5.17 モンテカルロシミュレーション MCMC　　ver.0.1

ここでは、当初開発された離散ジョイント分布による Gibbs Sampling から始めて、連続分布によるもの、そして、Metropolis-Hastings の方法、その Bivariate のケース、そして Metropolis-within-Gibbs の方法へと発展させてプログラムをしていきます。

Gibbs Sampling

離散でも連続でもどちらでもかまいませんが、ある分布 D のもとに、ｎ個のパラメータからなるθをシミュレーションすることにしましょう。今、初期値が仮に

$$\theta_1^{(0)}=k_1, \theta_2^{(0)}=k_2, \ldots, \theta_n^{(0)}=k_n$$

と D の範囲内にある一定の値に決まっているものとします。ここから、

$$\theta_1^{(k)} \sim p(\theta_1 \mid \theta_2^{(k-1)}, \theta_3^{(k-1)}, \ldots, \theta_n^{(k-1)}, D)$$
$$\theta_2^{(k)} \sim p(\theta_2 \mid \theta_1^{(k)}, \theta_3^{(k-1)}, \ldots, \theta_n^{(k-1)}, D)$$
$$\cdots\cdots\cdots\cdots$$
$$\theta_{n-1}^{(k)} \sim p(\theta_{n-1} \mid \theta_1^{(k)}, \theta_2^{(k)}, \ldots, \theta_{n-2}^{(k)}, \theta_n^{(k-1)}, D)$$
$$\theta_n^{(k)} \sim p(\theta_n \mid \theta_1^{(k)}, \theta_2^{(k)}, \ldots, \theta_{n-2}^{(k)}, \theta_{n-1}^{(k)}, D)$$

上のような上付きのｋというステージをｋ＝１から times 番目まで times 回繰り返して、

$$\theta=(\theta_1, \theta_2, \ldots, \theta_n)$$

をシミュレーションにより求めます。これが Gibbs Sampling です。これを２つのパラメータからなる離散分布にもとづくもので考えたのが下の例です。すなわち、初期を仮に(1,1)から始めるために、Ｄの分布の範囲内の$\theta_1^{(0)}=1, \theta_2^{(0)}=1$と止めておいて、

$$\theta_1^{(k)} \sim p(\theta_1 \mid \theta_2^{(k-1)}, D)$$
$$\theta_2^{(k)} \sim p(\theta_2 \mid \theta_1^{(k)}, D)$$

の第ｋステップを逐次的に times 回繰り返すものです。$\theta_1=\{1,2\}, \theta_2=\{1,2,3\}$として、離散 Joint 分布

$$\begin{bmatrix} 0.1 & 0.2 & 0.2 \\ 0.1 & 0.1 & 0.3 \end{bmatrix}$$

という確率分布でこれらのパラメータが出てくるものとします。下の例では、θ_1とθ_2がそれぞれ離散 Joint 分布の行と列のインデックス番号ａとｂとします。初期値(a,b)=(1,1)から（Ｄは０を取らないので０と置いてはいけません）、それぞれの列それに行に注目して Conditional 分布を計算して、その確率にもとづいて逐次的にａを、そしてその当期のａを用いて（その行に注目して Conditional 分布を求めて）ｂを求めています。プログラムでは usample(x,p)に対して、ｘ＝{1,2}を前期のｂ列に注目した Conditional 分布から取りだし、ａとして、さらにその当期のａを使って、x={1,2,3}をａ行の Conditional 分布から取り出す計算をするものが、

```
a=usample(seqa(1,1,rows(joint)),joint[.,b]/sumc(joint[.,b]));
b=usample(seqa(1,1,cols(joint)),joint[a,.]'/sumc(joint[a,.]'));
```

という部分になっています。つまり、{1,2}および{1,2,3}の中から、それぞれの列または行にもとづいた Conditional 分布確率でサンプリングしています。その際、最初のサンプリングではｂは前期のもの、第２式のサンプリングでは、第１式で得られたその期のａにもとづいてサンプリングが行なわれます。このステージを初期値のａとｂを固定してやってから、times 回繰り返してａとｂの出方を調べて、ma および mb という列ベクトルに格納していきます。これらの列ベクトルが Gibbs Sampling によるシミュレーション結果です。

