mode では 2 次元データの取得が行えた。新たに「TPC mode」を用いて、3 次元測定を行った。

4.1　TPC mode

TPC mode では、Xray mode に加え FADC の信号も同期させることで、1 イベントごとのデータ取得が可能の上、時間幅が 10nsec なので 1 イベント内のデータ点の時間差も見ることができる。検出時間がわかれば、印加電圧から電子雲の Drift 速度がわかるので、粒子の軌跡の相対的な高さ情報も得られる。また、FADC のデータも得られるため、同時に粒子のエネルギーも測定可能である。

4.2　結果

今回、私たちはこの TPC mode を用いて、3.2 で行った中性子-陽子散乱の散乱角の解析を行う予定であったが、なかなか件数がたまらず残念ながら十分なデータを得ることができなかった。

ちなみに、参考として、得られたデータの 1 例は図 26 のようになった。

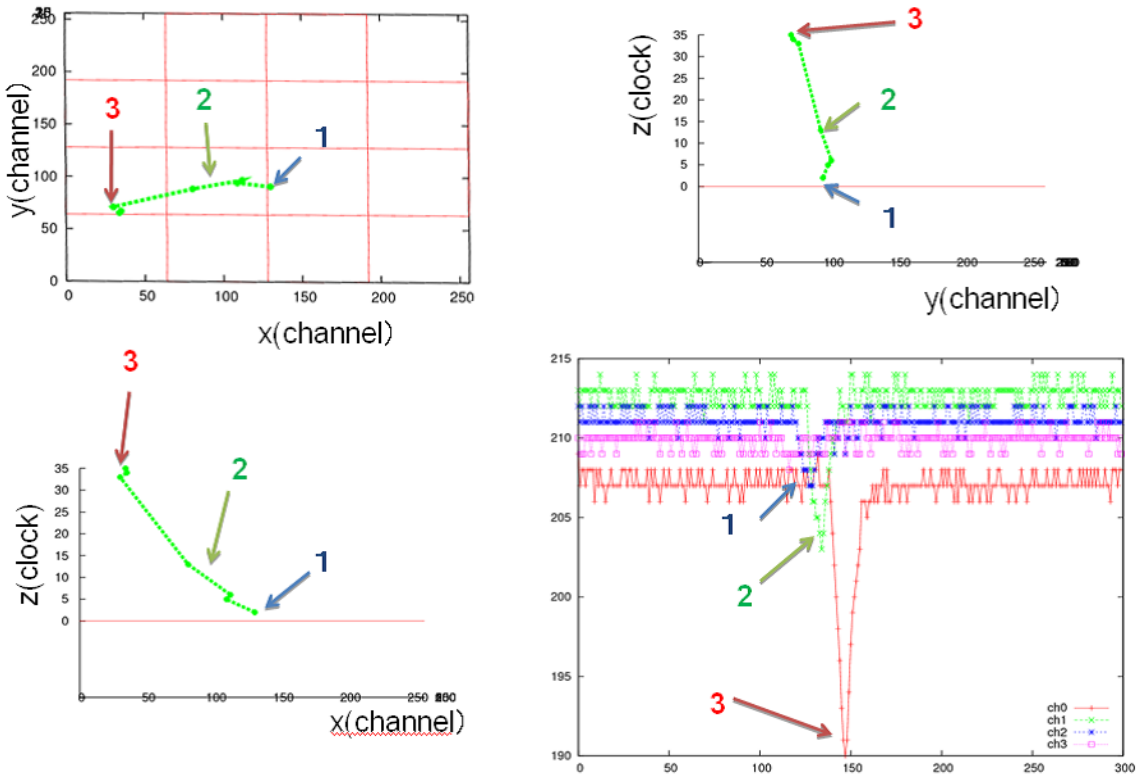


図 26　3 次元測定データ

FADC の測定時間と TPC mode による高さに相当する時間を比べてみる。
FADC の ch0 と ch1 のピークの時間差は約 224nsec、ch1 と ch2 の差が約 80nsec であり、高さはそれぞれ 220nsec、80nsec となっており、正しく対応しているのがわかる (図 27)。

