5.25 モンテカルロシミュレーション 分散減少法(3)　　ver. 0.1

層化抽出法 Stratified Sampling

この方法はその名前からわかるように、基本的には、分布を細い短冊状に切って、その区間から１つないし複数のサンプルを取るものです。分布全域にわたって、万遍なく散らばりが広がってくれる利点をもっています。２種類ほど説明をしたいのですが、１つには０と１の間の区間を乱数の数と同じ数のｎ等分をした上でその中点を確定した値として取っていくものです。もう１つは、同じく区間をｎ等分した上で、そのそれぞれの区間から１つないしは複数個のサンプルをそれぞれランダムに行なうものです。

それぞれの区間の中点をサンプリングする方法

まずは第１番目のｎ等分した区間の中点を代表値としてサンプルしていく方法を見ていきましょう。その中点の求め方は具体的には、**（ｉ－0.5）÷ｎ**　で求められます。ただし、ここでｉは 1,2,3,….,n であるとします。

プログラム

```
new; cls;
n=10;

x=zeros(n,1);
i=1;
do while i<=n;
    x[i]=(i-0.5)/n;
    i=i+1;
endo;
print x;
```

画面表示

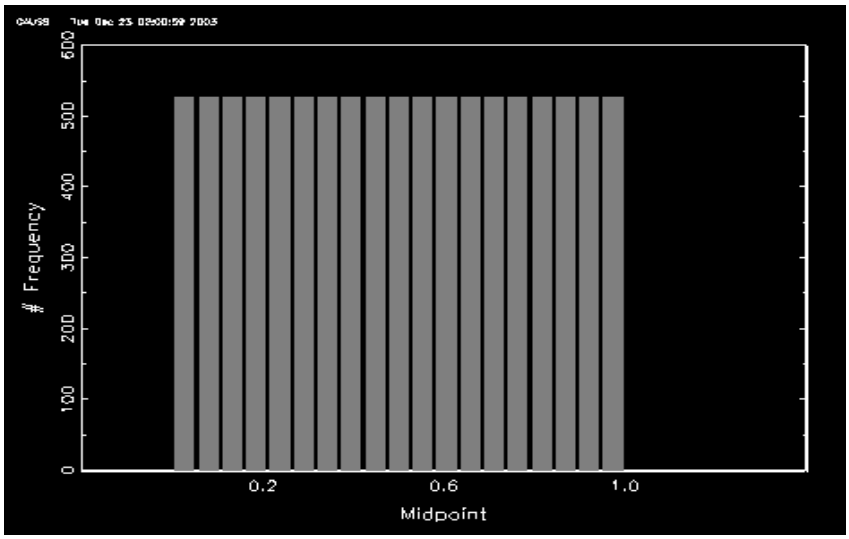
```
      0.050000000
       0.15000000
       0.25000000
       0.35000000
       0.45000000
       0.55000000
       0.65000000
       0.75000000
       0.85000000
       0.95000000
```

この例は、０と１の間の区間を n=10 等分した上で、その中点を計算しています。さらにｎを大きくして、さらに細かくｎ等分してやった中点をヒストグラムでプロットしてやるには次のようになります。

プログラム

```
new; cls;
n=10000;

x=zeros(n,1);
i=1;
do while i<=n;
    x[i]=(i-0.5)/n;
    i=i+1;
endo;
library pgraph;
graphset;
call hist(x,19);
```

画面表示



きれいな０と１の間の一様分布を示しています。なお、これはｎの数と一対一に対応した確定値であって、乱数ではありません。また、このままでは、小さい方から大きな方に順に並んでいます。

これはこれですべて確定した値の０と１の間を小さいほうから大きい方へ並べたものなのですが、GAUSS 関数の０と１の間の一様乱数を用いて、その並びをランダム化したものが結果となるようにしたものが、以下の procedure です。

プログラム

```
new; cls;
n=5000;
print U01(n);

proc U01(n);
    local x,i,y,index;
    x=zeros(n,1);
    i=1;
    do while i<=n;
        x[i]=(i-0.5)/n;
        i=i+1;
    endo;
    y=zeros(n,1);
    i=1;
    do while i<=n;
        index=ceil((n-(i-1))*rndu(1,1));
        y[i]=x[index];
        x=delif(x,x.==x[index]);
        i=i+1;
    endo;
    retp(y);
endp;
```

procesure 内部の前半部はこれまでと同じくｎ等分したものの中点を求めています。それがｘの列ベクトル。さらにこれをランダムに並べ替えたものがｙの列ベクトルで、これをリターンとして外に出しています。