プログラム

```
new; cls;
rndseed 1000;
joint={0.1 0.2 0.2,
       0.1 0.1 0.3};
call dgsample(joint,1000);

proc dgsample(joint,times);
    local a,b,ma,mb,mjoint,i,result;
    a=1; b=1;   @ Beginning from (1,1). @
    ma=zeros(times,1); mb=zeros(times,1);
    mjoint=zeros(rows(joint),cols(joint));
    i=1;
    do while i<=times;
        a=usample(seqa(1,1,rows(joint)),joint[.,b]/sumc(joint[.,b]));
        b=usample(seqa(1,1,cols(joint)),joint[a,.]'/sumc(joint[a,.]'));
        ma[i]=a;
        mb[i]=b;
        mjoint[a,b]=mjoint[a,b]+1;
        i=i+1;
    endo;
    result=mjoint/sumc(sumc(mjoint));
    print joint;
    print result;
    retp(result);
endp;

proc usample(x,p);
    local n,u,y;
    n=rows(x);
```

```
if n>10;
    errorlog "ERROR: This function can sample only n<=10.";
    retp(".");
endif;
if rows(x)/=rows(p);
    errorlog "ERROR: Rows of x and its p must be the same.";
    retp(".");
endif;
_fcmptol=1e-2;
if fne(sumc(p),1);
    errorlog "ERROR: Sum of the probabilities must be 1.";
    retp(".");
endif;
u=rndu(1,1);
if u>=0 and u<p[1];
    y=x[1];
elseif u>=p[1] and u<(p[1]+p[2]);
    y=x[2];
elseif u>=p[2] and u<(p[1]+p[2]+p[3]);
    if rows(x)<3;
        y=x[2];
    else;
        y=x[3];
    endif;
elseif u>=p[3] and u<(p[1]+p[2]+p[3]+p[4]);
    if rows(x)<4;
        y=x[3];
    else;
        y=x[4];
    endif;
elseif u>=p[4] and u<(p[1]+p[2]+p[3]+p[4]+p[5]);
    if rows(x)<5;
        y=x[4];
    else;
        y=x[5];
    endif;
```

```
    elseif u>=p[2] and u<(p[1]+p[2]+p[3]+p[4]+p[5]+p[6]);
        if rows(x)<6;
            y=x[5];
        else;
            y=x[6];
        endif;
    elseif u>=p[2] and u<(p[1]+p[2]+p[3]+p[4]+p[5]+p[6]+p[7]);
        if rows(x)<7;
            y=x[6];
        else;
            y=x[7];
        endif;
    elseif u>=p[2] and u<(p[1]+p[2]+p[3]+p[4]+p[5]+p[6]+p[7]+p[8]);
        if rows(x)<8;
            y=x[7];
        else;
            y=x[8];
        endif;
    elseif u>=p[2] and u<(p[1]+p[2]+p[3]+p[4]+p[5]+p[6]+p[7]+p[8]+p[9]);
        if rows(x)<9;
            y=x[8];
        else;
            y=x[9];
        endif;
    else;
        y=x[10];
    endif;
    retp(y);
endp;
```

画面表示

```
        0.10000000        0.20000000        0.20000000
        0.10000000        0.10000000        0.30000000

        0.10400000        0.19600000        0.21100000
       0.096000000        0.10500000        0.28800000
```

最初に示した離散 Joint 分布の値に近い割合で出てくることが上の結果からわかると思います。これは、それぞれのステージで Gibbs Sampling で得られたａおよびｂを使って、mjoint[a,b]=mjoint[a,b]+1;によって、その配列(a,b)のところを１足して勘定していき、最終的に、result=mjoint/sumc(sumc(mjoint));で列と行の合計の分母分の２×３の行列の要素のそれぞれの頻度を分子にもって行くことで、合計が１の頻度の割合が計算されます。プログラム的には、GAUSS にはある確率でサンプリングする関数は存在しないのでモンテカルロシミュレーションの最初のところでやったように一様分布 U[0,1)を用いて、その区間を確率分布の割合であらかじめ区切っておいて、その一様乱数の出方に応じてその区間に相当する目の出方を計算してやります。それが、usample の procedure です。これは 10 個のサンプルまで合計が１になる離散分布にもとづいて引くことができます。

技術的には、procedure usample(x,p)において、インプットのｘとｐは列で与えられるのですが、エラーチェックを冒頭でやってやります。ｘの行数（すなわちサンプルの個数）ｎが 10 よりも大きいときは想定していないので If 文で除外します。同様に、ｘとｐの行数が一致していない設定、それに、ｐの合計が１におおよそのところならない設定も除外してやります。なお、「おおよそ」１にならないかどうかを判別するために、fne(a,b)というファジー組込み関数を用いて、その前にそのグローバル設定で 1e-2 で 0.01 に許容桁数を設定してやります。これは 0.3333…が３つの場合など必ずしも合計が１にならないものにも対応しています。メインのアルゴリズムは、０から１までの区間を確率で区切ってやって、その中にあればどれを選択するのかを決めています。ただし、確率の最終合計が１よりも小さい場合、次の elseif に行ってしまうので、これを防止するために、

```
if rows(x)<3;
    y=x[2];
else;
    y=x[3];
endif;
```

などというように、サンプルの個数がそのステージの区切り番号よりも小さい場合には、１つ戻ってサンプルのリターンｙを設定しています。そうでない通常の場合は、else 以降に行って一番大きいインデックスのサンプルがｙに設定されます。