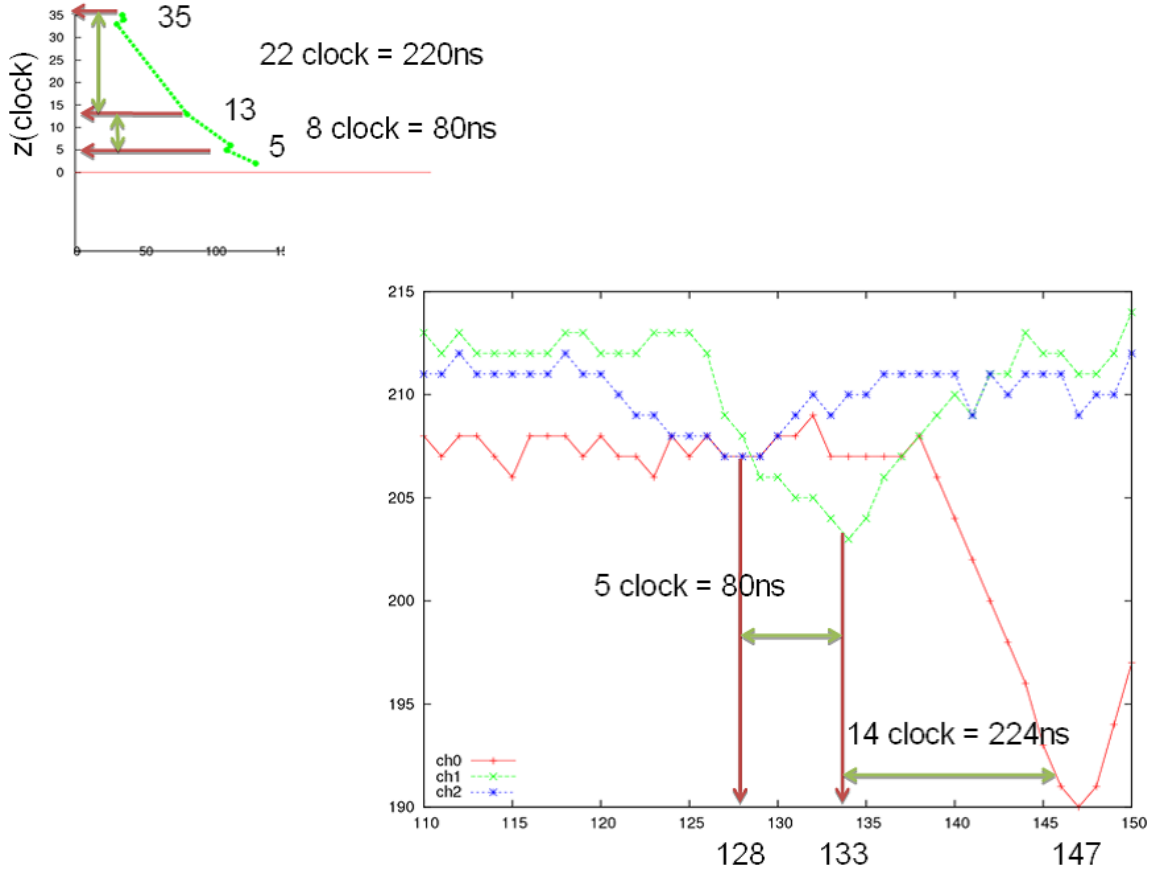


図 27　測定時間の比較

5　まとめ

最終的に、2 次元イメージングでは、中性子-陽子散乱の散乱角の取得に成功することができ、シミュレーションとの比較も行えた。これを 3 次元イメージングに応用すればより良い結果が得られるはずである。
また、今回、私たちの実験では、アノードへの印加電圧を 420V 前後と、比較的小さい値で行った。初期のころの放電を考慮してその程度の値で行っていたのだが、後半では、放電に対してかなり安定していたので、もう少し大きな値でも測定可能であったようである。アノード電圧を上げることができれば、増幅率があがり、検出効率・エネルギー分解能なども良くなることが期待される。

6　参考文献

G.F.Knoll 木村逸郎/阪井英次訳『放射線計測ハンドブック第 3 版』
日刊工業新聞社 2001 年
村上悠紀雄/団野皓文/小林昌敏 『放射線データブック』
地人書館 1982 年
加藤貞幸『放射線計測』
培風館 2003 年