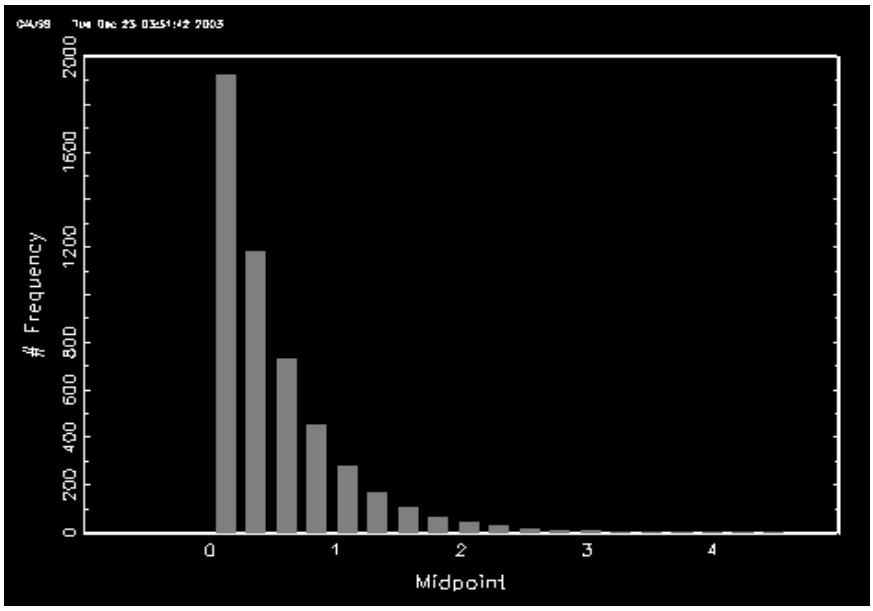
　このやり方は、逆関数法を用いてある分布にしたがう散らばりを作り出すのに重宝されます。上の U01(n)をそのまま、その分布の逆関数に代入してやるだけでよいわけです。例えば、exponential 分布のケースでは、次のようになります。

プログラム

```
new; cls;
n=5000;
lambda=2;
Xe=-ln(1-U01(n))/lambda;
library pgraph;
```

```
graphset;
call hist(Xe,19);

proc U01(n);
    local x,i,y,index;
    x=zeros(n,1);
    i=1;
    do while i<=n;
        x[i]=(i-0.5)/n;
        i=i+1;
    endo;
    y=zeros(n,1);
    i=1;
    do while i<=n;
        index=ceil((n-(i-1))*rndu(1,1));
        y[i]=x[index];
        x=delif(x,x.==x[index]);
        i=i+1;
    endo;
    retp(y);
endp;
```

グラフ表示



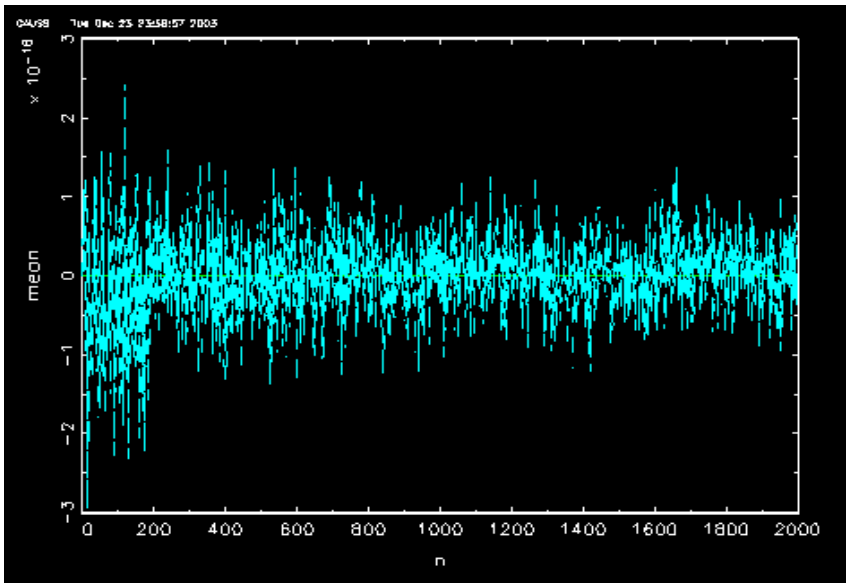
同じことは、その他の逆関数がわかっている分布にも適用できます。乱数ではないので、ｎの大きさに応じて、いつも同じ形、同じ統計量の分布になります。

