## 5.19 モンテカルロシミュレーション　採択棄却法

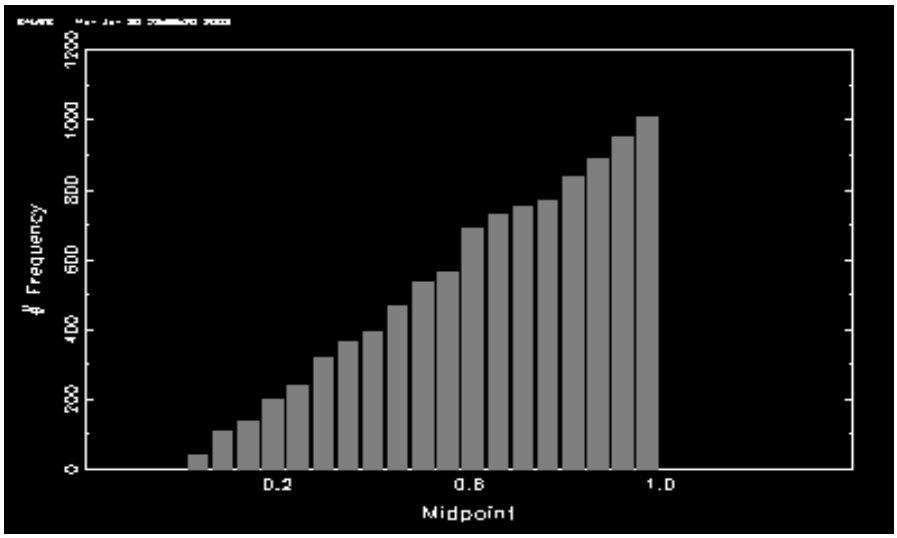
ここでは、まず逆関数法の説明で用いた三角分布をこの acceptance rejection method を用いて乱数発生させてみよう。この方法は、pdf さえわかればどんな分布の乱数であっても簡便に発生させることができる。もちろん、逆関数を知る必要は全くない。棄却される部分の発生効率の面をどうするかの問題はあるものの、計算機が十分に発展した今日においては非常に強力かつ簡便な方法となっている。

プログラム

```
new; cls;
n=10000;
fn f(x)=2*x;
fn g(x)=1;
a=2;

X=zeros(n,1);
i=1;
do while i<=n;
    Y=rndu(1,1);
    U=rndu(1,1);
    if U<=f(Y)/(a*g(Y));
        X[i]=Y;
        i=i+1;
    endif;
endo;

library pgraph;
graphset;
call hist(X,19);
```

上のプログラムでは、a 倍（a は１以上）された g(Y)が pdf の全域にわたって求める乱数の pdf である f(Y)よりも理論上等しいか大きいものとする。上のケースでは、この三角分布は０と１の間で最高の高さが２であるから、それに対して２倍された０と１の間に対して常に１という特殊な pdf である g(x)は常に等しいか大きい関係が成り立つことは容易にわかるであろう。すなわち、計算すると U<=f(Y)/(a*g(Y))の部分は U<=1-Y でも同じことである。どのような分布に対しても基本的にこのアルゴリズムが成り立つように、ここではあえて計算する前の f(x)と a それに g(x)を用いて示している。

グラフ表示




基本的なアルゴリズム

１）g(x)から乱数 Y を発生させる

２）０と１の間の一様乱数 U を発生させる

３）$U \le \frac{f(Y)}{ag(Y)}$であれば採択して、X=Y とする。そうでなければ、棄却して元に戻る。

４）１から３を繰り返す。

５）採択された X がｎ個になるまで繰り返したものを並べて乱数列とする。

したがって、棄却される部分の面積が大きくなってしまって効率的でなくなっても構わないのならば、g(Y)をある区間の間の一様乱数として、a×g(Y)の部分のまとめた大きさが f(Y)の最高値と一致しているかそれよりも大きければよいことになる。すなわち、ag(Y)は f(Y)をすべての Y について（実際にはその区間の Y についてだが）カバーすることになる。

**一般的な pdf 関数からの乱数の発生（GED 分布の例）**

こうした考え方を一般化して、求めたい分布の pdf が f(x)である時のあらゆる乱数を作り出す procedure を作成すると次のようになる。なお、ここでは g(x)を up と low の間の一様乱数として、max を a×g(Y)の大きさとして計算している。