上の離散 Gibbs Sampling のケースは、Joint 分布の確率行列が２×３の場合ですが、同じプログラムで冒頭の行列を変更することによって、最大 10×10 まで計算できます。またパラメータｃを加えて、式を一本加えて、３本の式を a,b,c を代入していくものにすれば、３者によるような３次元のケースにも簡単なプログラムの変更によって対応できます。これらは、ゲーム戦略などの行列にもすぐに実用できる分野です。

次に、今度は離散確率分布ではなく一般的なケースを考えます。今度は、上の行列のインデックスであるａおよびｂに相当するところを平均μと Var のσ²として、その初期値をそれぞれ０として Gibbs Sampling で逐次的にシミュレーションします。初期値μとσ²を

それぞれ０に止めておいて、上付きの(k)を第ｋ番目のステージであるとすると、

$\mu^{(k)} \sim N(xbar, \sigma^{2(k-1)}/n)$

$\sigma^{2(k)} \sim \Sigma(x-\mu^{(k)})^2/\chi^2(df=n)$

という Gibbs Sampling を繰り返します。ただし、xbar のところには、元のＸのデータを与えて、その平均を xbar とするものとします。第１ステージでは$\sigma^{2(0)}$は０ですから$\mu^{(1)}$は xbar そのものとなります。また上の過程の２つ目の式は、もともと与えるデータの個数をｎとした自由度のχ^2に対する分散ターム$\Sigma(x-\mu)^2$でもってその何倍になるかで、直感的には、ノーマライズされたものに対する割合でサンプル平均の分散を得ます。なお、１つめの式の分散はサンプル平均の分散ですから、当然のことながら、ｎで割ったものになります。これを GAUSS の書式にすれば、

```
myu=sqrt(var/n)*rndn(1,1)+xbar;
var=sumc((x-myu)^2)/rndx2(1,1,n);
```

となります。初期値 my と var をそれぞれ０としておいて、ｘベクトルの分布をあらかじめ定めておいて、そこから xbar をｘベクトルの平均としてやって、そこから出発させ times 回のステージを繰り返します。

プログラム

```
new; cls;
rndseed 1000;
x=rndnorm(20,1,5,2);            /* mean=5   sigma=2 (var=4)   */
call gsample(x,0,0,10000);      /* myu0=0 var0=0 times=1000 */

proc(2)=gsample(x,myu0,var0,times);
    local myu,var,n,xbar,myum,varm,i;
    myu=myu0; var=var0;
    n=rows(x);
    xbar=meanc(x);
    myum=zeros(times,1); varm=zeros(times,1);
    i=1;
    do while i<=times;
        myu=sqrt(var/n)*rndn(1,1)+xbar;
        var=sumc((x-myu)^2)/rndx2(1,1,n);
        myum[i]=myu;
        varm[i]=var;
        i=i+1;
    endo;
/* Display and Graph the Results */
```

```
    print "posterior mean=" meanc(myum);
    print "posterior mean of var=" meanc(varm);
    library pgraph;
    graphset;
    pqgwin many;
    call hist(myum,50);
    call hist(varm,50);
    xy(seqa(1,1,times),myum);
    xy(seqa(1,1,times),varm);
    retp(myum,varm);
endp;

proc rndx2(r,c,df);
    local m,x,i;
    m=zeros(r,c);
    i=1;
    do while i<=c;
        x=rndn(r,df);
        x=x.*x;
        m[.,i]=sumc(x');
        i=i+1;
    endo;
    retp(m);
endp;

fn rndnorm(r,c,m,s)=s*rndn(r,c)+m;
```

なお、GAUSS にはχ2 の乱数関数はありませんから、定義にしたがって標準正規乱数から作成した簡易な乱数関数を用いています。また、標準偏差がｓ、平均がmの正規乱数は、上のように標準正規乱数にｓをかけて伸ばした上で、mだけ平行移動させればできます。これらを procedure にしてプログラムの末尾に付属させて、組込み関数と同じように使っています。

画面表示

```
posterior mean=          4.9766538
posterior mean of var=          3.3166233
```

