5.1　モンテカルロシミュレーション 根本原理　　ver.0.2

「モンテカルロ」とは、フォンノイマンとウラムなどの科学者が第二次大戦末期とその後にロスアラモス研究所やネバダの核実験場で行なっていた原子爆弾開発の際の乱数シミュレーションによる核反応実験のコードネームでした。砂漠でかたやギャンブルや享楽をするためにラスベガスへ人々はやってくる。かたや自分たち研究者は砂漠の研究所に閉じ込められて日夜原子爆弾を研究する。ちょうど、モナコの賭博場モンテカルロを思い浮かべて洒落気たっぷりにつけられたコードネームだったのでしょう。そういう中で醸成され、その後認知されるようになったシミュレーションの方法を人々はモンテカルロ法と呼ぶようになりました。その後、偏微分方程式などの解法や逆行列の解法をこの乱数シミュレーションによって行なう応用が数学工学などに広まり、ＯＲなどにも後に広まります。現在では、その方法が経済学計量経済学にも伝わったものです。面白いことに、そうした結果が不確かな核開発「モンテカルロ」に対して、結果はいつも同じ解になるのですがで途中経路は、乱数化ソートなどの、乱数化されたいくつもの経路をとるシミュレーションの方法「ラスベガス」というのも存在します。ただし、現在では、少なくとも統計学や計量経済学ではこの両者の区別はなく、乱数を使って擬似的に得たデータを用いて、またはそうした擬似的な手続きを経て、分析をすることの総称を広い意味でのモンテカルロシミュレーションと呼んでいます。

通常のソフトウエアには、一様分布の乱数を発生させる関数しかないのが普通でしたがGAUSSが「乱数に強い」と呼ばれるようになったのは、ソフトウエアの名前からも明らかなように、標準正規分布を含む、ほとんどすべての乱数を発生させる関数を標準で装備している点にあります。ここでは、通常のモンテカルロ法の教科書などで行なわれるような、円の面積からはじまって境界問題や積分といった工学等で語られるような計量経済学とは無縁の内容も扱わないことにします。もちろん、原子物理学で発達した原子炉の中での中性子の動きのシミュレーションなども扱いません。また、一様乱数からボックス・ミューラー法によって標準正規乱数を作りだし、それらをもとにまたその他の分布の乱数をつくることは扱いません。なぜなら、GAUSS自体がボックス・ミューラー法に準拠しない他のやりかたで成り立っているからです。むしろ、ここでは信頼性をもって考案されているそれぞれの乱数関数やＰＤＦ、ＣＤＦから出発し、それを経済学周辺でどのように利用するかに重点を置きます。極めて単刀直入な実践的な内容で話を進めます。

(Weak) Law of Large Numbers(大数の(弱)法則)

ギャンブルで、１から６までの目をもつサイコロをふったしましょう。いかさまがなければ、どの目も１／６ずつの確率で出るはずです。ここで、合計６回ちょうどふったとしても、それぞれの目が１回ずつ出るとはかぎりません。そうでないことがほとんでしょう。しかしながら、それを百回、千回、一万回とふり続けると、徐々に１／６の確率でそれぞ

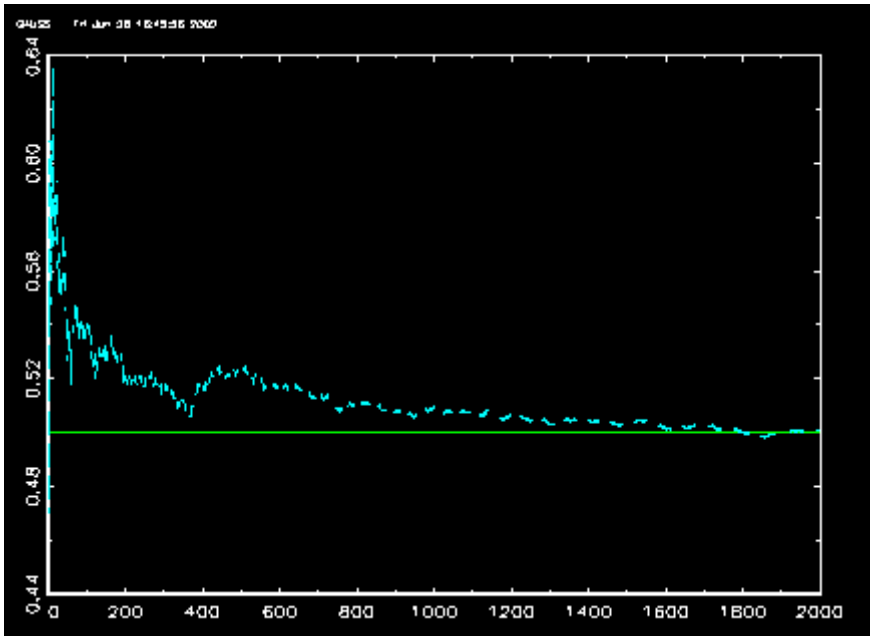
れの目がでるように近づいていきます。これを、おおよそのところ、大数の(弱)法則と呼びます。乱数もモンテカルロシミュレーションもすべて、大数の(弱)法則の上に成り立っています。平均であるとか、上の例でいうとそれぞれの目のでる確率であるとかが、その期待値（この場合）１／６からハズレる確率が、一回きりの試行であると当然のことながら大幅にはずれるが、試行を繰り返すにしたがってゼロに近づいていくということです。

プログラム(一様分布の平均への収束)

```
    new; cls;
    call uniformsim(2000);

proc(0)=uniformsim(n);
    local m,t,e,myu;
    m=rndu(n,1);
    t=seqa(1,1,n);
    e=cumsumc(m)./t;
    myu=0.5*ones(n,1);
        library pgraph;
        graphset;
        xy(t,myu~e);
endp;
```

グラフ表示



上のプログラムは、擬似一様乱数を発生させる組込み関数を用いて、rndu(n,1)としてｎ行１列の乱数を作りｍとします。そして、１から１刻みのｎまでのシークエンスをｔとしま