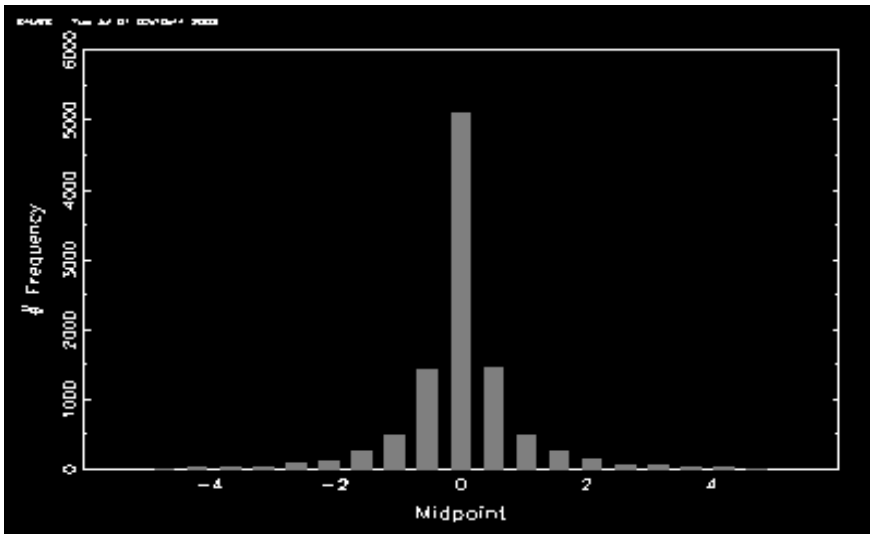
プログラム

```
new; cls;
nu=0.5;
n=10000;
fn f(x)=pdfged(x,nu);
call rejmethod(&f,n);
```

```
proc rejmethod(&f,n);
    local low,up,s,fs,max,X,i,Y,U,f:proc;
    low=-5; up=5;
/* get max f(x) */
    s=seqa(low,0.01,(up-low)/0.01+1);
    fs=zeros((up-low)/0.01+1,1);
    i=1;
    do while i<=rows(fs);
        fs[i]=f(s[i]);
        i=i+1;
    endo;
    max=maxc(fs);
/* acceptance rejection method */
    X=zeros(n,1);
    i=1;
    do while i<=n;
        Y=(up-low)*rndu(1,1)+low;
        U=rndu(1,1);
        if U<=f(Y)/max; /* max=a*g(x) */
            X[i]=Y;
            i=i+1;
        endif;
    endo;
    library pgraph;
    graphset;
    call hist(X,19);
    retp(X);
endp;

proc pdfged(x,nu);
    local b;
    if nu<=0;
        errorlog "ERROR: Shape parameter nu is not positive" ;
        end;
    endif ;
```

```
    b=((2^(-2/nu))*(gamma(1/nu))/(gamma(3/nu)))^(0.5);
    retp(   nu*exp(-0.5*(abs(x/b)^(nu)))/(b*gamma(1/nu)*(2^(1+1/nu)))   );
endp;
```

グラフ表示




上の乱数の分布はν＝0.5 のケースである。νが２のケースは標準正規乱数と等しくなり、νがそれより小さい場合には尖る。反対にνが２より大きい場合には大きくなるほど台形のような分布になる。このことは先の GED の章で述べた通りである。ここでは、採択棄却法でもって GED の pdf が与えられている場合に、その情報のみから乱数を生成したものである。ここでは、f(x)に自作した gedpdf を定義しているが、pdf がわかっているのであればどのような分布の pdf でも構わない。

ただし、デフォルト設定では up と low がそれぞれ５と－５という有限の区間のみからのみ乱数が発生することができるという設定にしている。偏りがあるケースや、もっと分布が広がるケースには適宜 up と low の値を変更する必要がある。実際、－∞と∞の間の広がりをもつケースがほとんどなので、この GED のケースも含めて－５と５の間に分布を限定するのは、理論上は正確とは言えない。しかしながら、実際のデータを想定してシミュレーションする場合には、ある程度の範囲で下限と上限を決めてしまうのも想定上おかしなことではないであろう。