Metropolis-Hastings

この方法は、Proposal をいつも受け入れて次のステージに移行する Gibbs Sampling とは異なり、Proposal Distribution の前のものと当期のもの割合が０と１の間の一様乱数よりも大きいか小さいかによって、受け入れるのか受け入れないのかを判別して、受け入れる場合にのみ、次のステージ（または状態）に移行し、そうではない場合にはその状態にとどまるアルゴリズムです。すなわち、手順は、ある Proposal される Target Distribution があって、そこにある任意の初期状態から出発して（仮に０としましょう)、下の例では、ｘが０から出発して、今の状態のｘすなわちｘｐがｘに下限上限があらかじめ定められた一様乱数の和から得られるものとして、N(0,1)の標準正規分布の前の状態と今の状態の割合を計算して、それが０と１の間の一様乱数の値よりも大きければ、その Proposal を受諾して、ｘｐをｘに代入することによって次の状態に移行します。これを times 回繰り返したものが Metropolis-Hastings のアルゴリズムです。なお、慣習上またコンピュータの桁の実行桁の問題上、通常は、一様乱数と分布の割合の両者の自然対数をとることに比較がなされ判別されます。以下のプログラムでは、自然対数をとった分布の差、つまり分布の割合が計算され、定数部分がキャンセルアウトして、指数関数の部分のマイナスがついている項の差をとっているので、alpha=(-xp^2+x^2)/2;となっています。次の例のように完全な形で書いても、結局は定数項がキャンセルアウトされますから同じことです。

プログラム

```
new; cls;
rndseed 1000;
call mhsample(2,0,10000);

proc(2)=mhsample(up,x0,times);
    local x,mx,accept,i,xp,alpha,u,ratio;
    x=x0;
    mx=zeros(times,1); accept=0;
    i=1;
    do while i<=times;
        xp=x+rndunif(1,1,-up,up);
        alpha=(-xp^2+x^2)/2; @ Log difference of the target distribution of N(0,1). @
        u=ln(rndu(1,1));
        if u<alpha;
            x=xp;
            accept=accept+1;
        endif;
        mx[i]=x;
```

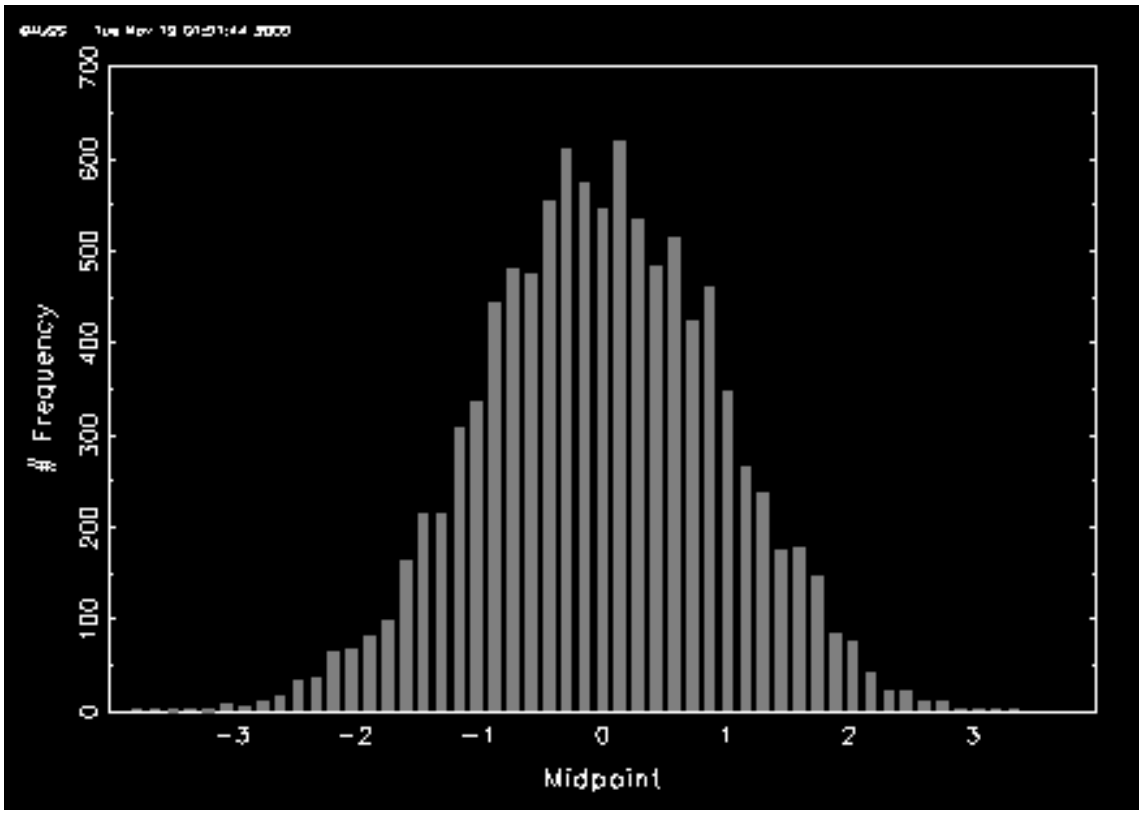
```
        i=i+1;
    endo;
    ratio=accept/times;
/* Display and Graph the Results */
    print "Acceptance Ratio=" ratio;
    print "mean=" meanc(mx);
    print "s.d.=" stdc(mx);
    library pgraph;
    graphset;
    pqgwin many;
    call hist(mx,50);
    xy(seqa(1,1,times),mx);
    retp(mx,ratio);
endp;

fn rndunif(r,c,low,up)=(up-low)*(rndu(r,c)-0.5)+low+(up-low)/2;
```