7 付録 プログラムソース

7.1 波形から電荷量を求める

2.1 で用いた、波形解析プログラム。
反応した ch の数も数えている。

```
/* 測定生データから電荷量を求める
   ソートの条件は、maen から WIDTH 以下のデータ
   + 隣りのデータも（連続であるか否か）        */
/* 出力は電荷単位（クーロン値）*/
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fname.cxx"

#define TAKE_CLOCK 300
#define TAKE_CH 8
#define PEAK_LENGTH 200

#define WIDTH 1.9 //条件幅

using namespace std;

double chenge(double charge, int ch);

int main(int argc, char *argv[]){

  if(argc =! 2){
    cout << "入力ファイルを記入してください\n";
    return 1;
  }

  char ifname[32];
  sprintf(ifname, argv[1]);
  ifstream fin(ifname);
```

```
  char ofname[32];
  int data_num;
  fname_charge(ofname, ifname, &data_num);
  ofstream fout(ofname);
  cout << ofname << endl;

  int num, t, ch;
  double charge[TAKE_CH];
  double mean[TAKE_CH];
  int data[TAKE_CH][TAKE_CLOCK];
  int clock;
  int hit=0;
  int hit_1=0;

  for(num=0;num<data_num;num++){
    //初期化
    for(ch=0;ch<TAKE_CH;ch++){
      charge[ch] = 0;
      mean[ch] = 0;
    }
    //まずは mean を求める
    for(t=0;t<TAKE_CLOCK-PEAK_LENGTH;t++){
      fin >> clock;
      for(ch=0;ch<TAKE_CH;ch++){
fin >> data[ch][t];
mean[ch] += (double)data[ch][t];
      }
    }
    for(ch=0;ch<TAKE_CH;ch++){
      mean[ch] = mean[ch]/(TAKE_CLOCK-PEAK_LENGTH);//平均
    }

    //次に charge を求める
    //とりあえず配列に収容
    for(t=TAKE_CLOCK-PEAK_LENGTH;t<TAKE_CLOCK;t++){
      fin >> clock;
      for(ch=0;ch<TAKE_CH;ch++){
fin >> data[ch][t];
      }
    }
```

```
    //次に条件をチェックし求める
    for(t=TAKE_CLOCK-PEAK_LENGTH;t<TAKE_CLOCK-2;t++){
      for(ch=0;ch<TAKE_CH;ch++){
//反応したとみなす条件
if(data[ch][t] < mean[ch]- WIDTH){
  if((data[ch][t-1] < mean[ch]- WIDTH&&data[ch][t+1] < mean[ch]-
 WIDTH)||(data[ch][t-1] < mean[ch]- WIDTH&&data[ch][t-2] < mean[ch]- WIDTH)
||(data[ch][t+2] < mean[ch]- WIDTH&&data[ch][t+1] < mean[ch]- WIDTH)){
    charge[ch] = charge[ch] + mean[ch] - data[ch][t];
  }
}
      }
    }

    //hit 数を数える
    for(ch=0;ch<TAKE_CH;ch++){
     if(charge[ch]!=0){hit++;}
    }

    if(hit==1){

      fout << num << " ";
      for(ch=0;ch<TAKE_CH;ch++){
fout << chenge(charge[ch],ch) << " ";
      }t
      hit_1++;
      fout << endl;
    }else{
      fout << num << " ";
      for(ch=0;ch<TAKE_CH;ch++){
fout << chenge(charge[ch],ch) << " ";
      }
      fout << endl;
    }
    hit=0;
  }

  cout << hit_1 << endl;

  fin.close();
```

```
  fout.close();
  return 0;
}

double chenge(double charge, int ch){
  double a[8]; //1ch 当たりの電圧 (mV)
  a[0]=3.8754;
  a[1]=3.8844;
  a[2]=3.9571;
  a[3]=4.1290;
  a[4]=3.9595;
  a[5]=3.9155;
  a[6]=4.0371;
  a[7]=4.0114;
  double t = 16.0; //1ch 当たりの時間 (nsec)
  double regis = 50; //オシロの抵抗値 (Ω)

  charge = charge * a[ch] * t / regis;

  return charge; // 検出された電荷量 (*10e-12 C)
}
```