## Kernel Density Monte Carlo

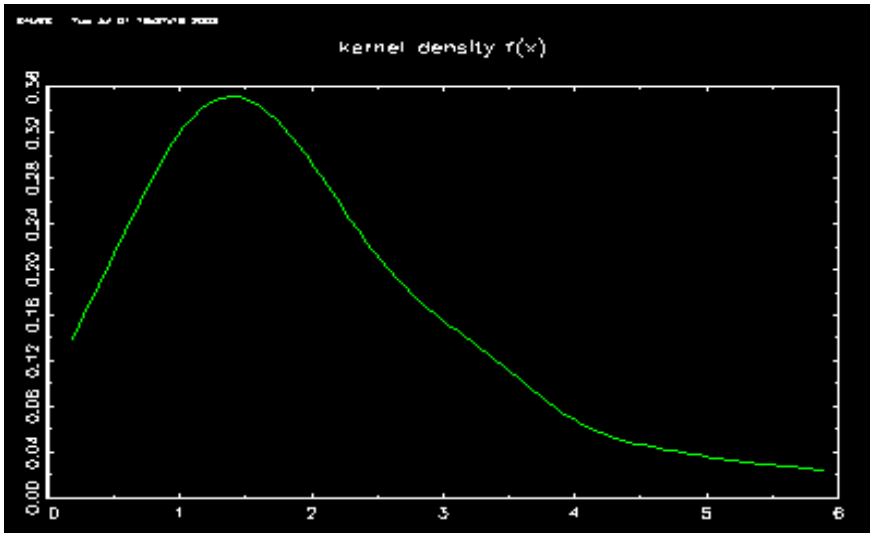
プログラム

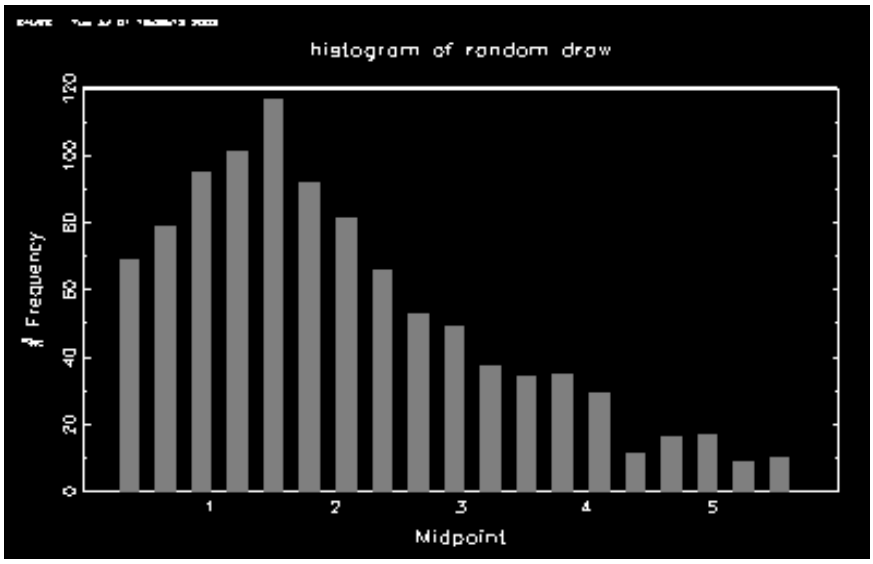
```
new; cls;
data=rndgam(100,1,2);
n=1000;
call rndkern(data,n);

proc rndkern(data,n);
    local m,nn,low,up,maxf,X,i,Y,U;
    m=plotdens(data);
    nn=rows(m);
    low=m[1,1]; up=m[nn,1];
    maxf=maxc(m[.,2]);
    library pgraph;
    graphset;
    pqgwin auto;
    title("kernel density f(x)");
    xy(m[.,1],m[.,2]);
    print "maxf=" maxf;
    print/rz "[" low "," up "]";
    print;
    print "Proceeding...(wait for a long time)";
/* acceptance rejection method */
    X=zeros(n,1);
    i=1;
    do while i<=n;
        Y=(up-low)*rndu(1,1)+low;
        U=rndu(1,1);
        if U<=f(Y)/maxf;
            X[i]=Y;
                call sleep(0.1);
            print/lz i "out of " n;
            i=i+1;
        endif;
    endo;
    title("histogram of random draw");
```

```
    call hist(X,19);
    retp(X);
endp;

proc f(x);
    local m,fx,index;
    m=plotdens(data);
    fx=m[.,2];
    index=minindc(abs(m[.,1]-x));
    retp(fx[index]);
endp;

proc plotdens(x);
    local npoints,n,h,points,kern,i,j;
    npoints=1001;
    n=rows(x);
    x=sortc(x,1);
    h=1.059*stdc(x)*n^(-0.2)*1;
    points=seqa(x[1],(x[n]-x[1])/(npoints-1),npoints);
    i=1; kern=zeros(npoints,1);
    do while i<=npoints;
        j=1;
        do while j<=rows(h);
            kern[i,j]=gkernel(points[i],x,h[j]);
            j=j+1;
        endo;
        i=i+1;
    endo;
    retp(points~kern);
endp;

proc gkernel(x0,x,h);
    local u,kv;
    u=(x0-x)/h;
    kv=(1/sqrt(2*pi))*exp(-0.5*u^2);
    retp( sumc(kv/(rows(x)*h)) );
```

endp;

グラフ表示




これは、先に実際のデータの Kernel Density を求めておいて、それを f(x)として、そこから採択棄却法によって乱数を発生させている。Kernel Density Monte Carlo とでも呼べるものであろう。プログラムでは、データをΓ分布にしたがう乱数を用いているが、この data の部分は１列の実際のデータがきても計算できる general procedure になっている。

**Density データからの乱数発生（GH 分布の例）**

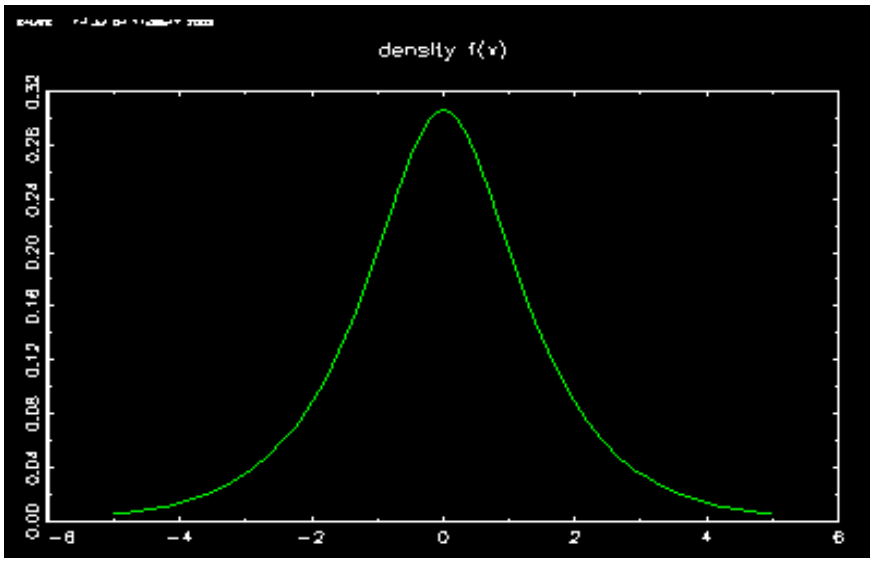
上の Kernel Density Monte Carlo の方法とほとんど同じ方法で、今度は分布の density データがわかっている際のプログラムを行なってみよう。1 つ前の pdf から乱数を発生させる方法も十分に汎用性はあるのだが、複雑にグローバルに変数が通っている場合にはそれは利用できない。そこで、先に density データを作成しておいてから、それにもとづいてその都度の高さの近似値を計算する f(x,dens)を用いて同様の計算をする。以下では GH 分布を考える。

