6 付録

- 2γ の検出器にかかる確率を出したプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>

#define const1 0.021075
#define const2 0.046937

int randP();
double random2();
double naiseki(double x[], double y[]);
double absolute(double x[], double y[]);
double absolute2(double x[]);
int kettei(double kakuritsu);
double hokaku(double x[], double y[], double E);
double phot(double E);
double compt(double E);
double comptH(double nagasa, double E);
double nishina(double cosin, double E);

main()
{
  int m, i, N;
  int owari;
  int count=0, count1=0, count3=0;
  double x[3], x2[3];
  double a, b, c;
  double y[3], z[3];
  double z2[3];
  double distance=30.0, width=10.0, height=27.5, length=55.0, length2=170.0;
  double radius=29.0;
  double E=500.0;
  double hojoX, hojoY;
  double houkou[3];
  double norm;
  double cosin;
  double hokaku1=0, hokaku3=0.0;
  double touka1, touka3;
  double hokakuSUM = 0.0;
  double answer;

  scanf("%d", &N);

    srand((unsigned int)time(NULL));
    for(m=0;m<N;++m)
      {
for(i=0;i<=2;++i)
  x[i]=0.0;

while(x[0]==0.0 && x[1]==0.0 && x[2]==0.0){
  for(i=0;i<=2;++i)
    x[i]=(double)randP();

  while(sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2])>10000.0){
    for(i=0;i<=2;++i)
      x[i]=(double)randP();
  }
}

E = 500.0;

if(x[0]>0.0){
  y[0] = distance;
  y[1] = x[1] * distance / x[0];
  y[2] = x[2] * distance / x[0];
  if(sqrt(pow(y[1],2)+pow(y[2],2))<=radius
     && 0.0-height<=0.0-y[2] && 0.0-y[2]<=height
     && 0.0-width<=0.0-y[1] && 0.0-y[1]<=2.0*height-width){
    z[0] = (distance + length);
    z[1] = x[1] * z[0] / x[0];
    z[2] = x[2] * z[0] / x[0];
    if(sqrt(pow(z[1],2) + pow(z[2],2)) <=radius)
      /* 円筒との切片がNaIの底にあったとき */
```

```
   hokaku1 = hokaku(y, z, E);
  else{ /* そうでなければ側面で交わる */
   z[0] = x[0] * radius / sqrt(pow(x[1],2) + pow(x[2],2));
   z[1] = z[0] * x[1] / x[0];
   z[2] = z[0] * x[2] / x[0];
   hokaku1 = hokaku(y, z, E);
  }
  /* 他のNaIにぶつかっていたら透過率をかける */
  if(y[1] < 0.0-width)
   {
z2[1] = 0.0 - width;
z2[0] = z2[1] * x[0] / x[1];
z2[2] = z2[1] * x[2] / x[1];
touka1 = 1.0 - hokaku(y, z2, E);
hokaku1 = hokaku1 * touka1;
   }
  if(width < y[1])
   {
z2[1] = width;
z2[0] = z2[1] * x[0] / x[1];
z2[2] = z2[1] * x[2] / x[1];
touka1 = 1.0 - hokaku(y, z2, E);
hokaku1 = hokaku1 * touka1;
   }

  /* ここから、外に逃げるか光電効果が起こるまで繰り返し */
  owari = 0;
  count1 = 0;
  while(owari != 1){
   if(kettei(hokaku1) == 0) /* 捕まらなかったら終わり */
owari = 1;
   else{
if(kettei(phot(E)/(phot(E)+compt(E))) == 1)
 {
  owari = 1;
  count1 = 1; /* 光電効果したら終わり */
 }
else{ /* コンプトン散乱になった場合 */
 hojoY = 2.0;
 hojoX = 0.0;
 while(hojoY > comptH(hojoX, E)) /* 重み付き確率 */
  {
   hojoY = random2();
   hojoX = absolute(y, z) * random2();
  }
 norm = absolute(y, z);
 for(i=0;i<=2;++i) /* 反応の座標 */
  {
   houkou[i] = z[i] - y[i];
   y[i] = (z[i] - y[i]) * hojoX / norm + y[i];
  }

 hojoY = 7.0;
 cosin = 1.0;
 while(hojoY > nishina(cosin, E)) /* 重み付き確率 */
  {
   hojoY = 2.0 * random2();
   for(i=0;i<=2;++i)
x2[i] = 0;
   while((x2[0]==0 && x2[1]==0 && x2[2]==0)
  || absolute2(x2)>10000.0)
for(i=0;i<=2;++i)
   x2[i] = (double)randP(); /* γ線の方向 */
   cosin = naiseki(houkou, x2) / absolute2(x2) / norm;
  }
 /* もう一度交点を求める */
 for(i=0;i<=2;++i)
  z[i] = 0.0;
  /* x=distanceでの交点 */
 if(x2[0]<0)
  {
   z2[0] = distance - y[0];
   z2[1] = z2[0] * x2[1] / x2[0];
   z2[2] = z2[0] * x2[2] / x2[0];
   for(i=0;i<=2;++i)
z2[i] = z2[i] + y[i];
   if(sqrt(pow(z2[1],2)+pow(z2[2],2))<=radius)
for(i=0;i<=2;++i)
```