7.2 軌跡のプロット数

3.2.2 で用いた、同一イベントのデータ数を数えるプログラム。

```
/* 2D Cf   のソート用
   clock 数の連続数を数えてからソート
   出力は cm                                */
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>

#define CUT 3

using namespace std;

double channel_to_cm_x(int x);
double channel_to_cm_y(int y);
```

```
/* 入力ファイル名は"Cf_num.dat"の形で NUM 数を引数として受ける */
int main(int argc,char *argv[]){

  if(argc =! 2){
    cout << "ファイル数を入力してください\n";
    return 1;
  }

  int num;
  num = atoi(argv[1]);

  int i,j,k;
  int l = 0;
  int sum = 0;
  int number = 0;
  char fname[128];
  string ofname = "Cf_sort.dat";
  FILE *fp;
  fp =fopen("Cf_sort.dat","w");
  ifstream fin;

  int x[800];
  int y[800];
  int clock[800];
  double x_cm, y_cm;

  int n,m;
  int hit[800];

  for(i=0;i<num;i++){
    sprintf( fname, "Cf_%d.dat", i );
    cout << "OPEN:" << fname << endl;
    fin.open(fname);

    /*とりあえず収容＋有効件数を数える**************/
    k = 799;
    for(j=0;j<800;j++){
      fin >> x[j] >> y[j] >> clock[j];
      if(x[j]==0&&y[j]==0&&clock[j]==0){
```

```
k = j-1;
break;
      }
    }
    cout << "有効件数:" << k+1 << endl;
    /*****************************************/


    /*1 イベントの hit 数（連続数）を数える****************/
    j = n = 0;
    for(j=0;j<k; ){
      hit[n] = 1;
      while(clock[j]==clock[j+hit[n]]){
hit[n]++;
      }
      //とりあえず CUT 数で判断
      if(hit[n]>CUT-1){
number++;
fprintf(fp,"%2d  %2d\n",hit[n],number);
for(m=0;m<hit[n];m++){
  x_cm = channel_to_cm_x(x[j+m]);
  y_cm = channel_to_cm_y(y[j+m]);
  fprintf(fp,"%7.5e %7.5e %5d\n",x_cm,y_cm,clock[j+m]);
}
l++;
      }

      j = j + hit[n];
      n++;
    }
    /*********************************************/

    sum += l;
    printf("Cf_%d からの抽出イベント数:%d 合計:%d\n",i,l,sum);
    l = 0;
    fin.close();
  }

  return 0;
}
```

```
double channel_to_cm_x(int x){
  return (double)(x/2 * 0.04483 + 3.153);
}
double channel_to_cm_y(int y){
  return (double)(y/2 * 0.04101 + 4.520);
}
```

7.3 軌跡とフィッティングとの比較＋散乱角の取得

3.2.2 でのフィッティングからの距離の大きさのカットのプログラム。ROOT で行い、散乱角まで求めた。

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstdlib>
#include <stdlib.h>
#define NUM_MAX 500
#define SIGMA_MAX 0.1 //Fitting からの距離の平均

using namespace std;

void scat(const char *fname){
  gROOT->SetStyle("Plain");
  gStyle->SetOptStat(0);
  gStyle->SetFrameFillColor(0);
  int i;

  // 中性子源の場所
  double x0 = -5.0 ;
  double y0 =  9.0 ;

  TCanvas **c = new TCanvas*[4];
  char c_name[16];
  for(i=0;i<4;i++){
    sprintf(c_name,"c%d",i);
    c[i] = new TCanvas(c_name, c_name, 900, 900);
```

```
    TH1F *frame = gPad->DrawFrame(-1,-1,19,19);
  }

  TLegend *leg = new TLegend(0.85,0.1,0.95,0.9,"Cf");

  /*COLOR TABLE 作成**********************************/
  int color[20] = {1,2,3,4,5,6,7,8,9,12,50,51,65,30,41,
91,98,28,38,20};
  /***************************************************/


  //--Read the data file.
  ifstream fin(fname);

  if(!fin){
    cerr << "Cannot open file." << endl;
    exit(1);
  }

  //解析用あれこれ*****************************
  double x_low, x_high, sigma;
  double pol0, pol1;
  double theta2;  //Fitting の傾きの角度
  double phi;     //入射角
  double cos_theta;   //散乱角
  int k;
  ofstream fout("Cf_scat.dat");
  TF1 **f = new TF1*[NUM_MAX];
  char f_name[16];
  //*****************************************


  int clock;
  double x[NUM_MAX][20];
  double y[NUM_MAX][20];
  int hit, gomi;
  TGraph **gra = new TGraph*[NUM_MAX];
  char name[5];

  /* μ PIC の外面 ***********************/
```

```
double x_min =  0 ;
double x_max = 18 ;
double y_min =  0 ;
double y_max = 18 ;
b = new TBox(x_min, y_min, x_max, y_max);
b->SetFillColor(1);
b->SetFillStyle(3001);
for(i=0;i<4;i++){
  c[i]->cd(0);
  b->Draw("same");
}
/**********************************/

/* μPIC内の検出面 ********************/
x_min =  3.153;
x_max =  3.153 + 0.04483*255;
y_min =  4.520;
y_max =  4.520 + 0.04101*255;
b2 = new TBox(x_min, y_min, x_max, y_max);
b2->SetFillColor(5);
b2->SetFillStyle(3001);
for(i=0;i<4;i++){
  c[i]->cd(0);
  b2->Draw("same");
}
/**********************************/

int num = 0;
int n = 0;
int j = 0;
while(fin >> hit >> gomi){
  //if(gomi==16){break;}
  for(i=0;i<hit;i++){
    fin >> x[num][i] >> y[num][i] >> clock;
  }
  gra[num] = new TGraph(hit,x[num],y[num]);
  gra[num]->SetLineColor(color[j]);
  gra[num]->SetMarkerColor(color[j]);
  gra[num]->SetLineWidth(3);
  gra[num]->SetFillColor(0);
```