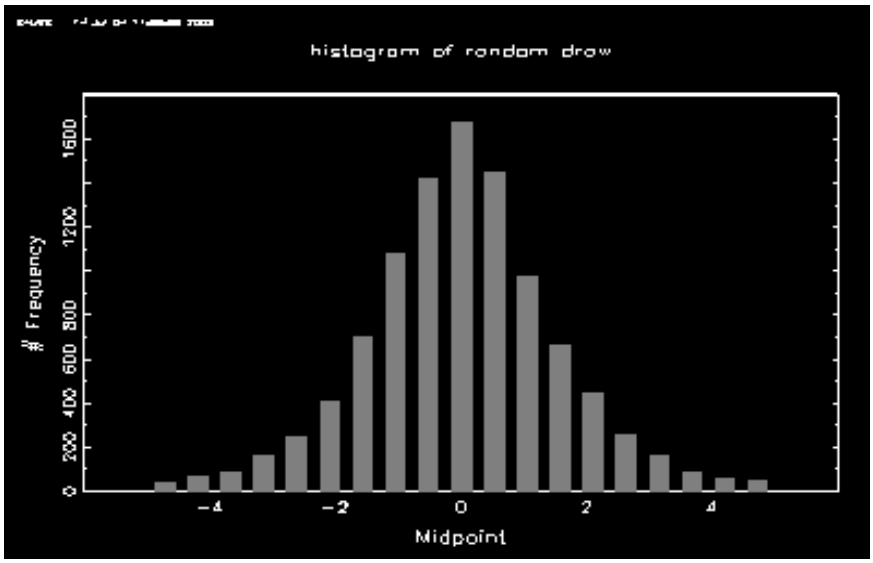
プログラム

```
new; cls;
lambda=1; alpha=1; beta=0; delta=1; mu=0;
x=seqa(-5,0.1,101);   /* x=seqa(-5,0.01,1001); is better for normal version of GAUSS. */
dens=pdfgh(x,lambda,alpha,beta,delta,mu);
n=10000;
call rndfun(dens,n);

proc rndfun(dens,n);
    local m,nn,low,up,maxf,X,i,Y,U;
    m=dens;
    nn=rows(m);
    low=m[1,1]; up=m[nn,1];
    maxf=maxc(m[.,2]);
    library pgraph;
    graphset;
    pqgwin auto;
    title("density f(x)");
    xy(m[.,1],m[.,2]);
    print "maxf=" maxf;
    print/rz "[" low "," up "]";
    print;
    print "Proceeding...(wait for a long time)";
/* acceptance rejection method */
    X=zeros(n,1);
    i=1;
    do while i<=n;
        Y=(up-low)*rndu(1,1)+low;
        U=rndu(1,1);
```

```
        if U<=f(Y,dens)/maxf;
            X[i]=Y;
            call sleep(0.1);
            print/lz i "out of " n;
            i=i+1;
        endif;
    endo;
    title("histogram of random draw");
    call hist(X,19);
    retp(X);
endp;

proc f(x,dens);
    local m,fx,index;
    m=dens;
    fx=m[.,2];
    index=minindc(abs(m[.,1]-x));
    retp(fx[index]);
endp;

proc pdfgh(x,lambda,alpha,beta,delta,mu);
    local yl,K1,K2,gam;
    yl=29|
          0;
    _intord=40;
    K1=0.5*intquad1(&f1,yl);
    K2=0.5*intquad1(&f2,yl);
    gam=sqrt(alpha^2-beta^2);
    retp( x~((gam./delta)^lambda./(sqrt(2*pi)*K2))
        .*(K1./(sqrt(delta^2+(x-mu)^2)./alpha)^(0.5-lambda)).*exp(beta.*(x-mu)));
endp;

fn f1(y)=y^((lambda-1)-0.5).*exp(-0.5.*alpha.*sqrt(delta^2+(x-mu)^2).*(y+y^(-1)));
fn f2(y)=y^(lambda-1).*exp(-0.5.*delta.*sqrt(alpha^2-beta^2).*(y+y^(-1)));
```

グラフ表示




なお、GH の pdf は、Bibby-Sorensen, Hyperbolic Processes in Finance によれば、

$$f(x;\lambda,\alpha,\beta,\delta,\mu)=\frac{(\gamma/\delta)^{\lambda}}{\sqrt{2\pi}K_{\lambda}(\delta\gamma)}\cdot\frac{K_{\lambda-\frac{1}{2}}(\alpha\sqrt{\delta^2+(x-\mu)^2})}{(\sqrt{\delta^2+(x-\mu)^2}/\alpha)^{\frac{1}{2}-\lambda}}\cdot e^{\beta(x-\mu)}$$

$$where \quad \gamma^2=\alpha^2-\beta^2$$

また、$K_{\lambda}$は Modified Bessel 3rd の K であって、次のような積分表現ができる。

$$K_{\lambda}(x)=\frac{1}{2}\int_0^{\infty}u^{\lambda-1}e^{-\frac{1}{2}x(u+u^{-1})}du, \quad x>0$$