```
 z[i] = z2[i];
  }

 if(x2[0]>0)
  { /* x=distance+length での交点 */
   z2[0] = distance + length - y[0];
   z2[1] = z2[0] * x2[1] / x2[0];
   z2[2] = z2[0] * x2[2] / x2[0];
   for(i=0;i<=2;++i)
z2[i] = z2[i] + y[i];
   if(sqrt(pow(z2[1],2)+pow(z2[2],2))<=radius)
for(i=0;i<=2;++i)
 z[i] = z2[i];
  }
 if(z[0]==0 && z[1]==0 && z[2]==0)
  { /* そうでなければ円筒での交点 */
   a = pow(x2[1],2) + pow(x2[2],2);
   b = x2[1]*(x2[0]*y[1]-x2[1]*y[0])+x2[2]
*(x2[0]*y[2]-x2[2]*y[0]);
   c = pow((x2[0]*y[1]-x2[1]*y[0]),2)
+pow((x2[0]*y[2]-x2[2]*y[0]),2)-pow(radius*x2[0],2);
   if((pow(b,2)-a*c)<0)
z[0] = 0.0;
   else
{
 if(x2[0]>0)
  z[0] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
  z[0] = (-b - sqrt(pow(b,2)-a*c))/a;
}
   b = x2[2]*(x2[1]*y[2]-x2[2]*y[1]);
   c = pow((x2[1]*y[2]-x2[2]*y[1]),2)-pow(radius*x2[1],2);
   if((pow(b,2)-a*c)<0)
z[1] = 0.0;
   else
{
 if(x2[1]>0)
  z[1] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
```

```
  z[1] = (-b - sqrt(pow(b,2)-a*c))/a;
}
   b = x2[1]*(x2[2]*y[1]-x2[1]*y[2]);
   c = pow((x2[2]*y[1]-x2[1]*y[2]),2)-pow(radius*x2[2],2);
   if((pow(b,2)-a*c)<0)
z[2] = 0.0;
   else
{
 if(x2[2]>0)
  z[2] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
  z[2] = (-b - sqrt(pow(b,2)-a*c))/a;
}

  }

 E = E / (1.0 + (E/500.0) * (1.0 - cosin));
 if(E<=0)owari=1;

 hokaku1 = hokaku(y, z, E);




}
   }
  } /* while... （これで一方は終わり） */

  /*  もう一方の交点を求める */
  E = 500.0;

  y[0] = 0.0 - distance;
  y[1] = y[0] * x[1] / x[0];
  y[2] = y[0] * x[2] / x[0];
      /* 最初に x=-(distance + length2) での交点を調べ
る */
  z2[0] = 0.0 - (distance + length2);
  z2[1] = z2[0] * x[1] / x[0];
  z2[2] = z2[0] * x[2] / x[0];
  if(0-width<=z2[1] && z2[1]<=2*height-width
       && 0-height<=z2[2] && z2[2]<=height)
      for(i=0;i<=2;++i)
z[i] = z2[i];
    if(x[1] < 0){ /* 2番目にy=2*height-widthでの交点を調べ
る */
      z2[1] = 2 * height - width;
      z2[0] = z2[1] * x[0] / x[1];
      z2[2] = z2[1] * x[2] / x[1];
      if(0-(distance+length2)<=z2[0] && z2[0]<=0-distance
 && 0-height<=z2[2] && z2[2]<=height)
for(i=0;i<=2;++i)
  z[i] = z2[i];
    }
    if(x[1] > 0){ /* 3番目にy=-widthでの交点を調べる */
      z2[1] = 0 - width;
      z2[0] = z2[1] * x[0] / x[1];
      z2[2] = z2[1] * x[2] / x[1];
      if(0-(distance+length2)<=z2[0] && z2[0]<=0-distance
 && 0-height<=z2[2] && z2[2]<=height)
for(i=0;i<=2;++i)
  z[i] = z2[i];
    }
    if(x[2] < 0){ /* 4番目にz=heightでの交点を調べる */
      z2[2] = height;
      z2[0] = z2[2] * x[0] / x[2];
      z2[1] = z2[2] * x[1] / x[2];
      if(0-(distance+length2)<=z2[0] && z2[0]<=0-distance
 && 0-width<=z2[1] && z2[1]<=2*height-width)
for(i=0;i<=2;++i)
  z[i] = z2[i];
    }
    if(x[2] > 0){ /* どれも違ってたらここが切片 */
      z2[2] = 0 - height;
      z2[0] = z2[2] * x[0] / x[2];
      z2[1] = z2[2] * x[1] / x[2];
      if(0.0-(distance+length2)<=z2[0] && z2[0]<=0.0-distance
 && 0.0-width<=z2[1] && z2[1]<=2*height-width)
for(i=0;i<=2;++i)
  z[i] = z2[i];
```

```
    }
    hokaku3 = hokaku(y, z, E);

    /* 他のNaIにぶつかってたら透過率をかける */
    if(width<=y[1])
      {
z2[1] = 0.0 - width;
z2[0] = z2[1] * x[0] / x[1];
z2[2] = z2[1] * x[2] / x[1];
touka3 = 1.0 - hokaku(y, z2, E);
hokaku3 = hokaku3 * touka3;
      }
    /* ここから、外に逃げるか光電効果が起こるまで繰り返し */

    owari = 0;
    count3 = 0;
    while(owari != 1){
      if(kettei(hokaku3) == 0) /* 捕まらなかったら終わり */
owari = 1;
      else{
if(kettei(phot(E)/(phot(E)+compt(E))) == 1)
  {
    owari = 1;
    count3 = 1; /* 光電効果したら終わり */
  }
else{ /* コンプトン散乱になった場合 */
  hojoY = 2.0;
  hojoX = 0.0;
  while(hojoY > comptH(hojoX, E)) /* 重み付き確率 */
    {
      hojoY = random2();
      hojoX = absolute(y, z) * random2();
    }
  norm = absolute(y, z);
  for(i=0;i<=2;++i) /* 反応の座標 */
    {
      houkou[i] = z[i] - y[i];
      y[i] = (z[i] - y[i]) * hojoX / norm + y[i];
    }
  hojoY = 7.0;
  cosin = 1.0;
  while(hojoY > nishina(cosin, E)) /* 重み付き確率 */
    {
      hojoY = 2.0 * random2();
      for(i=0;i<=2;++i)
x2[i] = 0;
      while((x2[0]==0 && x2[1]==0 && x2[2]==0)
    || absolute2(x2)>10000.0)
for(i=0;i<=2;++i)
      x2[i] = (double)randP(); /* γ線の方向 */
      cosin = naiseki(houkou, x2) / absolute2(x2) / norm;
    }
   /* x=-distance との交点 */
    z2[0] = 0.0 - distance - y[0];
    z2[1] = z2[0] * x2[1] / x2[0];
    z2[2] = z2[0] * x2[2] / x2[0];
    for(i=0;i<=2;++i)
      z2[i] = z2[i] + y[i];
    if(x2[0]>0 && 0.0-height<=z2[1] && z2[1]<=height
       && 0.0-height<=z2[2] && z2[2]<=height)
      for(i=0;i<=2;++i)
z[i] = z2[i];