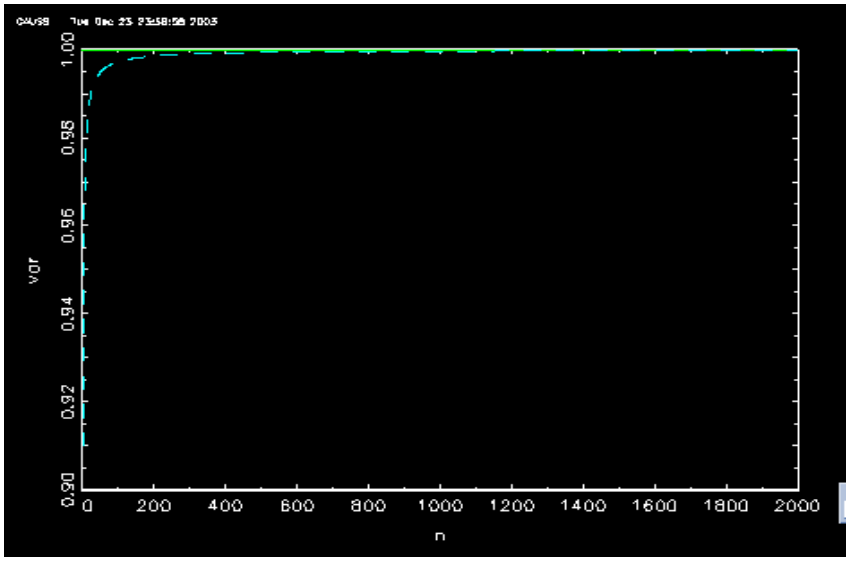
GAUSS の組込み関数には、標準正規分布の逆関数を求める cdfni ほか、数は最小限しかありませんが装備されているので、それを用いて変換した分布がｎの大きさに応じてどのように変化するのかを確かめてみます。

プログラム（長時間所要）

```
new; cls;
nmax=2000;
call Nsim(nmax);

proc Nsim(nmax);
    local m,var,i,x;
    m=zeros(nmax,1);
    var=zeros(nmax,1);
    i=1;
    do while i<=nmax;
        x=cdfni(U01(i));
        m[i]=meanc(x);
        var[i]=vcx(x);
        i=i+1;
    endo;
    library pgraph;
    graphset;
    pqgwin auto;
    xlabel("n");
    ylabel("mean");
    xy(seqa(1,1,nmax),zeros(nmax,1)~m);
    ylabel("var");
    xy(seqa(1,1,nmax),ones(nmax,1)~var);
    retp(m);
endp;

proc U01(n);
    local x,i,y,index;
    x=zeros(n,1);
    i=1;
    do while i<=n;
        x[i]=(i-0.5)/n;
```

```
        i=i+1;
    endo;
    y=zeros(n,1);
    i=1;
    do while i<=n;
        index=ceil((n-(i-1))*rndu(1,1));
        y[i]=x[index];
        x=delif(x,x.==x[index]);
        i=i+1;
    endo;
    retp(y);
endp;
```





上の結果は、プロットするのにかなりの時間を要しますが、nmax=の数字を小さくすれば、その回数までのプロットなり所要時間は減少します。また、後半の procedure の中のそのまた後半部分は単に順番をランダムにしているだけで、結果には影響を与えませんから、その部分を無効にしてｘをリターンにしてもかまいません。見てわかるように、標準正規分布の真の平均値ゼロのまわりにシミュレーションの結果は散らばっています。少ない回数、例えば数十程度のｎの値に対してもかなりの精度を最初からもっていますが、その後はｎをどんなに増してみても、さほどの分散減少効果が表れていないことがわかります。また、分散の方は真の値１に比較的少ない回数で収束します。小さいｎでも最初からかなりの精度で推定量、例えば分布の平均値や分散を求められることがうかがえます。

それぞれの区間からランダムにサンプリングする方法

　今度は、第２の方法である、ｎ区間に分布を区分した上で、そのそれぞれの区間から１つないし複数のサンプルを行なうやり方を見ていきます。

プログラム

```
new; cls;
n=10;

x=zeros(n,1);
i=1;
do while i<=n;
    x[i]=1/n*rndu(1,1)+(i-1)/n;
    i=i+1;
endo;
print x;
```