```
    gra[num]->SetMarkerStyle(8);
    gra[num]->SetMarkerSize(1.3);
    c[0]->cd(0);
    sprintf(f_name,"f%d",num);
    f[num] = new TF1(f_name,"[0]+[1]*x",-5,30);
    gra[num]->Draw("PL same");
    gra[num]->Fit(f[num],"N0","",-1,26);
    f[num]->SetLineColor(color[j]);
    f[num]->SetLineWidth(2);
    //f[num]->Draw("same");

    // SIGMA を求める  ************************
    pol1 = f[num]->GetParameter(1);
    pol0 = f[num]->GetParameter(0);
    for(k=0;k<i;k++){
      sigma = ( y[num][k] - pol1*x[num][k] - pol0 )*(
 y[num][k] - pol1*x[num][k] - pol0 );
    }
    sigma = ( sqrt( sigma )/( 1 + pol1*pol1 ) )/ i;
    //****************************************
    if( sigma < SIGMA_MAX ){
      c[1]->cd(0);
      gra[num]->Draw("PL same");
      c[2]->cd(0);
      gra[num]->Draw("PL same");
      f[num]->Draw("same");
      sprintf(name,"%2d",num+1);
      // 入射角に対する散乱角を得る******************
      theta2 = atan( pol1 );
      phi = atan( ( pol1*x_low+pol0 - y0 )/( x_low - x0 ) );
      cos_theta = cos( phi )*cos( theta2 ) +
sin( phi )*sin( theta2 );
      fout << cos_theta <<"  "<< sigma << endl;
      //****************************************
      n++;
      leg->AddEntry(gra[num],name);
    }else{
      c[3]->cd(0);
     gra[num]->Draw("PL same");
    }
```

```
    num++;
    j++;
    if(j==20){j=0;}
  }

  for(i=0;i<4;i++){
    c[i]->cd(0);
    leg->Draw("same");
  }
  cout << endl << n << endl;
  fin.close();
  fout.close();
}
```

7.4 シミュレーション

3.2.3 で用いた、シミュレーションプログラム。

```
/* 三次元 中性子-陽子 弾性散乱 シミュレーション
   z 軸に入射 ( r0_1=(0,0,1) ) した中性子が
   (0,0,0) で散乱し、反跳陽子がどの方向に飛ぶか
   + その方向を y-z 平面 (実験における xy 平面) に射影し、角度を求める

   入射中性子のスペクトルも考える                                    */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "mt19937ar.c"
#include "vector_sim.cxx"

#define NUM 6135

using namespace std;

double get_En();
```

```
int main(int argc,char *argv[]){

  FILE *fp;
  FILE *fp2;
  fp2 = fopen("pro900.dat","w");
  int randum = 100;
  if(argc == 2){
    randum = atoi(argv[1]);
  }else if(argc > 2){
    cout << "入力 ERROR" << endl;
    return 1;
  }
  init_genrand(randum);

  vector r0, r1, r2;
  vector r0_1, r1_2;
  double theta;//射影された角度
  double cos_theta;
  double y,z;
  double cos_THETA;//重心系の散乱角の cos
  double energy;   //陽子のエネルギー比
  double scat;     //陽子の本当の散乱角の cos
  double phi;      //陽子の散乱の方位角
  double En_energy;//入射中性子のエネルギー

  int i = 0;
  int j = 1;
  while(i<NUM){
    i++;

    r1.inter( 0, 0, 0);



    r1_2 = get_scat2( 0, 0, &cos_THETA, &energy, &scat, &phi);

    r2 = r1 + r1_2;

    y = r2.get_y();
    z = r2.get_z();
```

```
    theta = atan( y / z );
    cos_theta = cos( theta );

    En_energy = get_En();

    energy = En_energy * energy;

    fprintf(fp2,"%5d %12.11f %12.11f %12.11f %12.11f %12.11f
%12.11f %12.11f\n", i, theta, cos_theta, cos_THETA , energy, scat,
 phi, En_energy);

    if( (i/(j*100)) == 1){
      printf("%5d\n",i);
      j++;
    }
  }
}


//10 万分割で
double get_En(){
  double random = genrand_real1()*1.4954;//[ 0, 1.4954]
1.4954 は約 10MeV までの確率より下 (10MeV 以上はでない)

  double sum = 0;
  double delta = (double)10.0/NUM;
  double energy = 0;
  int i;
  for( i=0; i<100000; i++){
    energy += delta;
    sum += sqrt( energy )*exp( - energy / 1.42) *delta;
    if( sum > random ){
      return energy;
    }
  }
  printf("突破!!!!\n");
  exit(1);
  return energy;
}
```

7.5 ベクトル計算クラス

上のプログラムで用いた、ベクトル操作用クラス。