となる。これを数値積分でもとめたものが、プログラム中の $K_1$ および $K_2$ になっている。

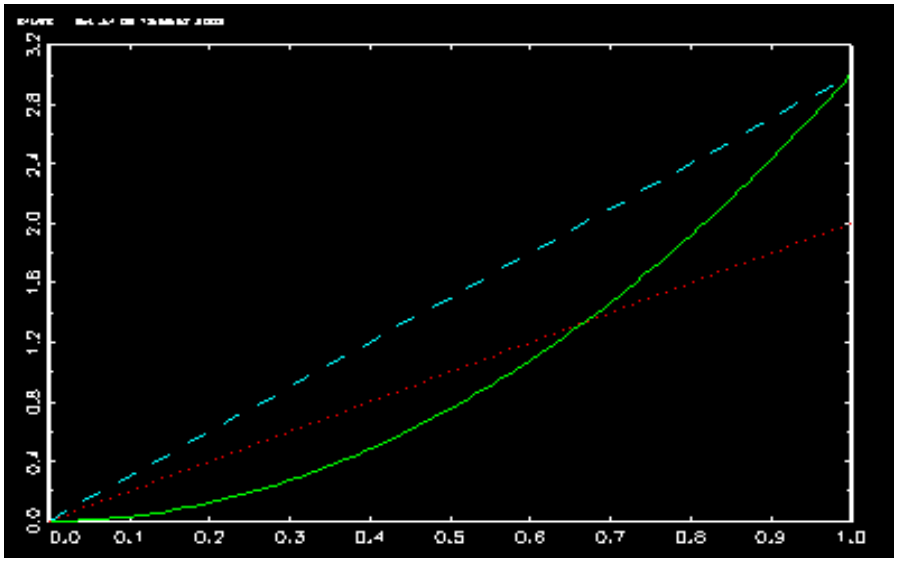
また、積分はすでにｘの中味を入れたものの積分を０から 29 まで最高精度の 40 に設定して上の公式の u のところをｙに置き換えて行なっている（これは標準正規分布の pdf とほぼ一致するように調整)。pdf の方のｘの値(BesselK の中味ではなくて)はプログラムでグローバルに通してある。プログラムでは、pdfgh(x,lambda,alpha,beta,delta,mu)と f1(y)および f2(y)がグローバルに通ったｘ列ベクトルに対する pdf をｘのインデックスつきでもとめるプログラムである（１列目はｘの値、２列目は GH の pdf の値)。これを dens としておいて、それをインプットｘが与えられたときの pdf の近似値（すなわち高さ）を求める関数 f(x,dens)を経由して、最終的に rndfun(dens,n)でこの density に覆いかぶさる一様分布の高さを求めた上、それをもとに単純な棄却法でこの density に対する乱数を発生させている。これら GED および Generalized Hyperbolic それに Kernel Density から乱数を発生させるプログラムで、どんな一般的な分布であろうと分布がそもそも数式で定義されなくとも乱数を発生できるようになった。

ただし、上でやってきた方法は、棄却法のアルゴリズムのａg(x)を対象とする分布の最高値を上底として、あらかじめ定められた最低値から最高値の範囲にたいする、そのような長方形で囲まれた部分の棄却法を行なっているものである。したがって、対象とする分布が一様乱数のように長方形に近ければ棄却される頻度は減少して効率のよいアルゴリズムになるが、反対に一部が狭い範囲で非常に尖っている分布の場合には棄却される部分が大半であって、このアルゴリズムは非常に効率の悪く（時間のかかる）なる。また、これまで述べたように、実際の分布は、実際には例えば－５から５の間でほとんどのケースが出てくるのであるが、数学的には－∞から∞までの広がりをみせるので、厳密にはこの方法は最初から範囲外の値をカットしている乱数生成法と言える。なお、この後者の範囲外の値の問題はなるべく範囲を広くとることによって解決ができるし、上で行った Kernel Density Monte Carlo はそもそも最低と最高の範囲が決まっているので問題は生じない。むしろ、問題があるのは前者の棄却される部分が多くなってしまって時間が途方もなくかかることであろう。以前のように計算機のスピードが遅かったり計算機を使用できる機会が順番に割り当てられていた時代にはこの問題は深刻であったが、現在ではないに等しい。しかしながら、もし効率化を考えるならば、対象となる分布に覆いかぶさるａg(x)の部分を長方形ではなくて、その分布にできるだけ近い、しかも全域にわたって上回っているような、分布を定数ａでスケールアップさせたものを考えることができる。

**採択効率の改善**

　これまでの乱数発生では、g(x)の形がｘの値に関わらずフラットである特殊なケースを見た。すなわち、ａg(x)のフラットな値がf(x)の一番高いところに相当する場合のアルゴリズムであった。ここでは、冒頭に述べた棄却法のアルゴリズムにおいてg(x)からの乱数の発生がフラットでない一般的なケースについて、最もわかりやすい例で見ていこう。

```
new; cls;
x=seqa(0,0.01,101);
fn f(x)=3*x^2;
fn g(x)=2*x;
a=1.5;

library pgraph;
graphset;
xtics(0,1,0.1,0);
xy(x,f(x)~a*g(x)~g(x));
```




今、実線の２次曲線とｘ軸とで囲まれる部分の面積が１であるようなpdfを考える。つまり

$$f(x)=3x^2 \quad 0\leqq x\leqq 1$$
$$=0 \quad \text{Otherwise}$$

を考える。$3x^2$の積分は$x^3$であるので（実際のところ$x^3$の微分は$3x^2$である）0から１までの定積分は１になる。前の章で逆関数法で乱数生成を行なった三角分布