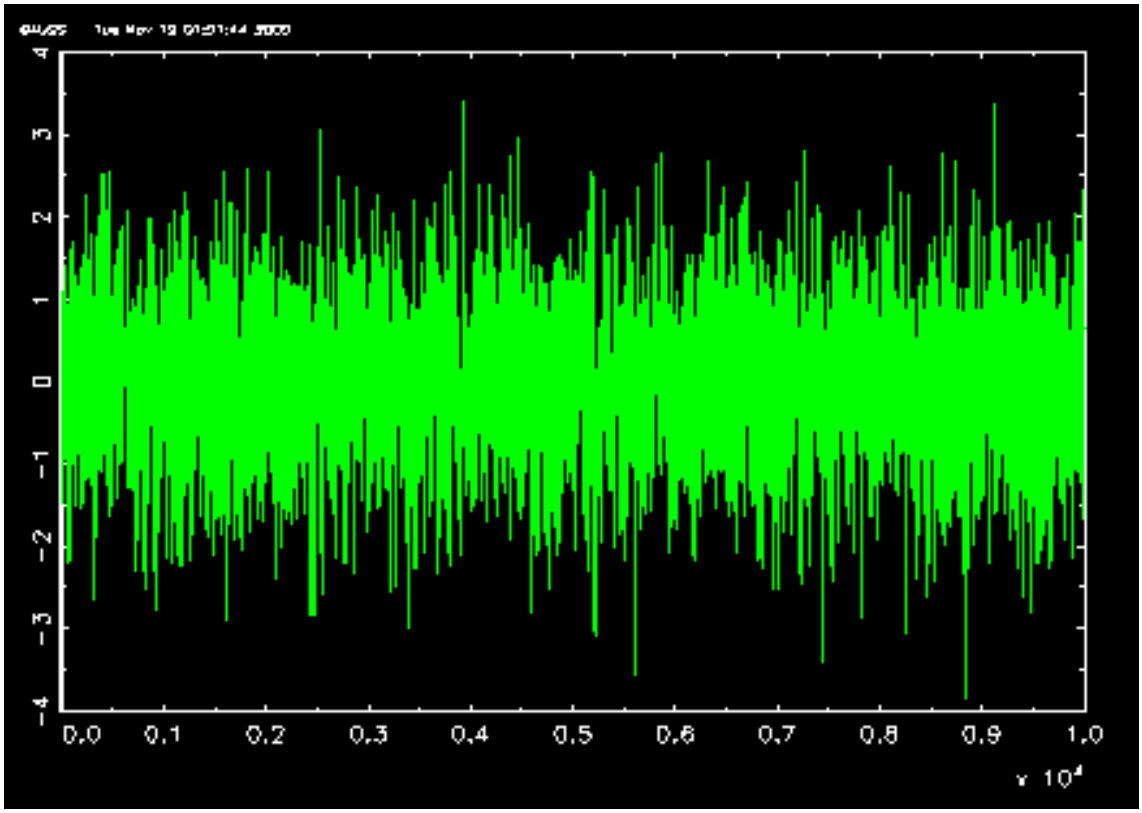
画面表示

```
Acceptance Ratio=          0.62770000
mean=       -0.016655616
s.d.=          0.97172303
```

上の結果は、初期の何期間かの切り捨てる部分を全くなしで計算しているので正確ではないが、ほぼ Target Distribution の N(0,1)に近いものになっている。このことは、以下のグラフからもわかる。