す。乱数のうちの上からｔまでの乱数の合計の平均は、cumsumc(m)./t で表せます。これをｙ軸に基準線のμ＝0.5 とともに、ｘ軸のｔのシークエンスに対してｘｙプロットしたものが上のグラフです。この場合一回で、列として乱数を発生させています。もちろん、１階ずつ rndu(1,1)を繰り返してもかまいません。なお、GAUSS では通常のデータと同じように列方向に乱数を展開するのが慣例となっています。

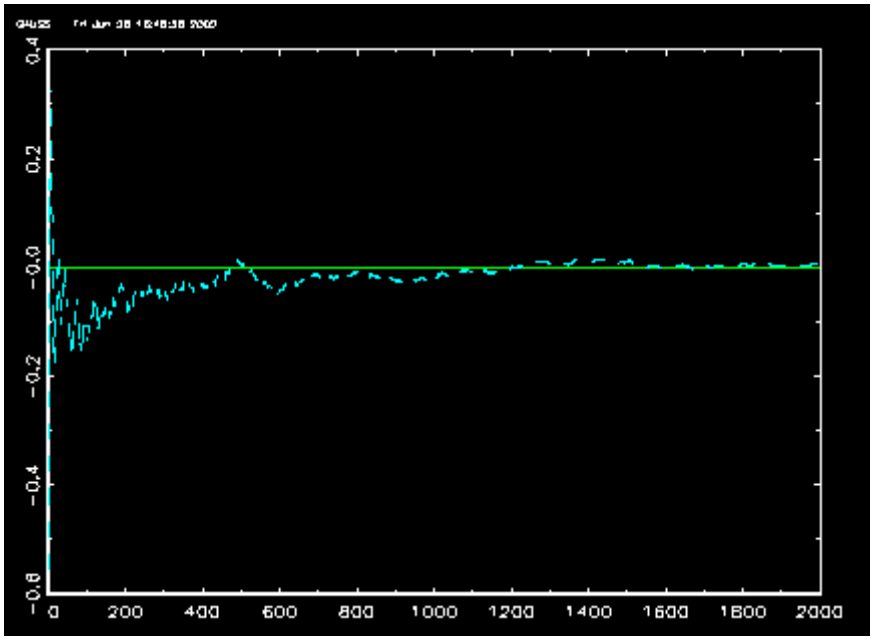
今度は、同じことを標準正規乱数に対しても行ないます。

プログラム（標準正規分布の平均へ収束）

```
        new; cls;
        call snsim(2000);


proc(0)=snsim(n);
    local m,t,e,myu;
    m=rndn(n,1);
    t=seqa(1,1,n);
    e=cumsumc(m)./t;
    myu=zeros(n,1);
        library pgraph;
        graphset;
        xy(t,myu~e);
endp;
```

グラフ表示



上のプログラムでは、一様乱数の組込み関数 rndu を標準正規乱数の組込み関数 rndn に変

更しただけのものです。ただし、基準線は当然のことながらμ＝0 になります。グラフより標準正規乱数も、一様乱数と同様に、回数を重ねるにしたがって平均値はμの値に収束していくことがわかると思います。

今度は、ｎ回の繰り返しからなる成功確率ｐ＝0.4 のベルヌーイ試行における成功の回数から導出される２項分布にしたがう乱数をプログラムして、同様に大数の法則を平均値について見るものです。

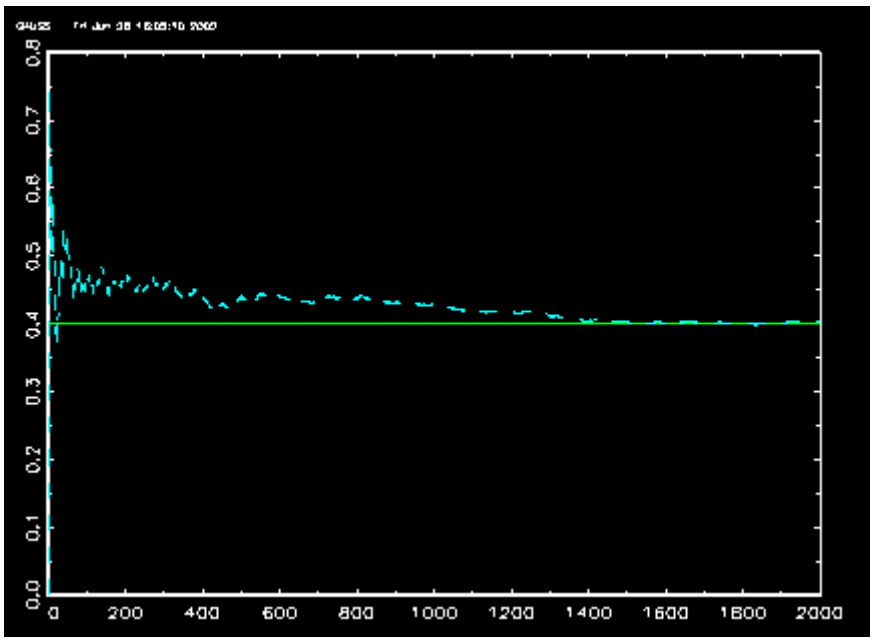
プログラム（２項分布の期待値ｐへの収束）

```
    new; cls;
    call bsim(2000,0.4);

proc bsim(n,p);
    local m,i,u,t,e,pline;
    m=zeros(n,1);
    i=1;
    do while i<=n;
        u=rndu(1,1);
        if u<=p;
            m[i]=1;
        else;
            m[i]=0;
        endif;
        i=i+1;
    endo;
    t=seqa(1,1,n);
    e=cumsumc(m)./t;
    pline=p*ones(n,1);
        library pgraph;
        graphset;
        xy(t,pline~e);
    retp(e);
endp;
```