  if(x2[0] < 0){ /* x=-distance-length2 との交点 */
    z2[0] = 0.0 - distance - length2 - y[0];
    z2[1] = z2[0] * x2[1] / x2[0];
    z2[2] = z2[0] * x2[2] / x2[0];
    for(i=0;i<=2;++i)
      z2[i] = z2[i] + y[i];
    if(0-width<=z2[1] && z2[1]<=2*height-width
       && 0-height<=z2[2] && z2[2]<=height)
      for(i=0;i<=2;++i)
z[i] = z2[i];
  }
  if(x2[1] > 0){ /* 2 番目に y=2*height-width での交点を調べ
る */
    z2[1] = 2 * height - width - y[1];
    z2[0] = z2[1] * x2[0] / x2[1];
    z2[2] = z2[1] * x2[2] / x2[1];
    for(i=0;i<=2;++i)
      z2[i] = z2[i] + y[i];
    if(0-(distance+length2)<=z2[0] && z2[0]<=0-distance
       && 0-height<=z2[2] && z2[2]<=height)
      for(i=0;i<=2;++i)
z[i] = z2[i];
  }
  if(x2[1] < 0){ /* 3番目にy=-widthでの交点を調べる */
    z2[1] = 0 - width - y[1];
    z2[0] = z2[1] * x2[0] / x2[1];
    z2[2] = z2[1] * x2[2] / x2[1];
    for(i=0;i<=2;++i)
      z2[i] = z2[i] + y[i];
    if(0-(distance+length2)<=z2[0] && z2[0]<=0-distance
       && 0-height<=z2[2] && z2[2]<=height)
      for(i=0;i<=2;++i)
z[i] = z2[i];
  }
  if(x2[2] > 0){ /* 4番目にz=heightでの交点を調べる */
    z2[2] = height - y[2];
    z2[0] = z2[2] * x2[0] / x2[2];
    z2[1] = z2[2] * x2[1] / x2[2];
    for(i=0;i<=2;++i)
      z2[i] = z2[i] + y[i];
    if(0-(distance+length2)<=z2[0] && z2[0]<=0-distance
       && 0-width<=z2[1] && z2[1]<=2*height-width)
      for(i=0;i<=2;++i)
z[i] = z2[i];
  }
  if(x2[2] < 0){ /* どれも違ってたらここが切片 */
    z2[2] = 0 - height - y[2];
    z2[0] = z2[2] * x2[0] / x2[2];
    z2[1] = z2[2] * x2[1] / x2[2];
    for(i=0;i<=2;++i)
      z2[i] = z2[i] + y[i];
    if(0.0-(distance+length2)<=z2[0] && z2[0]<=0.0-distance
       && 0.0-width<=z2[1] && z2[1]<=2*height-width)
      for(i=0;i<=2;++i)
z[i] = z2[i];
  }
  E = E / (1.0 + (E/500.0) * (1.0 - cosin));
  if(E<=0)owari=1;

  hokaku3 = hokaku(y, z, E);

}
      }
    }
    count = count + count1 * count3;
  }
}
      } /* for(m=0;m<N;++m) */

answer =2* (double)count / (double)N;

printf("probability is %lf\n", answer);


}


/* 乱数 (-10000 < random < 10000) */
int randP()
{
  int x=0;
  x = (rand() - (RAND_MAX /2)) % 10000;
  return x;
}

/* 乱数(0〜1) */
double random2()
{
  int x;
  double y;

  x = rand() % 10000;
  y = (double)x / 10000.0;
  return y;
}

/* ３次元ベクトルの内積 */
double naiseki(double x[], double y[])
{
  double s;

  s = x[0] * y[0] + x[1] * y[1] + x[2] * y[2];
  return s;
}
/* ３次元ベクトルのノルム */
double absolute(double x[], double y[])
{
  double L;

  L = sqrt(pow((x[0]-y[0]),2)+pow((x[1]-y[1]),2)+pow((x[2]-y[2]),2));
  return L;
}

/* ３次元ベクトルのノルム２ */
double absolute2(double x[])
{
  double L;

  L = sqrt(pow(x[0],2)+pow(x[1],2)+pow(x[2],2));
  return L;
}

/* 確率に応じて白黒つける */
int kettei(double kakuritsu)
{
  int x;
  double y;

  x = rand() % 10000;
  y = (double)x / 10000.0;
  if(y <= kakuritsu)
```

```
    return 1;
  else
    return 0;
}

/* 光電効果＋コンプトン散乱の捕獲率 */
double hokaku(double a[], double b[], double E){
  double way;
  double hokaku;

  way = absolute(a,b);
  hokaku = 1 - exp((0.0-phot(E)-compt(E)) * way);
  /* 単位はmm単位 */
  return hokaku;
 }