画面表示

```
      0.076743703
       0.13644011
       0.20595451
       0.36156771
       0.41782318
       0.55465621
       0.65493910
       0.72551855
       0.88498724
       0.90761012
```

この方法の場合、上のように、n=10 の場合には、０と１を１０等分するのですから、０と 0.1 の間から一様乱数にしたがって１個、0.2 と 0.3 の間からも１個、同じことを 0.9 と１の間まで続けます。この 1/n*rndu(1,1)+(i-1)/n;という部分は、一様分布 U(0,1)を広げて一般的な一様乱数 U(a,b)にする際の手法と全く同じです。すなわち、

$$U(a,b)=(b-a)U(0,1)+a$$

というように、０と１の区間を(ｂ－ａ)倍して、下限のａから始まるように０からシフトさせる要領と全く同じです。この(ｂ－ａ)倍して伸ばす部分が 1/n 倍して細分化させることに代わっていることと、下限が(i-1)/n となっていて、ｉ＝1,2,3,...,n と増すにつれて、0,1/n,2/n,3/n,...,n/n すなわち１まで変化させているのです。

ｎ等分されたそれぞれの区間から１個だけサンプルを取るのが上のプログラムでしたが、それぞれ数個、ここではすべてｋ個ずつ取ってくる方法をプログラムしてみます。

プログラム

```
new; cls;
n=10;
k=3;

x=zeros(n,k);
i=1;
do while i<=n;
    x[i,.]=1/n*rndu(1,k)+(i-1)/n;
    i=i+1;
endo;
print x;

x=vec(x');
print x;
```

画面表示

```
    0.0068894946     0.059569832     0.098345182
     0.13709047      0.13088079      0.17491258
     0.29197052      0.25608542      0.29916175
     0.33980253      0.32363558      0.34259098
     0.46939632      0.42627080      0.42719392
     0.58478686      0.56925613      0.58265966
```

0.68928573	0.64672748	0.67821376
0.78130923	0.76614080	0.74615568
0.81099606	0.83782632	0.88094796
0.92967975	0.91676019	0.98474775

0.0068894946
0.059569832
0.098345182
0.13709047
0.13088079
0.17491258
0.29197052
0.25608542
0.29916175
0.33980253
0.32363558
0.34259098
0.46939632
0.42627080
0.42719392
0.58478686
0.56925613
0.58265966
0.68928573
0.64672748
0.67821376
0.78130923
0.76614080
0.74615568
0.81099606
0.83782632
0.88094796
0.92967975
0.91676019
0.98474775

ｎ等分されたそれぞれの区間からｋ個ずつとってきているのが画面表示からわかると思います。それを vec 命令で行列をベクトル化すれば１列になります。ただしその時、一旦転置した変数を vec 命令でベクトル化する必要があります。なぜなら、GAUSS の vec 命令は他の組込み関数と同じく列ごとにベクトル化していくので、列と行を一旦転置で入れ換えてから vec 命令でベクトル化すればきれいに並びます。ただし、ｋ個ずつうまく小さい区間から順に並びますが、その内部では順番はばらばらとなります。必要とあれば、こうしてベクトル化したものをさらにランダムに並び替えてやればよいことになります。

上ではすべての区間に等しいウエイトを与えてきましたが、それを変更してウエイトを区間ごとに変更することも可能です。一般的なプログラムはできませんが、後の章で density を張り合わせる応用をする際に、それについては説明することにしましょう。

戻って、１個ずつのサンプルをとる方法に話を絞り込みます。これを乱数として用いるのならば、第１の方法と同様に、その並びをランダム化する必要があります。順序のランダム化をした結果を用いて、標準正規分布の逆関数を用いて作成したのが次の結果です。