ここでは、一様乱数を使って、でてきたものが成功確率ｐ以下である場合には成功として１カウントして、そうでない場合にはカウントしない（すなわち０である）という条件分岐で成功回数の分布にしたがう乱数に変換して、それを以前の２つのプログラムと同じように、その時点までの平均値をだしてやって、成功確率を基準線としたグラフで、その収束具合をプロットしているものです。

グラフ表示



上の２つのグラフと同様、収束していく様子がわかると思います。初期に１か０のどちらが多いかによって、その時々の平均値は期待値の基準線よりも上から収束していく場合も下から収束していく場合もどちらもあり得ます。ここでは乱数のシードを設定していませんから、プログラムを run するたびにグラフは変化します。

今までは、大数の（弱）法則の定義どおり平均値の収束を見たわけですが、大数の法則というのは、もっと一般的には、分布自体が試行を繰りかえすにしたがって滑らかに理想的な形になっていくというイメージがあります。これを一様乱数と標準正規乱数についてプログラムしてみます。

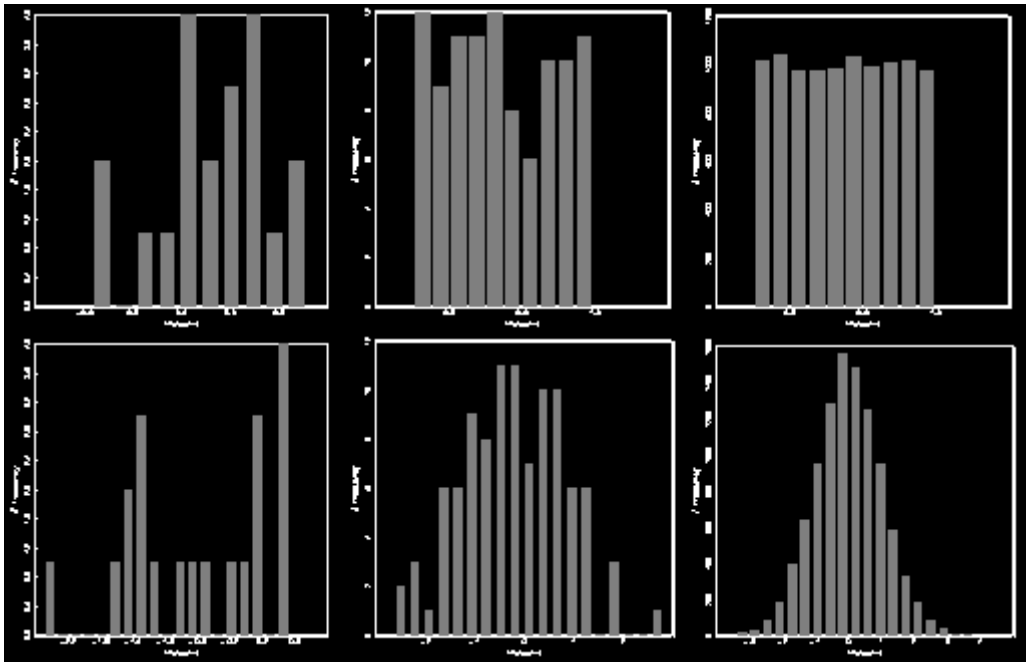
プログラム

```
new; cls;
call largesim(10000,20,100,10000);


proc(0)=largesim(n,n1,n2,n3);
local uniform,snormal;
uniform=rndu(n,1);
snormal=rndn(n,1);
    library pgraph;
    graphset;
    begwind;
    window(2,3,0);
    setwind(1);
```

```
        hist(uniform[1:n1],10);
    setwind(2);
        hist(uniform[1:n2],10);
    setwind(3);
        hist(uniform[1:n3],10);
    setwind(4);
        hist(snormal[1:n1],19);
    setwind(5);
        hist(snormal[1:n2],19);
    setwind(6);
        hist(snormal[1:n3],19);
    endwind;
endp;
```

グラフ表示



上段は、一様分布の乱数が試行をn=20,100,10000と増やすにつれて理想的な分布に近づいていく様子を示しています。下段は、その標準正規分布によるものです。プログラム自体はシンプルで２つの乱数列を発生させているだけなのですが、そのあと、I／Oの節のグラフ(3)で説明しましたように、begwind; window(2,3,0); ． ． ． .endwind;の部分で２行３列非透過のウインドウを起動し何かをして終了させています。その間に、setwind(1)から順に６つの指定をしてそのあとにグラフ関数xyを置いています。なお、独自にグラフに文字やタイトルを入れたりそれぞれの設定をする場合には、それぞれのsetwind(番号)のあとにgraphsetでグローバル変数をその都度初期化する必要があります。ここでは、何もしてい

ませんので、通常通り一括してgraphsetをlibrary pgraphのあとに置いています。乱数列の１行目から、あらかじめ引数として設定されたn1,n2,n3のところまでの乱数の分布をそれぞれのヒストグラムhistで描かせています。なお、ここでの第２引数はヒストグラムの区割りの数です。正規分布などの中心があるものは、奇数の設定がよいでしょう。

この大数の法則は、実社会ではサンプルやポピュレーションが大きいときに、たとえそれぞれの個人や企業が生存する期間が短かろうとなかろうと、全体として独立した試行の回数が十分に大きいと考えこれが成り立っているものとします。経済学や計量経済学に限らず保険や金融工学の分野でも、この大数の法則がシミュレーションの根本にあります。

Critical Value by Simulation

以下では、大数の法則が成り立っているとして、分布や Critical Value が Exact にわかっている場合について、その Critical Value とシミュレーションによって求められた値とを比較してみます。

プログラム

```
new; cls;
call snsim(1000,10000,0.025);

proc(0)=snsim(n,times,p);
    local m,i,u;
    m=zeros(times,1);
    i=1;
    do while i<=times;
        u=rndn(n,1);
        u=sortc(u,1);
        m[i,1]=u[round(n*(1-p)),1];
        i=i+1;
    endo;
    print "Standard Normal Distribution";
    print/rz "Upper" p*100 "%";
    print "Simulated Critical Value:" sumc(m)/times;
    print "     Actual Critical value:" cdfni(1-p);
endp;
```