グラフ表示


Frequency
0
100
200
300
400
500
600
700
−3
−2
−1
0
1
2
3
Midpoint

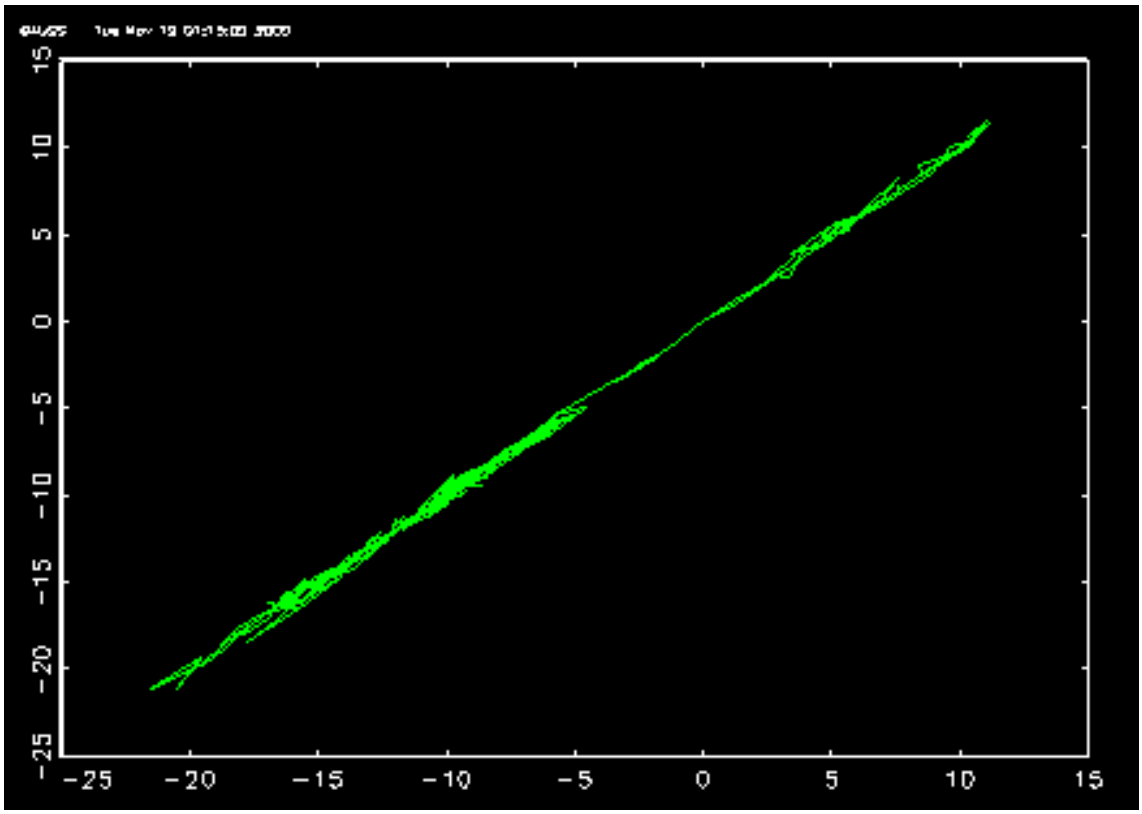

4
3
2
1
0
−1
−2
−3
−4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
x 10⁴

今度は、何がしのｘとｙからなる２次のマイナス関数の Log Target Distribution を考えてこれをもとに、ｘとｙの Bivariate のケースをプログラムしてみる。

プログラム

```
new; cls;
rndseed 1000;
fn targetdist(x,y)=-(5*x^2-2*5*x*y+5*y^2);       /* You could change this */
call mhsample(2,2,0,0,10000);   @ Bivariate Case @

proc(3)=mhsample(upx,upy,x0,y0,times);
    local x,y,mx,my,accept,i,xp,yp,alpha,u,ratio;
    x=x0; y=x0;
    mx=zeros(times,1); my=zeros(times,1); accept=0;
    i=1;
    do while i<=times;
        xp=x+rndunif(1,1,-upx,upx);
        yp=y+rndunif(1,1,-upy,upy);
        alpha=targetdist(xp,yp)-targetdist(x,y); @ Log difference of target distribution @
        u=ln(rndu(1,1));
        if u<alpha;
            x=xp; y=yp;
            accept=accept+1;
        endif;
        mx[i]=x; my[i]=y;
        i=i+1;
    endo;
    ratio=accept/times;
/* Display and Graph the Results */
    print "Acceptance Ratio=" ratio;
    library pgraph;
    graphset;
    xy(mx,my);
    retp(mx,my,ratio);
endp;

fn rndunif(r,c,low,up)=(up-low)*(rndu(r,c)-0.5)+low+(up-low)/2;
```

グラフ表示



与えられた Target 分布のケースは、ほぼ 45 度の角度をもって左上に伸びていきます。なお、iteration の数を減らすともっと短いものに、数を増やすともっと長いものになります。また、Target Distribution の-2*5*x*y の項の５のところを若干増やして、例えば、5.5 にすると、Acceptance Rate が高まります。
これは、ｘとｙのまわりの点を候補の中からピックアップしてきて、その density の比を計算して比べていることを意味します。Log difference はもともとの density の比と考えられます。上の２つのプログラム例は、Metropolis-Hastings のアルゴリズムのほんの１例に過ぎません。いろいろなバージョンがあります。

Metropolis-within-Gibbs

Gibbs Sampling は理論的にはもちろん Metropolis-Hastings のアルゴリズムの特殊なものと考えられるのだが、これを２つ組み合わせて、μおよびσ(またはσ^2)の導出のアルゴリズムにも Metropolis-Hastings を棄却法を導入したのが Metropolis-within-Gibbs です。平均０標準偏差１の初期状態から出発して、Target とする Distribution をまず、前の状態のσで止めておいてμの状態が異なる分布の割合をとって、一様乱数と大きさを比べて、大きければ Accept し、次の状態にいき、そうでなければそのままの状態にしておいて、また同じことを今度はσを止めておいて、σの状態が異なる分布の割合をとって、同様な判