```
/*3 次元シミュレーション用ベクトルクラス + 専用関数

   入り口を鉛で遮蔽し y=± 1 からしか入射できない          */
#include <iostream>
#include <math.h>
#include <stdlib.h>

using namespace std;
//         μ PIC 外枠       μ PIC
//          ----   9     ---- y_max
//          |  |          |  |
//          ----  -9     ---- y_min
//          7  25     x_min x_max
double x_min =  7 + 3.153;               // 10.153
double x_max =  7 + 3.153 + 0.04483*255; // 21.58465
double y_min = -9 + 4.520;               // -4.48
double y_max = -9 + 4.520 + 0.04101*255; //  5.97755
double z_min = -5;
double z_max =  5;

class vector;
ostream &operator<<(ostream &ost, vector obj);
vector operator*(double sloop, vector ob);
vector unit(vector ob);
vector reach_x(vector origin, vector unit, double x);
vector reach_y(vector origin, vector unit, double y);
vector reach_z(vector origin, vector unit, double z);
int check1(vector r0);
int check2(vector r1);
vector get_scat(double theta_A, double phi_A);
vector get_terminal(vector r1, vector r1_2);
int get_inter( vector &r1, vector r1_2 );
double get_length(vector r1, vector r2);
vector get_scat2(double theta_A, double phi_A,
double *cos_THETA, double *ene,
```

```
double *scat, double *phi);

class vector {
  double x, y, z;
  double length;
public:
  vector(double x0 = 0, double y0 = 0, double z0 = 0);
  void inter(double x0, double y0, double z0);
  double get_x(){return x;}
  double get_y(){return y;}
  double get_z(){return z;}
  double get_length(){return length;}
  vector operator-(vector ob);
  vector operator+(vector ob);
  void disp();
  friend ostream &operator<<(ostream &ost, vector obj);
  friend vector operator*(double sloop, vector ob);
  friend vector unit(vector ob);
  friend vector reach_x(vector origin, vector unit, double x);
  friend vector reach_y(vector origin, vector unit, double y);
  friend vector reach_z(vector origin, vector unit, double z);
  friend int check1(vector r0);
  friend int check2(vector r1);
  friend vector get_scat(double theta_A, double phi_A);
  friend vector get_terminal(vector r1, vector r1_2);
  friend int get_inter( vector &r1, vector r1_2 );
  friend double get_length(vector r1, vector r2);
  friend vector get_scat2(double theta_A, double phi_A,
double *cos_THETA, double
*ene, double *scat, double *phi);
};

vector::vector(double x0, double y0, double z0){
  x = x0;
  y = y0;
  z = z0;
  length = sqrt( x*x + y*y + z*z );
}

void vector::inter(double x0, double y0, double z0){
```

```
  x = x0;
  y = y0;
  z = z0;
  length = sqrt( x*x + y*y + z*z );
}


vector vector::operator-(vector ob){
  return vector( x - ob.x, y - ob.y, z - ob.z);
}

vector vector::operator+(vector ob){
  return vector( x + ob.x, y + ob.y, z + ob.z);
}


void vector::disp(){
  cout << " x:" << x << " y:" << y << " z:" << z << endl;
  cout << "LENGTH:" << length << endl;
}

ostream &operator<<(ostream &ost, vector ob){
  ost <<"( "<<ob.x<<", "<<ob.y<<", "<<ob.z<<" ) : Length "<<ob.length;
  return ost;
}
vector operator*(double sloop, vector ob){
  return vector( sloop * ob.x, sloop * ob.y, sloop * ob.z);
}
vector unit(vector ob){
  return vector(ob.x/ob.length, ob.y/ob.length, ob.z/ob.length);
}
// 初期座標 (origin) から unit 方向に進んで与えられた yz 面に到達する座標を求める
vector reach_x(vector origin, vector unit, double x){
  double sloop = ( x - origin.x ) / unit.x;

  if(sloop<0){
    return vector(10000,10000,10000);
  }

  return origin + sloop * unit;
```

```
}

vector reach_y(vector origin, vector unit, double y){
  double sloop = ( y - origin.y ) / unit.y;

  if(sloop<0){
    return vector(0,0,0);
  }

  return origin + sloop * unit;
}



vector reach_z(vector origin, vector unit, double z){
  double sloop = ( z - origin.z ) / unit.z;

  if(sloop<0){
    return vector(0,0,0);
  }

  return origin + sloop * unit;
}

/* μPIC に入射するかどうかを確認
   OK 0
   NG 1                          */
int check1(vector r0){
  if(r0.y>1 || r0.y<-1 || r0.z>z_max || r0.z<z_min){
    return 1;
  }
  return 0;
}