画面表示

```
Standard Normal Distribution
Upper                    2.5 %
Simulated Critical Value:          1.9501864
```

Actual Critical value:　　1.9599640

上のプログラムでは、標準正規乱数によって作られたｎ×１の列ベクトルを sortc で昇順にソートし、その上位ｐ%のところの値（つまり round(n*(1-p))番目の行の数値）をｍ列ベクトルのｉ行目に格納していき、ループで times 分だけまわします。ｍに入っている上位ｐ%の位置の数値を平均したものを最終的に計算して、それを simulated critical value にしています。GAUSS では代表的な分布の CDF と PDF は Exact に決まっていますから、組込み関数 cdfni(1-p)で標準正規分布の(1-p)のときの位置のインバースマッピングを得て、それを actual critical value にしています。同様ににして、その他いろいろな分布について同じことができます。このようにｎ個の乱数を発生させることをさらに times 回シミュレートすることをよくやります。また、データと同様に慣習的に列でシミュレーションします。

Central Limit Theorem(中心極限定理)

同じ分布にしたがう独立な確率変数の和は、試行を繰り返すにしたがって、正規分布に近づく。さらに、もともとの分布の平均μと分散 var がわかっていれば、

$$\frac{\overline{X}-\mu}{\sqrt{\mathrm{var}/n}} \sim N(0,1)$$

（標準正規分布）に十分に試行の回数が大であれば近似できるというものである。

プログラム

```
new; cls;
call clt("beta",1000,10000);

proc(0)=clt(opt,n,times);
    local dist,m,i,x,r,a,b,k,p,myu,var,name;
    m=zeros(times,1);
    i=1;
    do while i<=times;
        if opt$=="uniform";
            x=rndu(n,1);
            myu=0.5;
            var=1/12;
            m[i]=(sumc(x)/n-myu)/sqrt(var/n);
        elseif opt$=="poisson";
            r=2;
            x=rndp(n,1,r);
            myu=r;
            var=r;
```

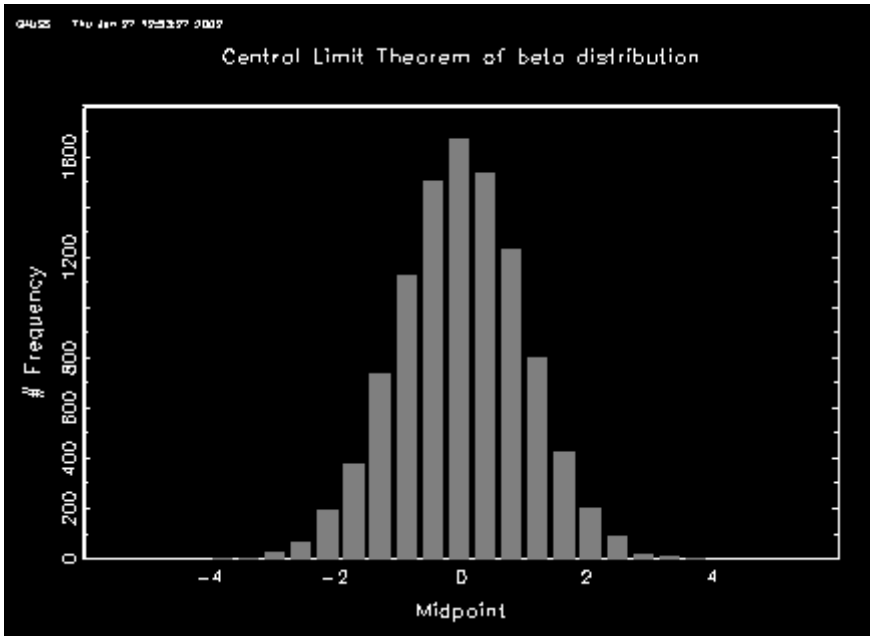
```
        m[i]=(sumc(x)/n-myu)/sqrt(var/n);
    elseif opt$=="beta";
        a=1;
        b=1;
        x=rndbeta(n,1,a,b);
        myu=a/(a+b);
        var=a*b/((a+b+1)*(a+b)^2);
        m[i]=(sumc(x)/n-myu)/sqrt(var/n);
    elseif opt$=="gamma";
        a=1;
        x=rndgam(n,1,a);
        myu=a;
        var=a;
        m[i]=(sumc(x)/n-myu)/sqrt(var/n);
    elseif opt$=="negative binomial";
        k=1;
        p=0.25;
        x=rndnb(n,1,k,p);
        myu=k*p/(1-p);
        var=k*p/(1-p)^2;
        m[i]=(sumc(x)/n-myu)/sqrt(var/n);
    else;
        errorlog "ERROR:Check back the input";
        print "The first input must be one of the following:";
        print "     uniform";
        print "     poisson";
        print "     beta";
        print "     gamma";
        print "     negative binomial";
        retp(-1);
    endif;
    i=i+1;
endo;
library pgraph;
graphset;
name="Central Limit Theorem of "$+opt$+" distribution";
```

```
    title(name);
    hist(m,19);
    print "mean" meanc(m);
    print " var" stdc(m)^2;
endp;
```

画面表示

```
mean     -0.0014575121
 var          0.99252170
```

グラフ表示



上の結果とグラフは、ベータ分布にしたがう独立な乱数をｎ個を発生させその合計をとり、それを times 回繰り返したものです。Critical value のプログラムと同様に、１回ずつの乱数の合計と分布の平均と分散によって truncated された (sumc(x)/n-myu)/sqrt(var/n)の値を列ベクトルｍのｉ行目に格納して、最後にこれをヒストグラムで描かせています。上のプログラムでは、ベータ分布も含めて５つの分布にしたがう乱数を用いて Central Limit Theorem を確かめることができます。変更するには、proc clt の第１インプットのところに引用符にくるんだ形で文字列として uniform や、poisson、beta、gamma、negative binomial を入れてみてください。同様におおよそ標準正規分布になると思います。それぞれの分布にはパラメーターがありますから、それぞれのパラメータを内部で変えてみて、それでも結果が同じになることを確かめてみるとおもしろいでしょう。