別を行なって、状態を進めるのかそのままにするのかを決めます。この２つのサンプリングを times 回繰り返したものが下の Metropolis-within-Gibbs です。なお、このプログラムは、μおよびσの両方にこのアルゴリズムを施したもので、どちらか片方だけにアルゴリズムを施すこともできます。

プログラム

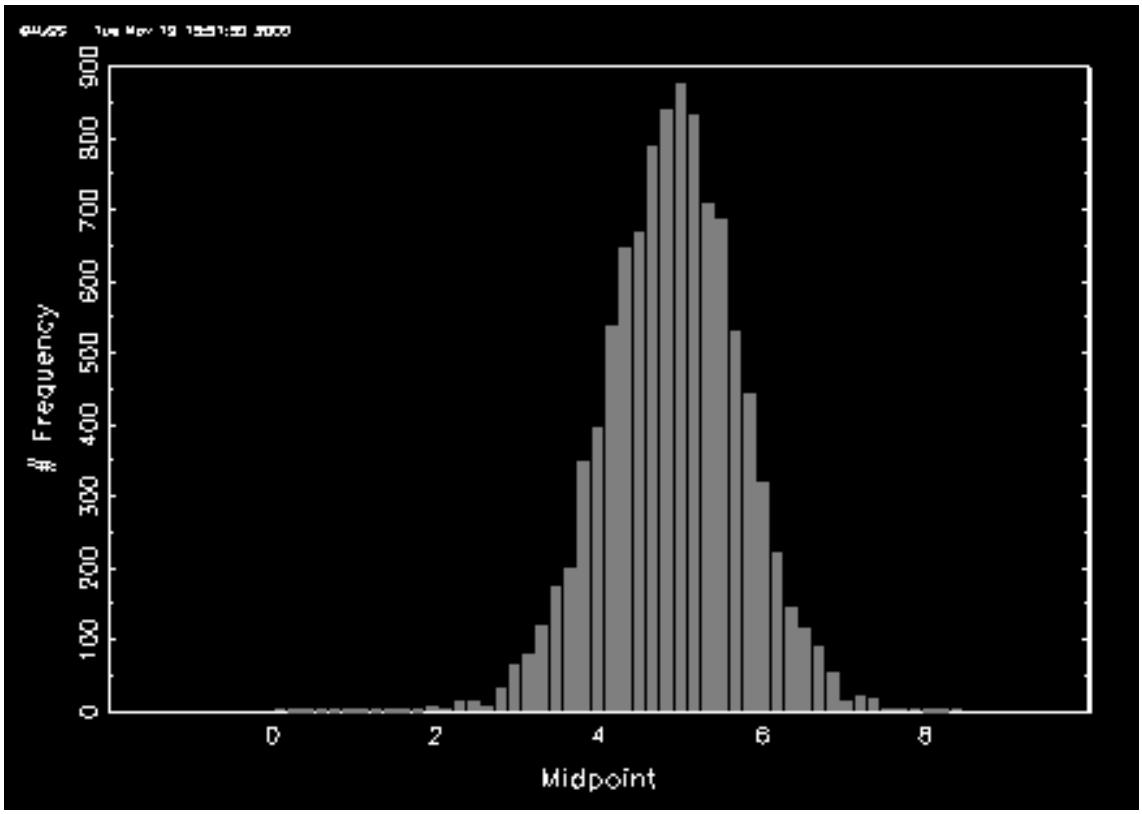
```
new; cls;
rndseed 1000;
x=rndnorm(20,1,5,4);
call mwgsample(x,0,1,2,1,10000);

proc(4)=mwgsample(x,myu0,sigma0,dmyu,dsigma,times);
    local myu,sigma,myum,sigmam,acceptmyu,acceptsigma,i;
    local myustar,sigmastar,lnalpha,u,ratiomyu,ratiosigma;
    myu=myu0; sigma=sigma0;
    myum=zeros(times,1); sigmam=zeros(times,1);
    acceptmyu=0; acceptsigma=0;
    i=1;
    do while i<=times;
        /* Sampling myu */
        myustar=myu+rndunif(1,1,-dmyu,dmyu);
        lnalpha=posterior(x,myustar,sigma)-posterior(x,myu,sigma);
        u=ln(rndu(1,1));
        if u<lnalpha;
            myu=myustar;
            acceptmyu=acceptmyu+1;
        endif;
        myum[i]=myu;
        /* Sampling sigma */
        sigmastar=sigma+rndunif(1,1,-dsigma,dsigma);
        lnalpha=posterior(x,myu,sigmastar)-posterior(x,myu,sigma);
        u=ln(rndu(1,1));
        if u<lnalpha;
            sigma=sigmastar;
            acceptsigma=acceptsigma+1;
        endif;
        sigmam[i]=sigma;
```