/* 光電効果の吸収係数（N*全断面積） */
double phot(double E)
{
  double ratio;
  double mu;

  ratio = 500.0 / E;
  mu = const1 * pow(ratio, 3.5);
  return mu;
}

/* コンプトン散乱の吸収係数（N*全断面積） */
double compt(double E)
{
  double ratio;
  double mu;

  ratio = E / 500.0;
  mu = const2 * ((1.0+ratio)/pow(ratio,3)*(2.0*ratio*(1.0+ratio)
/(1.0+2.0*ratio)-log(1.0+2.0*ratio))+1.0/(2.0*ratio)
*log(1.0+2.0*ratio)-(1.0+3.0*ratio)/pow((1.0+2.0*ratio),2.0));
  return mu;
}
```

```
/* コンプトン散乱の捕獲率 */
double comptH(double nagasa, double E)
{
  double hokaku;

  hokaku = exp((0.0-compt(E))*nagasa);
  return hokaku;
}

/* コンプトン散乱θ依存性 */
double nishina(double cosin, double E)
{
  double ratio;
  double BibunSanranDanmenseki;

  ratio = E / 500.0;
  BibunSanranDanmenseki = 1.0 / pow((1.0 + ratio * (1.0 - cosin)),2)
* (1.0 + pow(cosin,2) + pow((ratio*(1.0-cosin)),2)
/ (1.0 + ratio * (1.0 - cosin)));