/* 衝突座標を受け取りどこにいるか確認
   検出面内で  0
   外枠内で    1
   外で        2                  */
int check2(vector r1){
                              //        μPIC 外枠       μPIC
```

```
                              //          ----   9     ---- y_max
  if(r1.y >  9){return 2;}    //          |  |          |  |
  if(r1.y < -9){return 2;}    //          ----  -9     ---- y_min
  if(r1.x > 22){return 2;}    //          4  22     x_min x_max
  if(r1.z > z_max){return 2;}
  if(r1.z < z_min){return 2;}
  if(r1.y > y_max){return 1;}
  if(r1.y < y_min){return 1;}
  if(r1.x > x_max){return 1;}
  if(r1.x < x_min){return 1;}

  return 0;
}
```

/* 散乱方向ベクトルを乱数から得る
　散乱角は，反跳陽子のエネルギーが等確率で与えられるものとして */

```
vector get_scat(double theta_A, double phi_A){

  //入射方向（r，θ，Φ）で　r 方向を z 軸、θ方向を x 軸、-Φ方向を y 軸、
で散乱角を求める

  //反跳陽子のエネルギー（全エネルギー比）[0,1]
  double energy = genrand_real1();
  //phi(散乱角）の角度 エネルギーから [0，π/2]
  double phi = acos( sqrt(energy) );// [0，π/2]

  //theta 方向の角度 [0,2 π]
  double theta = genrand_real1()*2*M_PI;

  double x, y, z;

  x = -cos(theta)*sin(phi)*sin(theta_A) - sin(theta)*sin(phi)
*cos(theta_A)*cos(phi_A) + cos(phi)*cos(theta_A)*sin(phi_A);
  y = cos(theta)*sin(phi)*cos(theta_A) - sin(theta)*sin(phi)
*sin(theta_A)*cos(phi_A) + cos(phi)*sin(theta_A)*sin(phi_A);
  z = sin(theta)*sin(phi)*sin(phi_A) + cos(phi)*cos(phi_A);
  return vector( x, y , z );
}
```

```
/* 衝突点 or 入射点 r1 から r1_2 で進んでケースから出る点 r2 を求める */
vector get_terminal(vector r1, vector r1_2){
  vector r2;
  double sloop;

  //とりあえず x 方向の境界で sloop を求める
  if( r1_2.x > 0 ){
    sloop = ( x_max - r1.x ) / r1_2.x;
  }else{
    sloop = ( x_min - r1.x ) / r1_2.x;
  }
  if( sloop < 0 ){
    cout << "SLOOP:ERROR \n";
    exit(1);
  }
  r2 = r1 + sloop*r1_2;

  //その点が y 方向で収まっているかチェック 収まっていない場合は
 sloop を取り直す
  if( r2.y > y_max ){
    sloop = ( y_max - r1.y ) / r1_2.y;
  }else if( r2.y < y_min ){
    sloop = ( y_min - r1.y ) / r1_2.y;
  }
  if( sloop < 0 ){
    cout << "SLOOP:ERROR \n";
    exit(1);
  }
  r2 = r1 + sloop*r1_2;

  //同様に z 方向でも

  if( r2.z > z_max ){
    sloop = ( z_max - r1.z ) / r1_2.z;
  }else if( r2.z < z_min ){
    sloop = ( z_min - r1.z ) / r1_2.z;
  }
```

```
  if( sloop < 0 ){
    cout << "SLOOP:ERROR \n";
    exit(1);
  }
  r2 = r1 + sloop*r1_2;

  return r2;
}


/* ケース内で散乱した粒子が検出面にはいるか否か 入る場合は入射点を返す */
int get_inter( vector &r1, vector r1_2 ){
  vector r12;
  double sloop;
  if( r1.x < x_min ){
    //エリア(1)(4)(7)
    //右に進か否か　左の場合はNG
    if( r1_2.x < 0 ){ return 0; }
    r12 = reach_x(r1, r1_2, x_min);
    //右上に進か、右下に進か
    if( r1_2.y > 0 ){
      //右上に進む
      if( r12.y > y_max ){ return 0;}
      else if( r12.y > y_min ){
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;

      }else{
//下から入射するか否か
r12 = reach_y( r1, r1_2, y_min );
if(r12.x > x_max){ return 0;}
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
```

```
  return 1;
}
return 0;
      }

    }else{
      //右下に進む
      if( r12.y < y_min ){ return 0;}
      else if( r12.y < y_max ){
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;