プログラム

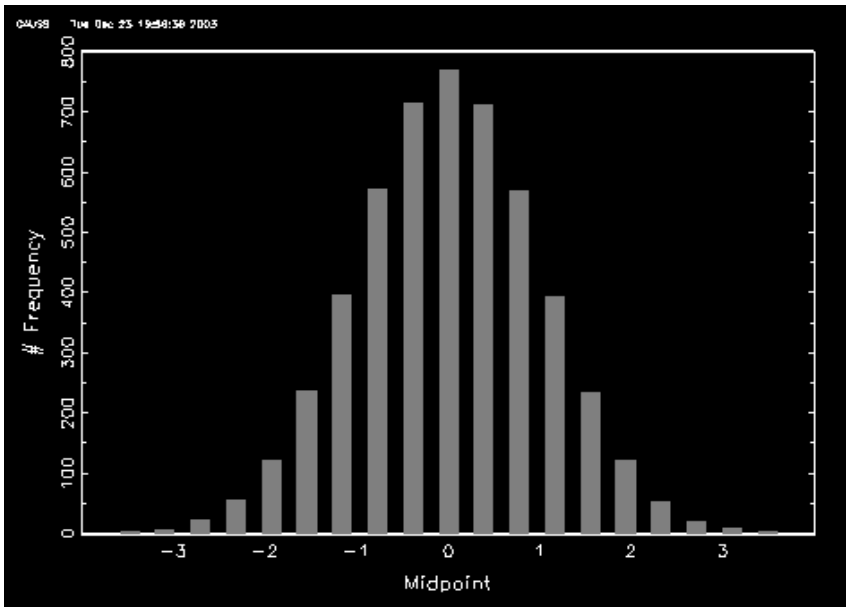
```
n=5000;
Xn=cdfni(U01(n));
print "mean=" meanc(Xn);
print "s.d.=" stdc(Xn);
library pgraph;
graphset;
call hist(Xn,19);

proc U01(n);
    local x,i,y,index;
    x=zeros(n,1);
    i=1;
    do while i<=n;
        x[i]=1/n*rndu(1,1)+(i-1)/n;
        i=i+1;
    endo;
    y=zeros(n,1);
    i=1;
    do while i<=n;
        index=ceil((n-(i-1))*rndu(1,1));
        y[i]=x[index];
        x=delif(x,x.==x[index]);
```

```
        i=i+1;
    endo;
    retp(y);
endp;
```

画面表示

```
mean=   1.4811593e-005
s.d.=         0.99991506
```

グラフ表示



きれいな標準正規分布となります。この方法は、第 1 の方法にｎを大きくしていけばほとんど同じになります。なぜならば、区間が限りなく狭くなり、その中点も区間から得られるランダムな値もほとんどかわりがなくなるからです。

一般に Stratified Sampling の方法は、一次元の系列に限って見れば、後の章で述べる準乱数のシークエンスを用いた乱数よりも、よいパフォーマンスを示します。なぜならば、準乱数が隙間を埋めるように生成され、区切りのよい数でなくてはきっちりと等間隔に散らばっていないのに対して、この等間隔のバージョンの Stratified Sampling の考え方は、どんなｎの区切りに対しても、常に同じ数ずつ（または１個ずつ）サンプルを取るために、それよりも静的な散らばりを見るときには、単独では最強の方法となります。特に、一様乱数から逆関数法を用いて裾野の広い分布に変換する際には、この方法が単独の方法としては最もパフォーマンスよく均等に広がりを見せてくれます。ただし、2 番目の方法とても最初から順番がランダムになっているわけではなくて、その並べ替えは例えば通常の組込み関数の一様乱数に依存しますから、乱数としての周期を考えるには、一次元の場合には問題はほぼありませんが、複数の系列の次元については、順番をランダムにする際の乱数

にその周期性は依存するものと思われます。また、第 1 の方法であるｎ等分した区間の中点を取っていく手法は、準乱数と同じように、ｎを読まれてしまうとパラメータの真の値とこの方法による値との差は常に既知となってしまい裁定の機会が生ずる原因にもなり得ますので、細心の注意が必要です。

多次元のケース

２系列以上の多次元の場合も、上の２つの方法を用いて容易に拡張することができます。すなわち、１）ｎ等分された区間の固定された中点をサンプルする方法を次元の個数であるｋ回分順番のみを変更して使用するケース　２）ｎ等分された区間からランダムにサンプルされた点を固定しておいて、その順番だけを変更してｋ回分使用するケース　３）ｎ等分された区間からランダムにサンプルする作業そのものを独立にｋ回分繰り返すケースの大きく分けて３つが考えられます。

１）中点をサンプルする方法のｋ次元への単純な拡張

プログラム

```
new; cls;
n=10;
k=3;
print U01(n,k);

proc U01(n,k);
    local x0,x,i,j,y,index;
    x0=zeros(n,1);
    i=1;
    do while i<=n;
        x0[i]=(i-0.5)/n;
        i=i+1;
    endo;
    y=zeros(n,k);
    j=1;
    do while j<=k;
        x=x0;
        i=1;
        do while i<=n;
            index=ceil((n-(i-1))*rndu(1,1));
            y[i,j]=x[index];
```

```
            x=delif(x,x.==x[index]);
            i=i+1;
        endo;
        j=j+1;
    endo;
    retp(y);
endp;
```