  return BibunSanranDanmenseki;
}
```

- 3γ の検出器にかかる確率を求める際に使ったプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>

#define const1 0.021075
#define const2 0.046937

int randE();
int randP();
```

```
double random2();
double naiseki(double x[], double y[]);
double absolute(double x[], double y[]);
double absolute2(double x[]);
int kettei(double kakuritsu);
double hokaku(double x[], double y[], double E);
double phot(double E);
double compt(double E);
double comptH(double nagasa, double E);
double nishina(double cosin, double E);

main()
{

  int h,m, i, N;
  int count=0;
  int countF=0, countR=0, countL=0;
  int owari=0;
  double Px[3], Py[3], Pz[3];
  double temp1[3], temp2[3], temp3[3];
  double Ex=0.0, Ey=0.0, Ez=0.0, aPx, aPy;
  double f1[3], r1[3], l1[3];
  double f2[3], r2[3], l2[3];
  double f3[3], r3[3], l3[3];
  double houkou[3];
  double a, b, c;
  double norm;
  double cosin;
  double hojoX, hojoY;
  double hosei;
  double touka;
  double hokakuF, hokakuR, hokakuL;
  int cut=1;
  double hokakuSUM=0.0;
  double distanceF=30.0, widthF=10.0, heightF=29.0;
  double distanceR=-10.0, widthR=27.5, heightR=27.5;
  double distanceL=10.0, radiusL=29.0;
  double lengthF=55.0, lengthR=55.0, lengthL=55.0;
  double answer;
```

```
 srand((unsigned int)time(NULL));
 scanf("%d", &N);

  for(m=0;m<N;++m)
   {
for(i=0;i<3;++i){
 Px[i]=0.0;
 Py[i]=0.0;
 Pz[i]=0.0;
}

   /* 更に更に新しいランダムプログラム */

   while((Px[0]==0.0 && Px[1]==0.0 && Px[2]==0.0)
  || (Py[0]==0.0 && Py[1]==0.0 && Py[2]==0.0)
  || (Pz[0]==0.0 && Pz[1]==0.0 && Pz[2]==0.0)
  )
{
 for(i=0;i<=2;++i)
  {
   temp1[i] = (double)randP();
   temp2[i] = (double)randP();
   temp3[i] = (double)randP();
  }
 while(absolute2(temp1)>10000.0
|| absolute2(temp2)>10000.0
|| absolute2(temp3)>10000.0
)
  {
   for(i=0;i<=2;++i)
{
 temp1[i] = (double)randP();
 temp2[i] = (double)randP();
 temp3[i] = (double)randP();
}
  }
 for(i=0;i<=2;++i)
  {
   Px[i] = temp1[i] - temp2[i];
   Py[i] = temp2[i] - temp3[i];
   Pz[i] = temp3[i] - temp1[i];
  }
   }

   hosei = (absolute2(Px) + absolute2(Py) + absolute2(Pz)) / 1000.0;

   Ex = absolute2(Px) / hosei;
   Ey = absolute2(Py) / hosei;
   Ez = absolute2(Pz) / hosei;

   for(i=0;i<=2;++i)
{
 Px[i] = Px[i] / hosei;
 Py[i] = Py[i] / hosei;
 Pz[i] = Pz[i] / hosei;
}

   /* ここまでが更に更に新しいランダムプログラム */

if(Px[0]>0.0 && Py[1]<0.0 && Pz[1]>0.0){
 f1[0] = distanceF;
 f1[1] = f1[0] * Px[1] / Px[0];
 f1[2] = f1[0] * Px[2] / Px[0];
 r1[1] = distanceR;
 r1[0] = r1[1] * Py[0] / Py[1];
 r1[2] = r1[1] * Py[2] / Py[1];
 l1[1] = distanceL;
 l1[0] = distanceL * Pz[0] / Pz[1];
 l1[2] = distanceL * Pz[2] / Pz[1];

         if((sqrt(pow(f1[1],2)+pow(f1[2],2))<=radiusL) &&
    (0.0-170.0+widthR<=r1[0]) && (r1[0]<=widthR) &&
    (0.0-heightR<=r1[2]) && (r1[2]<=heightR) &&
    (sqrt(pow(l1[0],2)+pow(l1[2],2)) <= radiusL)){

  /* Px の x=distanceF + lengthF での切片を調べる */
  f2[0] = distanceF + lengthF;
  f2[1] = f2[0] * Px[1] / Px[0];
  f2[2] = f2[0] * Px[2] / Px[0];
  if(sqrt(pow(f2[1],2) + pow(f2[2],2))<=radiusL)
   hokakuF = hokaku(f1, f2, Ex);
  else{ /* Px の円筒との切片 */
   f2[0] = Px[0] * radiusL / sqrt(pow(Px[1],2) + pow(Px[2],2));
   f2[1] = f2[0] * Px[1] / Px[0];
   f2[2] = f2[0] * Px[2] / Px[0];
   hokakuF = hokaku(f1, f2, Ex);
  }
  /* 他の NaI にぶつかっていたら透過率をかける */
  if(f1[1] < 0.0-widthF){
   f3[1] = 0.0 - widthF;
   f3[0] = f3[1] * Px[0] / Px[1];
   f3[2] = f3[1] * Px[2] / Px[1];
   touka = 1.0 - hokaku(f1, f3, Ex);
   hokakuF = hokakuF * touka;
  }
  if(widthF < f1[1]){
   f3[1] = widthF;
   f3[0] = f3[1] * Px[0] / Px[1];
   f3[2] = f3[1] * Px[2] / Px[1];
   touka = 1.0 - hokaku(f1, f3, Ex);
   hokakuF = hokakuF * touka;
  }

  /* ここから、外に逃げるか光電効果が起こるまで繰り返し */
  owari = 0;
  countF = 0;
  while(owari != 1){
   if(kettei(hokakuF) == 0) /* 捕まらなかったら終わり */
owari = 1;
   else{
if(kettei(phot(Ex)/(phot(Ex)+compt(Ex))) == 1)
 {
  owari = 1;
  countF = 1; /* 光電効果したら終わり */
 }
else{ /* コンプトン散乱になった場合 */
```

```
  hojoY = 2.0;
  hojoX = 0.0;
  while(hojoY > comptH(hojoX, Ex)) /* 重み付き確率 */
    {
      hojoY = random2();
      hojoX = absolute(f1, f2) * random2();
    }
  norm = absolute(f1, f2);