Random Walk

ここでは、サイコロをふるといったギャンブルのときと同じように一様分布にしたがう乱数を用いて、簡単なランダムウオークを作成してみましょう。今、[0,1)の一様分布にしたがう乱数をもとに、二次元のグリッド上を上下左右の４方向に動くとします。この一様分布にしたがう乱数を４倍すると、区間は[0,4)ということになります。これを組込み関数 floor（ほかの言語では int ということが多い）で小数点以下切り捨てにします。できあがった０、１、２、３の４つの数によって、下のような動きをさせるものです。

(x,y) → (x+1,y) 右 if 0
(x-1,y) 左 if 1
(x,y+1) 上 if 2
(x,y-1) 下 if 3

なお、動く距離は１で一定であるとし、グリッド上だけを動くことにします。

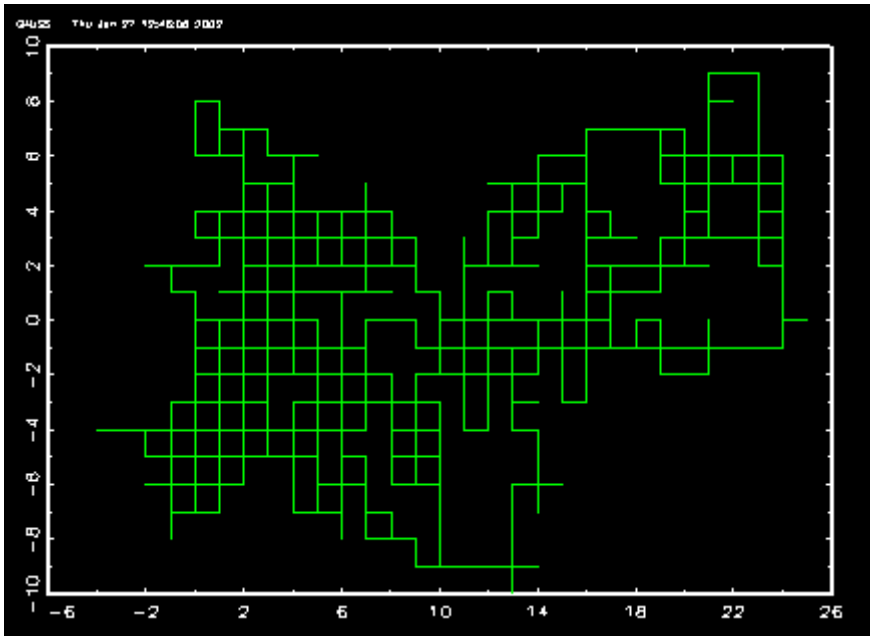
プログラム(2D Lattice バージョン)

```
new; cls;
call randomwalk(0,0,1000);

proc randomwalk(x,y,n);
    local m,i,p;
    m=zeros(n,2);
    i=1;
    do while i<=n;
        p=floor(4*rndu(1,1));
        if p==0;
            x=x+1; y=y;
        elseif p==1;
            x=x-1; y=y;
        elseif p==2;
            x=x; y=y+1;
        elseif p==3;
            x=x; y=y-1;
        else;
            errorlog "ERROR";
        endif;
        m[i,.]=x~y;
        i=i+1;
    endo;
```

```
        library pgraph;
        graphset;
        xy(m[.,1],m[.,2]);
    retp(m);
endp;
```

グラフ表示



上のプログラムでは、規則にしたがって場合わけをしてｎ回ループをまわします。それぞれの回数でのｘ座標とｙ座標を水平方向にマージしたものを列ベクトルｍのｉ行目に格納して、最後に再びこのｍの１列目をｘ、２列目をｙとしてｘｙプロットさせています。

今度は、動く距離は１のままで一定ですが、グリッド上ではなくて、360 度すべての方向に等しく動けるものとします。角度をθ＝２π*rndu としてランダムにし、これを

(x,y)　→　(x+cos θ,y+sin θ)

というふうに変換しています。ここでも、移動距離は１で一定であるとします。

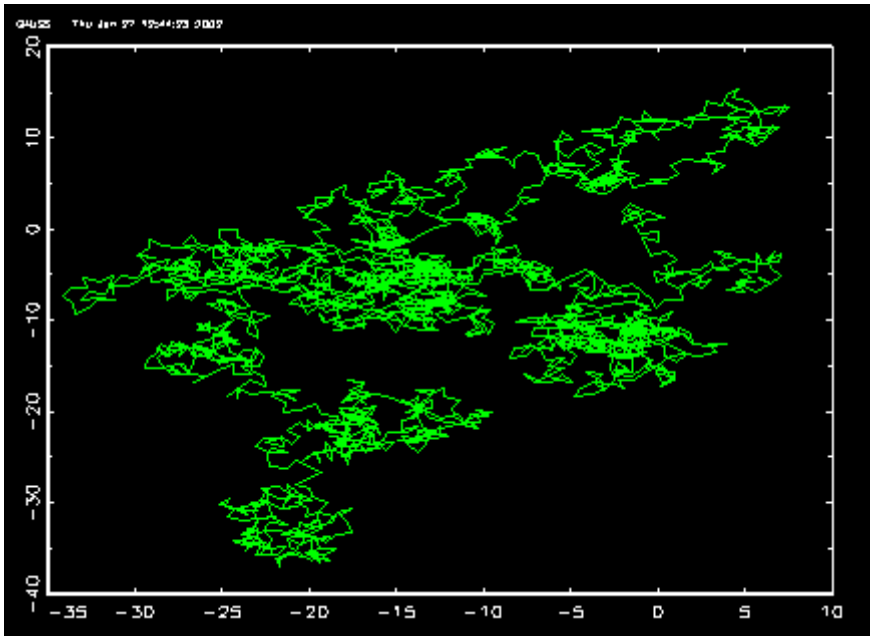
プログラム(2D Off Lattice バージョン)

```
    new; cls;
    call randomwalk2(0,0,2000);

proc randomwalk2(x,y,n);
    local m,i,theta;
    m=zeros(n,2);
    i=1;
```

```
    do while i<=n;
        theta=2*pi*rndu(1,1);
        x=x+cos(theta); y=y+sin(theta);
        m[i,.]=x~y;
        i=i+1;
    endo;
        library pgraph;
        graphset;
        xy(m[.,1],m[.,2]);
    retp(m);
endp;
```