$$g(x)=2x \quad 0\leqq x\leqq 1$$
$$=0 \quad \text{Otherwise}$$

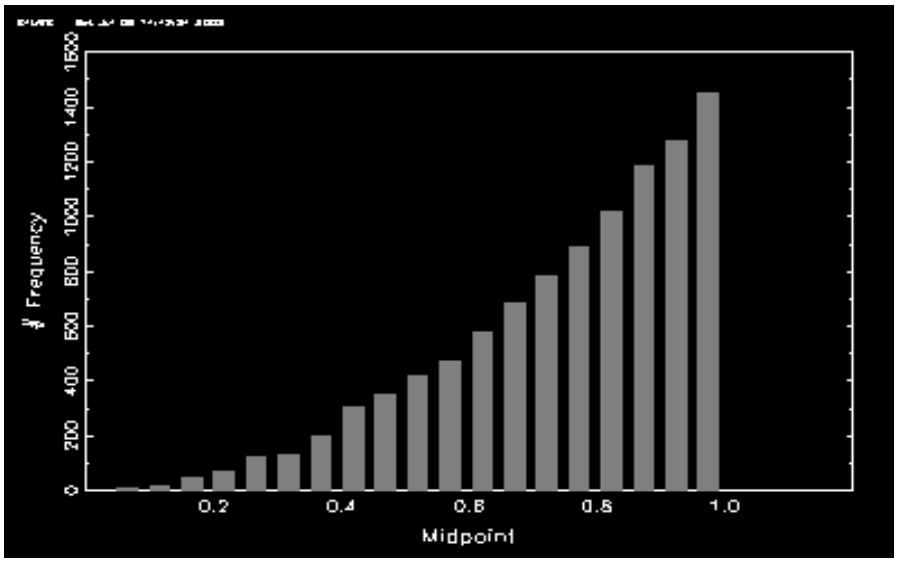
は細かい点線とｘ軸とで囲まれる部分の面積が１であるようなpdfである。これを今g(x)

としよう。上で表現した２次形式の分布 f(x)をすべての範囲のｘに対して覆いかぶさるようなａg(x)は上のような g(x)に対して a=1.5 であることは容易にわかるであろう。つまり、g(x)とｘ軸で囲まれた面積は、pdf と cdf の定義から、常に１でなければならない。これを定数ａでスケールアップさせたａg(x)が f(x)をカバーするようになる。これはグラフの大まかな破線に相当している。f(x)≦ａg(x)が成り立っていることは、この破線が対象となる分布 f(x)の実線を常に上回るか同値であることと同じなのである。以下では、このことをもとに、まず三角分布 g(x)を逆関数法で求めた上で、冒頭の採択棄却法のアルゴリズムでもって、より棄却される割合が小さくなる乱数発生を行なってみる。

プログラム

```
new; cls;
n=10000;
fn f(x)=3*x^2;
fn g(x)=2*x;
a=1.5;

X=zeros(n,1);
i=1;
do while i<=n;
    Y=rndtri(1,1);
    U=rndu(1,1);
    if U<=f(Y)/(a*g(Y));
        X[i]=Y;
        i=i+1;
    endif;
endo;

library pgraph;
graphset;
call hist(X,19);

proc rndtri(r,c);
    local x;
    x=rndu(r,c);
    retp( sqrt(x) );
endp;
```

グラフ表示




上のプログラムとグラフは、f(x)＝3ｘ$^2$を、また g(x)＝ 2ｘそれに a=1.5 として、Ｙを発生させるのに自作組込み関数 rndtri（この三角関数の逆関数は前章でやったとおりである）を用いている。その他の部分は、冒頭の三角分布の棄却法による乱数発生とほぼ同じである。ただし、そこでは g(x)＝1 であったため、どのような乱数Ｙに対しても g(Y)はフラットで１であった。それを a=2 でスケールアップさせていた。ここでは、乱数Ｙ自体が三角分布から発生させたものであり、かつ、g(Y)はフラットではなくて、g(Y)=2Y であることに注意してもらいたい。

以下では、ここでの三角分布を覆いかぶせるケースと従来の高さ３の長方形を上から覆いかぶせたａg(x)を考えるケースのそれぞれの棄却率を計算してみたい。

プログラム

```
new; cls;
n=10000;
fn f(x)=3*x^2;
fn g(x)=2*x;
a=1.5;

X=zeros(n,1);
i=1; times=0;
do while i<=n;
    Y=rndtri(1,1);
    U=rndu(1,1);
    if U<=f(Y)/(a*g(Y));
        X[i]=Y;
```

```
        i=i+1;
    endif;
    times=times+1;
endo;
print "rejection ratio=" (times-n)/times;

library pgraph;
graphset;
call hist(X,19);

proc rndtri(r,c);
    local x;
    x=rndu(r,c);
    retp( sqrt(x) );
endp;
```

画面表示

```
rejection ratio=          0.33342221
```

プログラム

```
new; cls;
n=10000;
fn f(x)=3*x^2;
fn g(x)=1;
a=3;

X=zeros(n,1);
i=1; times=0;
do while i<=n;
    Y=rndu(1,1);
    U=rndu(1,1);
    if U<=f(Y)/(a*g(Y));
        X[i]=Y;
        i=i+1;
    endif;
    times=times+1;
endo;
```

```
print "rejection ratio=" (times-n)/times;
```

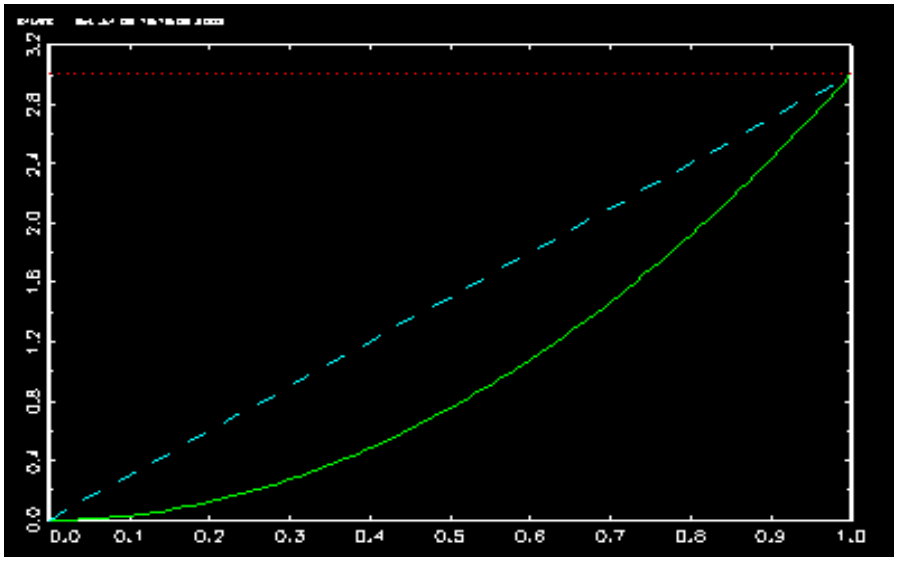
```
library pgraph;
graphset;
call hist(X,19);
```