  for(i=0;i<=2;++i) /* 反応の座標 */
    {
      houkou[i] = f2[i] - f1[i];
      f1[i] = (f2[i] - f1[i]) * hojoX / norm + f1[i];
    }

  hojoY = 7.0;
  cosin = 1.0;
  while(hojoY > nishina(cosin, Ex)) /* 重み付き確率 */
    {
      hojoY = 2.0 * random2();
      for(i=0;i<=2;++i)
Px[i] = 0;
      while((Px[0]==0 && Px[1]==0 && Px[2]==0)
    || absolute2(Px)>10000.0)
for(i=0;i<=2;++i)
      Px[i] = (double)randP(); /* γ線の方向 */
      cosin = naiseki(houkou, Px) / absolute2(Px) / norm;
    }
  /* もう一度交点を求める */
  for(i=0;i<=2;++i)
    f2[i] = 0.0;
  if(Px[0] < 0){ /* x=distanceF での交点 */
    f3[0] = distanceF - f1[0];
    f3[1] = f3[0] * Px[1] / Px[0];
    f3[2] = f3[0] * Px[2] / Px[0];
    for(i=0;i<=2;++i)
      f3[i] = f3[i] + f1[i];
    if(sqrt(pow(f3[1],2)+pow(f3[2],2))<=radiusL)
      for(i=0;i<=2;++i)
f2[i] = f3[i];
```

```
 }
 if(Px[0] > 0){ /* x=distanceF+lengthF での交点 */
  f3[0] = distanceF + lengthF - f1[0];
  f3[1] = f3[0] * Px[1] / Px[0];
  f3[2] = f3[0] * Px[2] / Px[0];
  for(i=0;i<=2;++i)
   f3[i] = f3[i] + f1[i];
  if(sqrt(pow(f3[1],2)+pow(f3[2],2))<=radiusL)
   for(i=0;i<=2;++i)
f2[i] = f3[i];
 }
 if(f2[0]==0 && f2[1]==0 && f2[2]==0)
  {
   a = pow(Px[1],2) + pow(Px[2],2);
   b = Px[1]*(Px[0]*f1[1]-Px[1]*f1[0])+Px[2]
*(Px[0]*f1[2]-Px[2]*f1[0]);
   c = pow((Px[0]*f1[1]-Px[1]*f1[0]),2)
+pow((Px[0]*f1[2]-Px[2]*f1[0]),2)-pow(radiusL*Px[0],2);
   if((pow(b,2)-a*c)<0)
f2[0] = 0.0;
   else
{
 if(Px[0]>0)
  f2[0] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
  f2[0] = (-b - sqrt(pow(b,2)-a*c))/a;
}
   b = Px[2]*(Px[1]*f1[2]-Px[2]*f1[1]);
   c = pow((Px[1]*f1[2]-Px[2]*f1[1]),2)
-pow(radiusL*Px[1],2);
   if((pow(b,2)-a*c)<0)
f2[1] = 0.0;
   else
{
 if(Px[1]>0)
  f2[1] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
  f2[1] = (-b - sqrt(pow(b,2)-a*c))/a;
}
```

```
   b = Px[1]*(Px[2]*f1[1]-Px[1]*f1[2]);
   c = pow((Px[2]*f1[1]-Px[1]*f1[2]),2)
-pow(radiusL*Px[2],2);
   if((pow(b,2)-a*c)<0)
f2[2] = 0.0;
   else
{
 if(Px[2]>0)
  f2[2] = (-b+sqrt(pow(b,2)-a*c))/a;
 else
  f2[2] = (-b-sqrt(pow(b,2)-a*c))/a;
}
  }
 Ex = Ex / (1.0 + (Ex/500.0) * (1.0 - cosin));
 if(Ex<=0)owari = 1;
 hokakuF = hokaku(f1, f2, Ex);

}
   }
  } /* while... （これで1番のNaIは終わり） */

  /* Pzのy=distanceL + lengthLでの切片を調べる */
  l2[1] = distanceL + lengthL;
  l2[0] = l2[1] * Pz[0] / Pz[1];
  l2[2] = l2[1] * Pz[2] / Pz[1];
  if(sqrt(pow(l2[0],2) + pow(l2[2],2))<=radiusL)
   hokakuL = hokaku(l1, l2, Ez);
  else{ /* Pzの円筒との切片 */
   l2[1] = Pz[1] * radiusL / sqrt(pow(Pz[0],2) + pow(Pz[2],2));
   l2[0] = l2[1] * Pz[0] / Pz[1];
   l2[2] = l2[1] * Pz[2] / Pz[1];
   hokakuL = hokaku(l1, l2, Ez);
  }
  /* ここから、外に逃げるか光電効果が起こるまで繰り返し */
  owari = 0;
  countL = 0;
  while(owari != 1){
   if(kettei(hokakuL) == 0) /* 捕まらなかったら終わり */
owari = 1;
```

```
   else{
if(kettei(phot(Ez)/(phot(Ez)+compt(Ez))) == 1)
 {
  owari = 1;
  countL = 1; /* 光電効果したら終わり */
 }
else{ /* コンプトン散乱になった場合 */
 hojoY = 2.0;
 hojoX = 0.0;
 while(hojoY > comptH(hojoX, Ez)) /* 重み付き確率 */
  {
   hojoY = random2();
   hojoX = absolute(l1, l2) * random2();
  }
 norm = absolute(l1, l2);
 for(i=0;i<=2;++i) /* 反応の座標 */
  {
   houkou[i] = l2[i] - l1[i];
   l1[i] = (l2[i] - l1[i]) * hojoX / norm + l1[i];
  }

 hojoY = 7.0;
 cosin = 1.0;
 while(hojoY > nishina(cosin, Ez)) /* 重み付き確率 */
  {
   hojoY = 2.0 * random2();
   for(i=0;i<=2;++i)
Pz[i] = 0;
   while((Pz[0]==0 && Pz[1]==0 && Pz[2]==0)
  || absolute2(Pz)>10000.0)
for(i=0;i<=2;++i)
   Pz[i] = (double)randP(); /* γ線の方向 */
   cosin = naiseki(houkou, Pz) / absolute2(Pz) / norm;
  }
 /* もう一度交点を求める */
 for(i=0;i<=2;++i)
  l2[i] = 0.0;
 if(Pz[1] < 0){ /* y=distanceL での交点 */
  l3[1] = distanceL - l1[1];
```

```
    l3[0] = l3[1] * Pz[0] / Pz[1];
    l3[2] = l3[1] * Pz[2] / Pz[1];
    for(i=0;i<=2;++i)
      l3[i] = l3[i] + l1[i];
    if(sqrt(pow(l3[0],2)+pow(l3[2],2))<=radiusL)
      for(i=0;i<=2;++i)
l2[i] = l3[i];
  }
  if(Pz[1] > 0){ /* y=distance+lengthでの交点 */
    l3[1] = distanceL + lengthL - l1[1];
    l3[0] = l3[1] * Pz[0] / Pz[1];
    l3[2] = l3[1] * Pz[2] / Pz[1];
    for(i=0;i<=2;++i)
      l3[i] = l3[i] + l1[i];
    if(sqrt(pow(l3[0],2)+pow(l3[2],2))<=radiusL)
      for(i=0;i<=2;++i)