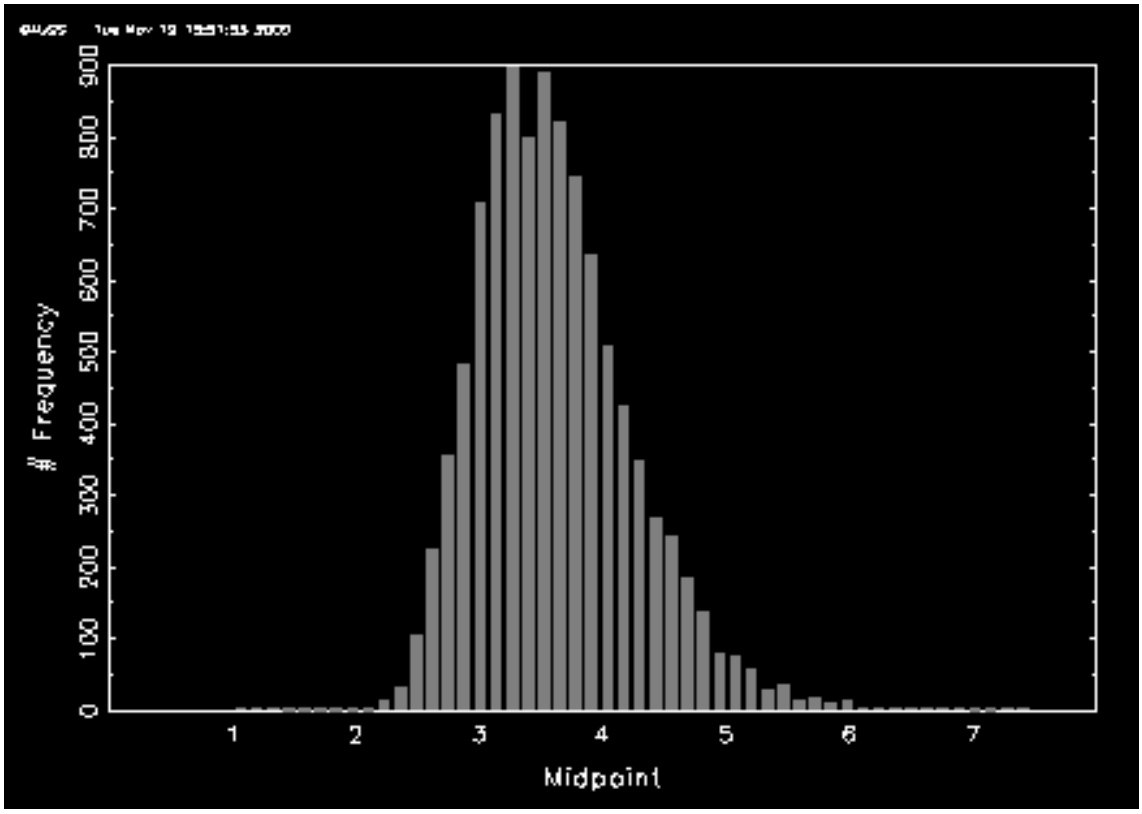
```
        i=i+1;
    endo;
    ratiomyu=acceptmyu/times;
    ratiosigma=acceptsigma/times;
/* Display and Graph the Results */
    print "Acceptance Ratio of     myu=" ratiomyu;
    print "Acceptance Ratio of sigma=" ratiosigma;
    print "posterior mean                =" meanc(myum);
    print "posterior mean of sigma    =" meanc(sigmam);
    library pgraph;
    graphset;
    pqgwin many;
    call hist(myum,50);
    call hist(sigmam,50);
    xy(seqa(1,1,times),myum);
    xy(seqa(1,1,times),sigmam);
    retp(myum,sigmam,ratiomyu,ratiosigma);
endp;

proc posterior(x,myu,sigma);
    local n,lnpost;
    n=rows(x);
    lnpost=-(n+1)*ln(sigma)-sumc((x-myu)^2)/(2*sigma^2);
    retp(lnpost);
endp;

fn rndunif(r,c,low,up)=(up-low)*(rndu(r,c)-0.5)+low+(up-low)/2;
fn rndnorm(r,c,m,s)=s*rndn(r,c)+m;
```

画面表示

```
Acceptance Ratio of     myu=         0.54980000
Acceptance Ratio of sigma=         0.65800000
posterior mean                =          4.9095802
posterior mean of sigma    =          3.6115127
```

グラフ表示


Frequency
0
100
200
300
400
500
600
700
800
900
0
2
4
6
8
Midpoint


Frequency
0
100
200
300
400
500
600
700
800
900
1
2
3
4
5
6
7
Midpoint