グラフ表示



格子があるときよりもだいぶん見た目がランダムになっていることがわかると思います。プログラム方法としては、ｎ回ループをまわして、それぞれのｘｙ座標をマージして、列ベクトルmのｉ行目に格納していくのは、その前のプログラムとまったく同じです。

今度は、最初のグリッド上のものにもどって、今度は３次元の動きを考えます。上下左右奥手前の６つの場合を考えます。これは一様分布を６等分するか、または６倍したものを小数点以下切り捨てによって０，１，２，３，４，５の６つの乱数を用いてｘ、ｙ、ｚの３つの変数を１つだけいずれかの方向に増やすか減らすことによって実現できます。

(x,y,z) → (x+1,y,z) 右　　if　0
(x-1,y,z) 左　　if　1
(x,y+1,z) 奥　　if　2

(x,y-1,z) 手前　if 3
(x,y,z+1) 上　if 4
(x,y,z-1) 下　if 5

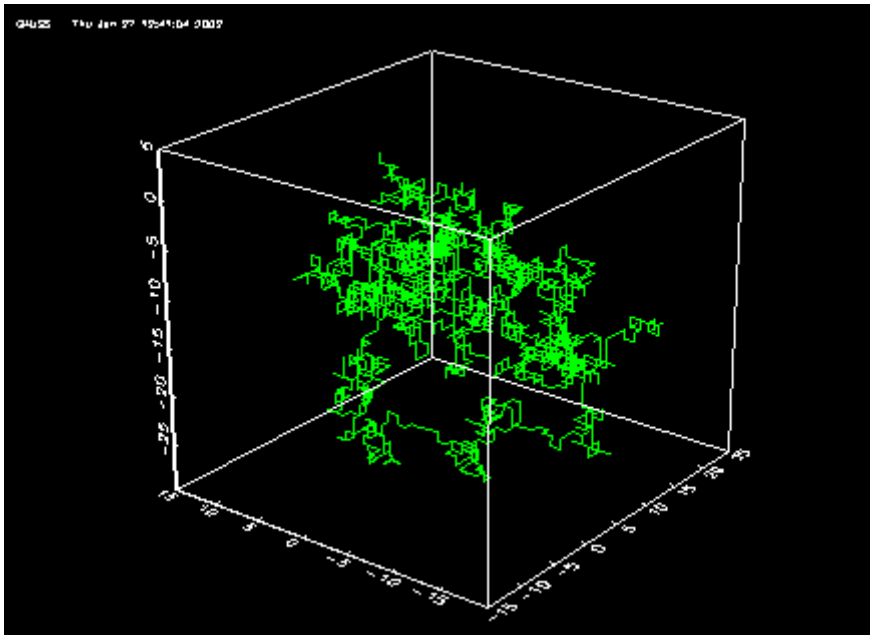
というパターンによって実現します。ここでも移動距離は常に１で一定であるとします。

プログラム(3D Lattice バージョン)

```
new; cls;
call randomwalk3(0,0,0,2000);

proc randomwalk3(x,y,z,n);
    local m,i,p;
    m=zeros(n,3);
    i=1;
    do while i<=n;
        p=floor(6*rndu(1,1));
        if p==0;
            x=x+1; y=y; z=z;
        elseif p==1;
            x=x-1; y=y; z=z;
        elseif p==2;
            x=x; y=y+1; z=z;
        elseif p==3;
            x=x; y=y-1; z=z;
        elseif p==4;
            x=x; y=y; z=z+1;
        elseif p==5;
            x=x; y=y; z=z-1;
        else;
            errorlog "ERROR";
        endif;
        m[i,.]=x~y~z;
        i=i+1;
    endo;
        library pgraph;
        graphset;
        xyz(m[.,1],m[.,2],m[.,3]);
    retp(m);
```

endp;
グラフ表示



プログラム方法は、２次元のものの４分岐を６分岐になおしただけです。十分に試行の回数が多ければ、０から１の一様乱数を４倍にのばしても６倍にのばしても高さがその分低く truncated されるだけで一様乱数であることにはかわりありません。また、中心極限定理でやった分布と分散によって truncated された逆のようなことも標準正規乱数に対して可能で、標準偏差でテイルをのばしたり縮めたりして高さを上下させたり、平均で平行移動したりできます。これらの手法もシミュレーションの基礎となるところです。