l2[i] = l3[i];
  }
  if(l2[0]==0 && l2[1]==0 && l2[2]==0)
    {
      a = pow(Pz[0],2) + pow((0.0-Pz[2]),2);
      b = Pz[0]*(Pz[1]*l1[0]-Pz[0]*l1[1])-Pz[2]
*(Pz[2]*l1[1]-Pz[1]*l1[2]);
      c = pow((Pz[1]*l1[0]-Pz[0]*l1[1]),2)
+pow((Pz[2]*l1[1]-Pz[1]*l1[2]),2)-pow(radiusL*Pz[1],2);
      if((pow(b,2)-a*c)<0)
l2[1] = 0.0;
      else
{
  if(Pz[1]>0)
      l2[1] = (-b + sqrt(pow(b,2)-a*c))/a;
  else
      l2[1] = (-b - sqrt(pow(b,2)-a*c))/a;
}
      b = Pz[2]*(Pz[0]*l1[2]-Pz[2]*l1[0]);
      c = pow((Pz[0]*l1[2]-Pz[2]*l1[0]),2)
-pow(radiusL*Pz[0],2);
      if((pow(b,2)-a*c)<0)
l2[0] = 0.0;
```

```
   else
{
 if(Pz[0]>0)
  l2[0] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
  l2[0] = (-b - sqrt(pow(b,2)-a*c))/a;
}
   b = Pz[0]*(Pz[2]*l1[0]-Pz[0]*l1[2]);
   c = pow((Pz[2]*l1[0]-Pz[0]*l1[2]),2)
-pow(radiusL*Pz[2],2);
   if((pow(b,2)-a*c)<0)
l2[2] = 0.0;
   else
{
 if(Pz[2]>0)
  l2[2] = (-b + sqrt(pow(b,2)-a*c))/a;
 else
  l2[2] = (-b - sqrt(pow(b,2)-a*c))/a;
}
  }
 Ez = Ez / (1.0 + (Ez/500.0) * (1.0 - cosin));
 if(Ez<=0)owari = 1;
 hokakuL = hokaku(l1, l2, Ez);

}
   }
  } /* while... （これで２番のNaI終わり） */

  /* Py の y=distanceR - lengthR での切片を調べる */
  for(i=0;i<=2;++i)
   r2[i] = 0.0;
  r3[1] = distanceR - lengthR;
  r3[0] = r3[1] * Py[0] / Py[1];
  r3[2] = r3[1] * Py[2] / Py[1];
  if(-170.0+widthR<=r3[0] && r3[0]<=widthR
    && 0.0-heightR<=r3[2] && r3[2]<=heightR)
   for(i=0;i<=2;++i)
r2[i] = r3[i];
  if(Py[0] > 0){ /* Py の x=widthR での切片を調べる */
```

```
      r3[0] = widthR;
      r3[1] = r3[0] * Py[1] / Py[0];
      r3[2] = r3[0] * Py[2] / Py[0];
      if(distanceR-lengthR<=r3[1] && r3[1]<=distanceR
 && 0.0-heightR<=r3[2] && r3[2]<=heightR)
for(i=0;i<=2;++i)
  r2[i] = r3[i];
    }
    if(Py[2] > 0){ /* Pyのz=heightRでの切片を調べる */
      r3[2] = heightR;
      r3[0] = r3[2] * Py[0] / Py[2];
      r3[1] = r3[2] * Py[1] / Py[2];
      if(-170.0+widthR<=r3[0] && r3[0]<=widthR
 && distanceR-lengthR<=r3[1] && r3[1]<=distanceR)
for(i=0;i<=2;++i)
  r2[i] = r3[i];
    }
    if(Py[2] < 0){ /* Pyのz=-heightRでの切片を調べる */
      r3[2] = 0.0 - heightR;
      r3[0] = r3[2] * Py[0] / Py[2];
      r3[1] = r3[2] * Py[1] / Py[2];
      if(-170.0+widthR<=r3[0] && r3[0]<=widthR
 && distanceR-lengthR<=r3[1] && r3[1]<=distanceR)
for(i=0;i<=2;++i)
  r2[i] = r3[i];
    }
    if(r2[0]==0 && r2[1]==0 && r2[2]==0){
      /* 全部はずれてたらPyはx=-170+widthRで交わる */
      r2[0] = -170.0 + widthR;
      r2[1] = r2[0] * Py[1] / Py[0];
      r2[2] = r2[0] * Py[2] / Py[0];
    }
    hokakuR = hokaku(r1, r2, Ey);
    /* ３番と交わってたら透過率をかける */
    if(r1[0]<=0.0-widthR)
      {
r3[0] = 0.0 - widthR;
r3[1] = r3[0] * Py[1] / Py[0];
r3[2] = r3[0] * Py[2] / Py[0];
```

```
touka = 1.0 - hokaku(r1, r2, Ey);
hokakuR = hokakuR * touka;
      }
    /* ここから、外に逃げるか光電効果が起こるまで繰り返し */
    owari = 0;
    countR = 0;
    while(owari != 1){
      if(kettei(hokakuR) == 0) /* 捕まらなかったら終わり */
owari = 1;
      else{
if(kettei(phot(Ey)/(phot(Ey)+compt(Ey))) == 1)
  {
    owari = 1;
    countR = 1; /* 光電効果したら終わり */
  }
else{ /* コンプトン散乱になった場合 */
  hojoY = 2.0;
  hojoX = 0.0;
  while(hojoY > comptH(hojoX, Ey)) /* 重み付き確率 */
    {
      hojoY = random2();
      hojoX = absolute(r1, r2) * random2();
    }
  norm = absolute(r1, r2);
  for(i=0;i<=2;++i) /* 反応の座標 */
    {
      houkou[i] = r2[i] - r1[i];
      r1[i] = (r2[i] - r1[i]) * hojoX / norm + r1[i];
    }

  hojoY = 7.0;
  cosin = 1.0;
  while(hojoY > nishina(cosin, Ey)) /* 重み付き確率 */
    {
      hojoY = 2.0 * random2();
      for(i=0;i<=2;++i)
Py[i] = 0;
      while((Py[0]==0 && Py[1]==0 && Py[2]==0)
    || absolute2(Py)>10000.0)
```

```
for(i=0;i<=2;++i)
   Py[i] = (double)randP(); /* γ線の方向 */
   cosin = naiseki(houkou, Py) / absolute2(Py) / norm;
  }
 /* もう一度交点を求める */
 for(i=0;i<=2;++i)
  r2[i] = 0.0;
 if(Py[1] > 0){ /* y=distanceRでの交点 */
  r3[1] = distanceR - r1[1];
  r3[0] = r3[1] * Py[0] / Py[1];
  r3[2] = r3[1] * Py[2] / Py[1];
  for(i=0;i<=2;++i)
   r3[i] = r3[i] + r1[i];
  if(widthR-170.0<=r3[0] && r3[0]<=widthR
    && 0.0-heightR<=r3[2] && r3[2]<=heightR)
   for(i=0;i<=2;++i)
r2[i] = r3[i];
 }
 if(Py[1] < 0){ /* y=distanceR-lengthRでの交点 */
  r3[1] = distanceR - lengthR - r1[1];