画面表示

```
rejection ratio=          0.66617706
```

これらの結果は、次のようなグラフの面積と関係がある。

```
new; cls;
x=seqa(0,0.01,101);
fn f(x)=3*x^2;
fn g(x)=2*x;
a=1.5;

library pgraph;
graphset;
xtics(0,1,0.1,0);
call xy(x,f(x)~a*g(x)~3*ones(101,1));
```




すなわち、第 1 番目の棄却率の計算で約 0.333 となっているのは、実線とｘ軸の間にくる乱数の頻度を作り出すのに、斜めの破線とｘ軸の間にくる三角形の部分の一様な乱数発生がトータルで必要で、この破線と実線の間の弧の面積の部分が棄却されて使われないことになる。正確に計算すると、三角形の面積は 1.5 で実線の下の面積は当然のことながら１で

あるから、1.5 のうち 1.5－１＝0.5 が捨てられることになり、理論上の棄却率は 0.5/1.5＝1/3 ということになる。シミュレーションの都度に棄却率は若干変化するが、この 0.333 という値はほぼこの理論値に近いものになっている。また、従来の長方形を覆いかぶせるケースでは、上のグラフでは点線の下の面積３に対して、この分布の実線部から下の面積が１であることから、トータル３の面積のうち、3－１＝２が捨てられることになり、理論上の棄却率は 2/3 となる。実際、シミュレーションによる棄却率は 0.666 前後となり、理論値に近いものになっている。こうした関係からもわかるように、冒頭の採択棄却法のアルゴリズムは､あらかじめその pdf 関数を覆いかぶすような面積に上下左右一様に広がる乱数分布の pdf 関数から下の部分だけを採択したものと考えることができる。その効率性とは、そこで棄却される面積が全体で考える面積に占める割合が小さければ小さいほど効率的であると考えることができる。すなわち、計算機上では効率的であればあるほど棄却される割合が小さくスピードが速いということを意味する。上の例では、従来のフラットなケースを考える棄却率が 2/3 に対して､この三角関数を覆いかぶせるケースでは棄却率は 1/3 にまで減少している。さらに弧を２分割３分割としていくにしたがって棄却率は下がり効率的となる。ただし、そのような分割するプログラムが実際上スピードの面や手軽さから有効であるのかどうかの問題は、棄却率低下による実際のスピードのアップとの兼ね合いで考えなければならない問題である。計算機が十分に発達した今日では、効率性を無視したプログラミングも有効になっており、また今後ますますシンプルな計算方法の有効性は増すであろう。

**効率性と極端な例**

　最後に、再び棄却法を整理しておいた上で、２つの賢くない極端な例を見ておこう。この棄却法が何であるかを実感できるであろう。まず、冒頭で述べたアルゴリズムは、これまでもプログラムしてきたように一番非効率な長方形を覆いかぶせるケースでは次のように書きかえることができる。

基本的なアルゴリズム

１）下限と上限が分布の定義域 domain に相当する一様乱数 Y を発生させる。

２）０と１の間の一様乱数 U を発生させる。

３）分布の変域 range の上限を maxh とする。

４）$U \le \dfrac{f(Y)}{\max h}$ であれば採択して、X=Y とする。そうでなければ、棄却して元に戻る。

５）１か４を繰り返す。

採択された X がｎ個になるまで繰り返したものを並べて乱数列とする。

このことは、グラフ上の面積を問題にするとき次のようにも解釈できる。

基本的なアルゴリズム

１）分布の定義域 domain および変域 range の２次元平面に一様に分布する乱数を考える。

２）ｘ≦f(x)にある部分のｘだけを採択する。そうでない部分は捨てる。

すなわち、domain の幅と range の高さに広がった長方形の２次元空間に一様に分布する乱数のうち、pdf の下の部分にくるものだけを採択して、あとは捨ててしまうのである。採択されたｘは pdf の高さ分だけあるはずで、これを並べればその pdf の乱数となる。pdf の上の部分の面積を全体の長方形の面積で割ったものが棄却率にほかならない。この考え方は、遠くはフォンノイマンまでさかのぼることができる。このことは、紙と鉛筆と一様乱数を作成する何か装置か道具さえあれば、たとえコンピュータがなくても、理論上はどんな pdf の乱数でも作成できることを意味している。

　では、完全に効率的なケースのプログラムを考えてみよう。また、再び三角分布で棄却率０の極端な場合を考えてみる。実は、これが賢くないことはすぐわかるはずである。

プログラム

```
new; cls;
n=10000;
fn f(x)=2*x;
fn g(x)=2*x;
a=1;


X=zeros(n,1);
i=1; times=0;
do while i<=n;
    Y=rndtri(1,1);
    U=rndu(1,1);
    if U<=f(Y)/(a*g(Y));
        X[i]=Y;
        i=i+1;
    endif;
    times=times+1;
endo;
print "rejection ratio=" (times-n)/times;


library pgraph;
```

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


proc rndtri(r,c);
    local x;
    x=rndu(r,c);
    retp( sqrt(x) );
endp;
```

画面表示