      }else{
//上から入射するか否か
r12 = reach_y( r1, r1_2, y_max );
if(r12.x > x_max){ return 0;}
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;
      }
    }
  }else if( r1.x < x_max ){
    //エリア (2)(8)
    if( r1.y > y_max ){
      //エリア (2)
      //上に進か、下に進か 上の場合は NG
      if( r1_2.y > 0){ return 0; }
      //上から入射するか否か
      r12 = reach_y( r1, r1_2, y_max );
      if( x_min > r12.x || r12.x > x_max ){ return 0; }
      //入射の可能性 z 方向も確認
```

```
      if( z_min < r12.z && r12.z < z_max ){
//大丈夫
r1 = r12;
return 1;
      }
      return 0;
    }else{
      //エリア(8)
     //上に進か、下に進か 下の場合はNG
      if( r1_2.y < 0){ return 0; }
      //下から入射するか否か
      r12 = reach_y( r1, r1_2, y_min );
      if( x_min > r12.x || r12.x > x_max ){ return 0; }
      //入射の可能性 z方向も確認
      if( z_min < r12.z && r12.z < z_max ){
//大丈夫
r1 = r12;
return 1;
      }
      return 0;
    }
  }else{
    //エリア(3)(6)(9)
    //右に進か否か　右の場合はNG
    if( r1_2.x > 0 ){ return 0; }
    r12 = reach_x(r1, r1_2, x_max);
    //左上に進か、左下に進か
    if( r1_2.y > 0 ){
      //左上に進む
      if( r12.y > y_max ){ return 0;}
      else if( r12.y > y_min ){
//入射の可能性 z方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;
      }else{
//下から入射するか否か
```

```
r12 = reach_y( r1, r1_2, y_min );
if(r12.x < x_min){ return 0;}
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;
      }
    }else{
      //左下に進む
      if( r12.y < y_min ){ return 0;}
      else if( r12.y < y_max ){
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;
      }else{
//上から入射するか否か
r12 = reach_y( r1, r1_2, y_max );
if(r12.x < x_min){ return 0;}
//入射の可能性 z 方向も確認
if( z_min < r12.z && r12.z < z_max ){
  //大丈夫
  r1 = r12;
  return 1;
}
return 0;
      }
    }
  }
}


double get_length(vector r1, vector r2){
  return sqrt( (r1.x-r2.x)*(r1.x-r2.x) + (r1.y-r2.y)*(r1.y-r2.y)
```

```
 + (r1.z-r2.z)*(r1.z-r2.z) );
}
```

/* 散乱方向ベクトルを乱数から得る

   確認用   */

```
vector get_scat2(double theta_A, double phi_A, double *cos_THETA,
 double *ene, double *scat, double *PHI){
```

//入射方向（r, θ , Φ）で　r 方向を z 軸、θ 方向を x 軸、-Φ方向を y 軸、で散乱角を求める

```
  //反跳陽子のエネルギー（全エネルギー比）[0,1]

  double energy = genrand_real1();

  //phi(散乱角）の角度 エネルギーから [0, π/2]
  double phi = acos( sqrt(energy) );// [0, π/2]

  *cos_THETA = 1 - 2 * energy;
  *ene = energy;
  *scat = cos( phi );

  //theta 方向の角度 [0,2 π]
  double theta = genrand_real1()*2*M_PI;
  *PHI = theta;

  double x, y, z;

  x = -cos(theta)*sin(phi)*sin(theta_A) - sin(theta)*sin(phi)
*cos(theta_A)*cos(phi_A) + cos(phi)*cos(theta_A)*sin(phi_A);
  y = cos(theta)*sin(phi)*cos(theta_A) - sin(theta)*sin(phi)
*sin(theta_A)*cos(phi_A) + cos(phi)*sin(theta_A)*sin(phi_A);
  z = sin(theta)*sin(phi)*sin(phi_A) + cos(phi)*cos(phi_A);
  return vector( x, y , z );
}
```

8 謝辞

本実験にあたっては、助教の身内賢太朗先生には実験方法、解析、発表準備に至るまで様々なアドバイスをいただき、大変お世話になりました。いつもハイテンションなノリで、僕たちの不安もやわらいだものです。また、TA の澤野達哉さんにも最後まで多々助けていただきました。時には深夜に呼び出すなど非常に迷惑をかけたことと思います。その他、宇宙線研究室のスタッフのみなさんにも装置をお借りしたり、アドバイスをいただくなど助けていただきました。最後に、シンチ班の青野君、橋本君、解析班の高田君も 1 年間ゼミや実験でいろいろお世話になりました。ありがとう。

皆様方の助けがあってなんとか課題研究を終えることができました。ここに感謝の意を述べたいと思います。本当にありがとうございました。