  r3[0] = r3[1] * Py[0] / Py[1];
  r3[2] = r3[1] * Py[2] / Py[1];
  for(i=0;i<=2;++i)
   r3[i] = r3[i] + r1[i];
  if(-170.0+widthR<=r3[0] && r3[0]<=widthR
    && 0.0-heightR<=r3[2] && r3[2]<=heightR)
   for(i=0;i<=2;++i)
r2[i] = r3[i];
 }
 if(Py[0] > 0){ /* Pyのx=widthRでの切片を調べる */
  r3[0] = widthR - r1[0];
  r3[1] = r3[0] * Py[1] / Py[0];
  r3[2] = r3[0] * Py[2] / Py[0];
  for(i=0;i<=2;++i)
   r3[i] = r3[i] + r1[i];
  if(distanceR-lengthR<=r3[1] && r3[1]<=distanceR
    && 0.0-heightR<=r3[2] && r3[2]<=heightR)
   for(i=0;i<=2;++i)
r2[i] = r3[i];
```

```
  }
  if(Py[2] > 0){ /* Py の z=heightR での切片を調べる */
    r3[2] = heightR - r1[2];
    r3[0] = r3[2] * Py[0] / Py[2];
    r3[1] = r3[2] * Py[1] / Py[2];
    for(i=0;i<=2;++i)
      r3[i] = r3[i] + r1[i];
    if(-170.0+widthR<=r3[0] && r3[0]<=widthR
       && distanceR-lengthR<=r3[1] && r3[1]<=distanceR)
      for(i=0;i<=2;++i)
r2[i] = r3[i];
  }
  if(Py[2] < 0){ /* Py の z=-heightR での切片を調べる */
    r3[2] = 0.0 - heightR - r1[2];
    r3[0] = r3[2] * Py[0] / Py[2];
    r3[1] = r3[2] * Py[1] / Py[2];
    for(i=0;i<=2;++i)
      r3[i] = r3[i] + r1[i];
    if(-170.0+widthR<=r3[0] && r3[0]<=widthR
       && distanceR-lengthR<=r3[1] && r3[1]<=distanceR)
      for(i=0;i<=2;++i)
r2[i] = r3[i];
  }
  if(r2[0]==0 && r2[1]==0 && r2[2]==0){
    /* 全部はずれてたら Py は x=-170+widthR で交わる */
    r2[0] = -170.0 + widthR - r1[0];
    r2[1] = r2[0] * Py[1] / Py[0];
    r2[2] = r2[0] * Py[2] / Py[0];
    for(i=0;i<=2;++i)
      r2[i] = r2[i] + r1[i];
  }
  Ey = Ey / (1.0 + (Ey/500.0) * (1.0 - cosin));
  if(Ey<=0)owari = 1;
  hokakuR = hokaku(r1, r2, Ey);

}
      }
    }/* while... ４番の NaI 終わり */
    /* カットを入れる */
    cut = 0;
    if((Ex<470.0)&&(Ey<470.0)&&(Ez<470.0)
       &&!(((Ex+Ey)>490.0)&&((Ex+Ey)<580.0))
       &&!(((Ex+Ez)>460.0)&&((Ex+Ez)<580.0))
       &&!(((Ey+Ez)>460.0)&&((Ey+Ez)<580.0))
       )
      cut = 1;
    count = count + countF * countL * countR * cut;
  }
}
      }/* for(m=0;m<N;++m) */
    answer = 6.0 * (double)count / (double)N;
    printf("probability is %lf", answer);
}



/* get random Energy (0 < random < 500) */
randE()
{
  int x=0;
  x = rand() % 500;
  return x;
}

/* get random Momentum (-10000 < random <10000) */
randP()
{
  int x=0;
  x = (rand() - (RAND_MAX /2)) % 10000;
  return x;
}

/* 乱数(0～1) */
double random2()
{
  int x;
  double y;
  x = rand() % 10000;
  y = (double)x / 10000.0;
  return y;
}

double naiseki(double x[], double y[])
{
  double s;

  s = x[0] * y[0] + x[1] * y[1] + x[2] * y[2];
  return s;
}

/* ３次元ベクトルのノルム */
double absolute(double x[], double y[])
{
  double L;

  L = sqrt(pow((x[0]-y[0]),2)+pow((x[1]-y[1]),2)+pow((x[2]-y[2]),2));
  return L;
}

/* ３次元ベクトルのノルム２ */
double absolute2(double x[])
{
  double L;

  L = sqrt(pow(x[0],2)+pow(x[1],2)+pow(x[2],2));
  return L;
}

/* 確率に応じて白黒つける */
int kettei(double kakuritsu)
{
  int x;
  double y;

  x = rand() % 10000;
  y = (double)x / 10000.0;
  if(y <= kakuritsu)
    return 1;
  else
    return 0;
}

/* 光電効果＋コンプトン散乱の捕獲率 */
double hokaku(double a[], double b[], double E){
  double way;
  double hokaku;

  way = absolute(a,b);
  hokaku = 1 - exp((0.0-phot(E)-compt(E)) * way);
  /* 単位はmm単位 */
  return hokaku;
 }

/* 光電効果の吸収係数（N*全断面積） */
double phot(double E)
{
  double ratio;
  double mu;

  ratio = 500.0 / E;
  mu = const1 * pow(ratio, 3.5);
  return mu;
}

/* コンプトン散乱の吸収係数（N*全断面積） */
double compt(double E)
{
  double ratio;
  double mu;

  ratio = E / 500.0;
  mu = const2 * ((1.0+ratio)/pow(ratio,3)*(2.0*ratio*(1.0+ratio)
/(1.0+2.0*ratio)-log(1.0+2.0*ratio))+1.0/(2.0*ratio)
*log(1.0+2.0*ratio)-(1.0+3.0*ratio)/pow((1.0+2.0*ratio),2.0));
```

```
  return mu;
}

/* コンプトン散乱の捕獲率 */
double comptH(double nagasa, double E)
{
  double hokaku;

  hokaku = exp((0.0-compt(E))*nagasa);
  return hokaku;
}

/* コンプトン散乱θ依存性 */
double nishina(double cosin, double E)
{
  double ratio;
  double BibunSanranDanmenseki;

  ratio = E / 500.0;
  BibunSanranDanmenseki = 1.0 / pow((1.0 + ratio * (1.0 - cosin)),2)
* (1.0 + pow(cosin,2) + pow((ratio*(1.0-cosin)),2)
/ (1.0 + ratio * (1.0 - cosin)));

  return BibunSanranDanmenseki;
}
```