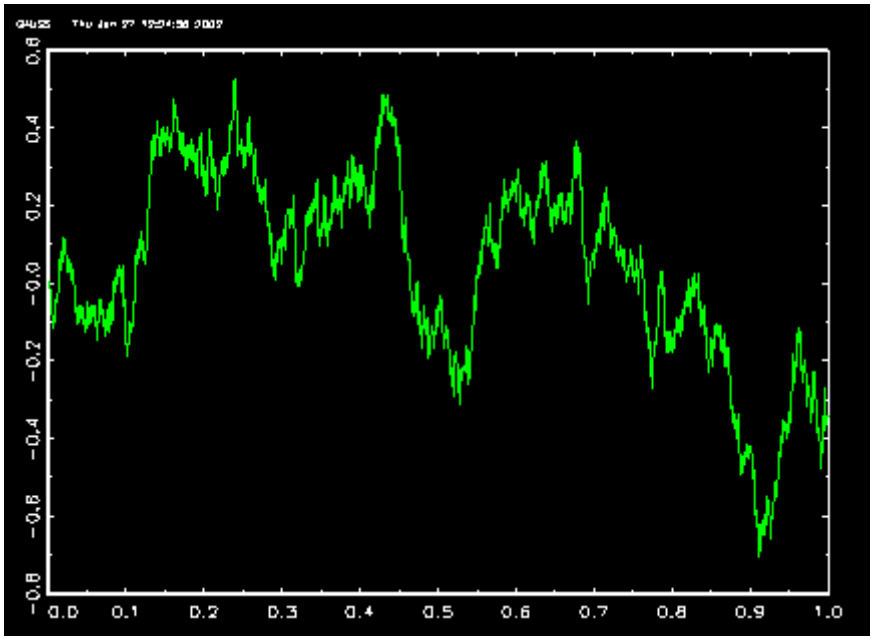
Wiener Process

ウィーナーはアメリカの学者でブラウン運動論をはじめ、現在の経済学とかかわりのある分野では時系列の平滑化に業績があるのですが、彼もまた第二次世界大戦中は、武器の自動照準の開発に深く関わりました。戦後これを応用してサイバネティクスの分野を切り開きました。冒頭で述べた２学者をはじめ、この分野に登場する人物は戦中日本人をせん滅するために最終兵器および通常兵器の武器開発に活躍していたのは興味深いことです。さて、ここでは２次元や３次元の分子や生物が動き得るディメンションではなく、経済学に関係する時系列上 dt 刻みにドリフトのない最も単純な Wiener Process を生成してみます。これは、標準正規乱数のそれぞれの時点の累積和によって実現します。ここでは、通常よくなされるように、時系列の dt の刻み幅のルートによってスケーリングしています。
プログラム

```
new; cls;
call wiener(500,1,2000);
```

```
proc wiener(w0,T,n);
    local dt,dw,w,t;
    dt=T/n;
    dw=sqrt(dt)*rndn(n,1);
    w=w0|(w0+cumsumc(dw));
    t=0|seqa(T/n,dt,n);
        library pgraph;
        graphset;
        xy(t,w);
    retp(t~w);
endp;
```

グラフ表示



プログラムでは、dw=sqrt(dt)*rndn(n,1)で計算された dw(=w(t)－w(t-1))の累積和をとったものをさらに始点の w0 の座標で上下にその分だけ平行移動させています。経済現象シミュレーションの一番単純明解なケースでは、上のプログラムの始点の w0 を現在の株価などに設定して、適当にＴとｎを設定してやることによって何期先かまでのシミュレーションがきます。さらに、この procedeure を例えば千回とか一万回繰り返してやっていろいろなケースを見ることが考えられます。上のグラフは今はそういう形状ですが、シードを設定していないので、その時々によって形状はまったく異なる結果になります。

White Noise

上の Wiener Process のより一般的な形として White Noise が考えられます。これは、少し数学的な解説をしますと、残差項の動きが

$E(\varepsilon_t)=0,\ Var(\varepsilon_t)=\sigma^2<\infty,\ Cov(\varepsilon_t,\varepsilon_{t-s})=0$

となるものを総称して一般にホワイトノイズといいます。そこで、次のような最も単純な確率微分方程式を考えます。

$\frac{dx}{dt}=\eta(t)=\frac{dW}{dt}$を考えて、両辺に$dt$をかけると

$dx=dW$となります。これを積分してやって

$\int_{ti-1}^{ti}dx=\int_{ti-1}^{ti}dW$　　さらに、これを差分近似します。

$x(t_i)-x(t_{i-1})=W(t_i)-W(t_{i-1})$となり、$dx=dW$に同じ結果です。

$x(t_i)=x(t_{i-1})+dW(t_i)$

と最終的になるものを標準正規乱数を時間刻みの dt のルートでスケーリングしたものをその都度再帰的に足していくと、下のようなプログラムになります。（ただし、これとは別に０を始点とするアルゴリズムに、始点 x0 にあわせて上下に平行移動させるところを加えています。）

プログラム(簡単な Wiener Process の場合)

```
new; cls;
call whitenoise(0,1,2000);


proc whitenoise(x0,T,n);
    local dt,x,dw,i,t;
    dt=T/n;
    x=zeros(n,1);
    dw=sqrt(dt)*rndn(n,1);
    x[1]=dw[1];
    i=2;
    do while i<=n;
        x[i]=x[i-1]+dw[i];
        i=i+1;
    endo;
    t=0|seqa(T/n,dt,n);
    x=x0|(x0+x);
        library pgraph;
        graphset;
```

```
    xy(t,x);
  retp(t~x);
endp;
```

上のプログラムは再帰的にｘを求めているのですが、基本的には、１つ前の標準正規乱数をスケーリングしたものの累積和のプログラムと同じことをフォーマルに計算しているだけです。$x(t_{i-1})$の前に何もついていない（係数が１の）ところがポイントです。

これに対して、もう一度、確率微分方程式に戻って、上のような dx/dt＝dW/dt のような単純な形ではなくて、

$$\frac{dx}{dt} = f(x(t)) + g(x(t))\frac{dW}{dt}$$

というふうにドリフトのようなものに相当するところに f(x(t))という x(t)の関数形が、いままで１であった変動のようなものに相当するところに g(x(t))という x(t)の関数形がくるような一般形を考えます。これを上と同様に dt を両辺にかけてから積分をして、さらに差分近似をして項をまとめると、今度は dt の項が新たに残ります。

$$x(t_i) = x(t_{i-1}) + f(x(t_{i-1}))dt + g(x(t_{i-1}))dW(t_i)$$

となります。例えば今、ｆとｇが実際にはそれぞれｘに定数をかけたものの関数

$$f(x(t_{i-1})) = \lambda x(t_{i-1}) \quad \text{および} \quad g(x(t_{i-1})) = \mu x(t_{i-1})$$