```
rejection ratio=        0.00000000
```

乱数を作成する対象となる pdf の f(x)もそれをカバーする g(x)もともに２ｘとなっている。したがって、当然のことながらａ＝１である。その特殊な完全に効率的なケースでは f(x)とａg(x)の比が１となって、上のプログラム中の **if U<=f(Y)/(a*g(Y));**の部分は常に真になり常にＹがこの場合の乱数として採択される。というよりも、正確には、この if クローズ自体が不要であって、Ｙはそもそも三角分布から発生された乱数そのものである。最初から三角分布から発生させた乱数を計算すればよかったわけである。Ｕの一様乱数生成は無駄になっている。当然のことながら、どんな乱数の出具合であろうとも棄却率は常に０となるはずである。実際に０になっている。つまり、完全に効率的な g(x)を考えることができるとき、この棄却法のアルゴリズム自体が不必要なのである。言い方をかえれば、採択棄却法なるものは、効率が悪いプログラムを最初から目指しているアルゴリズムなのである。その点をくれぐれも勘違いすることのないようにしてほしい。

　次の例は、もう１つの極端な例で、離散分布のケースである。一番効率が悪いケースのプログラムを考えるが、実はこの棄却法のアルゴリズム自体が、２重に計算をしているのであって、不必要であることがやがてわかるであろう。今、いんちきサイコロの pmf が次のようであるとする。

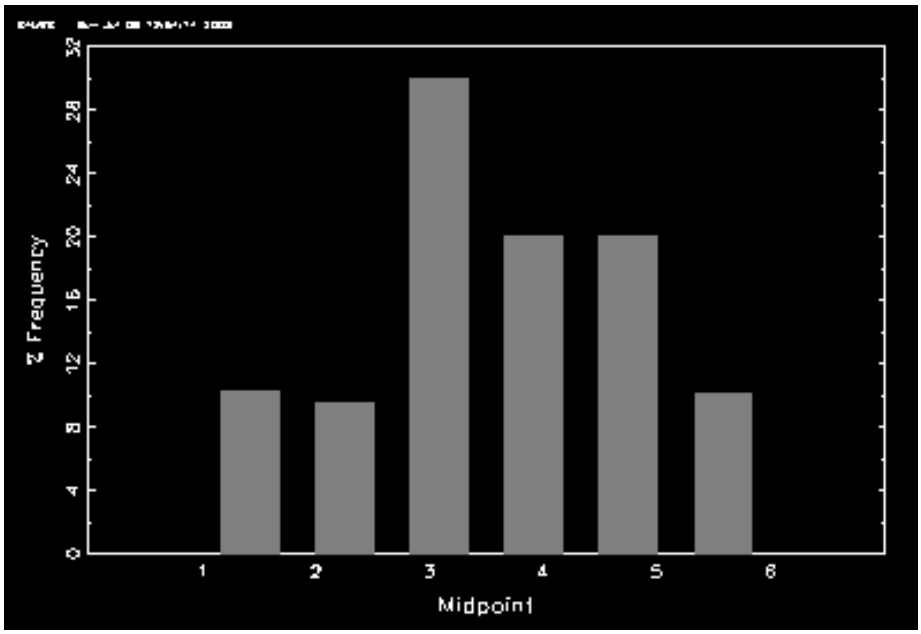
| | | |
|---|---|---|
| | 1 | ｐ1＝0.1 |
| | 2 | ｐ2＝0.1 |
| Ｘ＝ | 3 | ｐ3＝0.3 |
| | 4 | ｐ4＝0.2 |
| | 5 | ｐ5＝0.2 |
| | 6 | ｐ6＝0.1 |

プログラム

```
new; cls;
p={0.1,0.1,0.3,0.2,0.2,0.1};
```

```
maxh=maxc(p);
n=10000;

proc f(x,p);
    local pp;
    if x==1;
        pp=p[1];
    elseif x==2;
        pp=p[2];
    elseif x==3;
        pp=p[3];
    elseif x==4;
        pp=p[4];
    elseif x==5;
        pp=p[5];
    else;
        pp=p[6];
    endif;
    retp(pp);
endp;

X=zeros(n,1);
i=1; times=0;
do while i<=n;
    Y=ceil(6*rndu(1,1));
    U=rndu(1,1);
    if U<=f(Y,p)/maxh;
        X[i]=Y;
        i=i+1;
    endif;
    times=times+1;
endo;
print "rejection ratio=" (times-n)/times;

library pgraph;
graphset;
```

call histp(X,6);
画面表示
rejection ratio=　　0.44595268
グラフ表示




上のグラフは、histp で%表示のヒストグラムを作成したものである。最初に設定されたｐの確率とほぼ同じ出具合になっていることが確認できる。maxh の高さは p の中で一番大きいもの、そしてＹは{1,2,3,4,5,6}の中から一様に選ばれてくる整数値であるから切り上げの組込み関数 ceil を用いて **ceil(6*rndu(1,1))**から生成されることは容易にわかるであろう。このＹの出具合を冒頭のｐが与えられているときの関数 f(x,p)のｘのところに入れて採択棄却法を行なっている。これは、どのようなランダムな出具合の離散分布にもプログラム化することが容易である。しかしながら、よく考えてほしい。本来の離散分布の確率ｐの意味を考えれば、ｎ回の一様乱数の試行のうちｐのおのおのの確率で比例配分してやれば上のような pmf の分布そのものになるはずである。この場合も、一様乱数を２重に作って余計なことをしていることになる。無論、上のプログラムは練習問題としては正解である。しかしながら、実際的とは言えない。