画面表示

0.65000000	0.45000000	0.85000000
0.25000000	0.95000000	0.050000000
0.85000000	0.35000000	0.75000000
0.45000000	0.75000000	0.45000000
0.15000000	0.85000000	0.65000000
0.75000000	0.25000000	0.15000000
0.95000000	0.55000000	0.55000000
0.050000000	0.15000000	0.95000000
0.55000000	0.050000000	0.35000000
0.35000000	0.65000000	0.25000000

２）ｎ等分された区間からランダムにサンプルした点を固定してｋ次元に拡張するケース

プログラム

```
new; cls;
n=10;
k=3;
print U01(n,k);

proc U01(n,k);
    local x0,x,i,j,y,index;
    x0=zeros(n,1);
    i=1;
    do while i<=n;
        x0[i]=1/n*rndu(1,1)+(i-1)/n;
        i=i+1;
    endo;
    y=zeros(n,k);
    j=1;
```

```
    do while j<=k;
        x=x0;
        i=1;
        do while i<=n;
            index=ceil((n-(i-1))*rndu(1,1));
            y[i,j]=x[index];
            x=delif(x,x.==x[index]);
            i=i+1;
        endo;
        j=j+1;
    endo;
    retp(y);
endp;
```

画面表示

0.41140779	0.56976507	0.17532145
0.85166925	0.98980755	0.98980755
0.39247472	0.62761932	0.62761932
0.56976507	0.41140779	0.25592927
0.077143841	0.85166925	0.077143841
0.77085873	0.39247472	0.56976507
0.25592927	0.077143841	0.85166925
0.17532145	0.77085873	0.77085873
0.62761932	0.25592927	0.41140779
0.98980755	0.17532145	0.39247472

プログラム的には、procedure 内前半の x0 を作り出すところを、固定区間の中点の計算から、ｎ等分された区間からランダムにサンプルするものに取り換えるだけです。その結果、ランダムにさんぷるされた第１番目の系列の内容と同じものが、それ以降の系列にも順番を変えて出てきます。

３）ｎ等分された区間からランダムにサンプルする方法を次元数ｋ回分繰り返すケース

プログラム

```
new; cls;
n=10;
k=3;
print U01(n,k);
```

```
proc U01(n,k);
    local x,i,j,y,yy,index;
    yy=zeros(n,k);
    j=1;
    do while j<=k;
        x=zeros(n,1);
        i=1;
        do while i<=n;
            x[i]=1/n*rndu(1,1)+(i-1)/n;
            i=i+1;
        endo;
        y=zeros(n,1);
        i=1;
        do while i<=n;
            index=ceil((n-(i-1))*rndu(1,1));
            y[i]=x[index];
            x=delif(x,x.==x[index]);
            i=i+1;
        endo;
        yy[.,j]=y;
        j=j+1;
    endo;
    retp(yy);
endp;
```

画面表示

0.34010118	0.69145551	0.61238525
0.82650010	0.70209198	0.0080961824
0.66165484	0.25624973	0.58540502
0.44267586	0.30537349	0.12168187
0.55922965	0.99887256	0.49153516
0.25802491	0.18667051	0.23242963
0.19037553	0.87336710	0.77461474
0.026750494	0.53854912	0.36105091
0.74815752	0.079148564	0.82029501
0.90626402	0.43587985	0.98015671

プログラムは、ｎ等分された区間からランダムにサンプルする１系列のケースを１番外側のループで次元数ｋ回まわして、それぞれ独立に作成されたｙをその都度ｙｙのｊ列目に格納していくことを１列目からｋ列目まで繰り返したものです。このケースは、すべての系列で、ｎ等分された区間からランダムにサンプルが１つずつ取り出されています。これは、ちょうど３次元であればそれぞれｎ等分された一辺が 1/n の長さをもつ立方体のなかから１つだけサンプルするのに結果的に同じことになります。３次元よりも大きい場合には、想像することは困難を極めますが、ｎ等分された一辺が 1/n の長さの超次元のそれぞれのセルから１つずつサンプルすることに結果的に同じになります。なお、この方法はアルゴリズム的に改良の余地は大いにありますが、考え方としては上のプログラムが一番シンプルであります。