であると仮定（このλとμは任意の定数）して、それらを単に代入設定して、

$$x(t_i) = x(t_{i-1}) + \lambda x(t_{i-1})dt + \mu x(t_{i-1})dW(t_i)$$

となります。これを再帰的なプログラムに上のプログラムを利用して作りなおすと、次のような一般的な White Noise のプログラムになります。

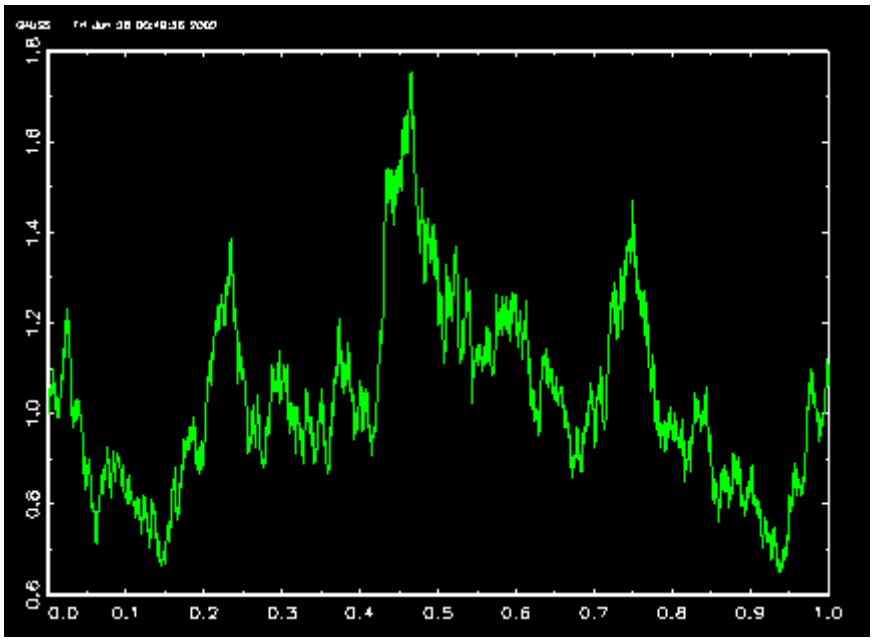
プログラム(一般的な White Noise)

```
    new; cls;
    call whitenoise(1,1,2000,2,1); @ x0 must be non-zero. @


proc whitenoise(x0,T,n,lambda,myu);
  local dt,x,dw,i,t;
  dt=T/n;
  x=zeros(n,1);
  dw=sqrt(dt)*rndn(n,1);
  x[1]=x0+dt*lambda*x0+myu*x0*dw[1];
  i=2;
  do while i<=n;
    x[i]=x[i-1]+dt*lambda*x[i-1]+myu*x[i-1]*dw[i];
    i=i+1;
  endo;
```

```
    t=0|seqa(T/n,dt,n);
    x=x0|x;
        library pgraph;
        graphset;
        xy(t,x);
    retp(t~x);
endp;
```

グラフ表示



プログラムで、x[i]=x[i-1]+dt*lambda*x[i-1]+myu*x[i-1]*dw[i]となっている部分は数式と全く同じことを意味します。先に、ループの外で x(1)を与えてやってから、ループで再帰的に計算しています。乱数生成と時間刻みのルートによるスケーリングは単純な Wiener のものと同じです。なお、このプログラムには、０期の値の平行移動のアルゴリズムは付け加えていません。右辺のすべての項に x[i-1]の項が含まれているので、ｘ0 には０がくると再帰式は０のままです。必ず０以外の数が入ります。

Geometric Brownian Motion

金融工学等でよく使われる幾何ブラウン運動をシミュレーションしてみよう。今、ｘを標準 Wiener 過程にしたがうものとすると、過程 S(t)が

$$dS(t) = \mu S(t)dt + \sigma S(t)dz$$

の関係を満たしながらｔ＝0,1,2,3,....と進んでいく。ここでν＝μ－0.5σ²として、２つの

方法でこの関係を作り出してみよう。

Method 1

$$S(t_0) = S_0$$
$$S(t_{k+1}) - S(t_k) = \mu S(t_k)\Delta t + \sigma S(t)\varepsilon(t_k)\sqrt{\Delta t}$$
ここで$\varepsilon(t_k) \sim N(0,1)$
$$S(t_{k+1}) = [1 + \mu\Delta t + \sigma\varepsilon(t_k)\sqrt{\Delta t}]S(t_k)$$

上では、１つ分の $S(t_k)$を右辺に移行しているので四角括弧の中の最初に１が来ている。この方法は簡便な方法ではありますが、幾何ブラウン運動の対象としている対数正規分布にしたがう過程を完全には作成できない欠点があります。

Method 2

$$\ln S(t_{k+1}) - \ln S(t_k) = \nu\Delta t + \sigma\varepsilon(t_k)\sqrt{\Delta t}$$
$$S(t_{k+1}) = e^{\nu\Delta t + \sigma\varepsilon(t_k)\sqrt{\Delta t}} S(t_k)$$

上では、１番目の式の左辺はログの引き算ですから $\ln S(t_{k+1})/S(t_k)$になりますから、両辺を指数 e で表して、左辺の分母の $S(t_k)$を右辺に移行すれば、２番目の式になります。この方法は、対数正規分布にしたがう過程を作成します。なお、Method 1 と Method 2 は、長い間シミュレーションを行なうと両者は一致していきます。

ここでは、初期値 S0＝10、ν＝0.15 および σ＝0.40 そして、期間の区切りは週ごと、すなわち Δt=1/52 に対して、52 期間分を２つの方法で幾何ブラウン運動をシミュレーションします。ただし、μ と ν を混同しないで、その変換には注意してください。下では、P1 と P2 をそれぞれ Method 1 と Method 2 によって求められた S(t)の値を表しています。

プログラム

```
new; cls;
S0=10;
nu=0.15;
sig=0.40;
delt=1/52;
n=52;

label1={"week" "dz" "mudt+sdz" "P1" "nudt+sdz" "P2"};
label2={"-----" "-----" "-----" "-----" "-----" "-----"};
print $label1;
print $label2;;
print/rd sim2ways(S0,nu,sig,delt,n);
```

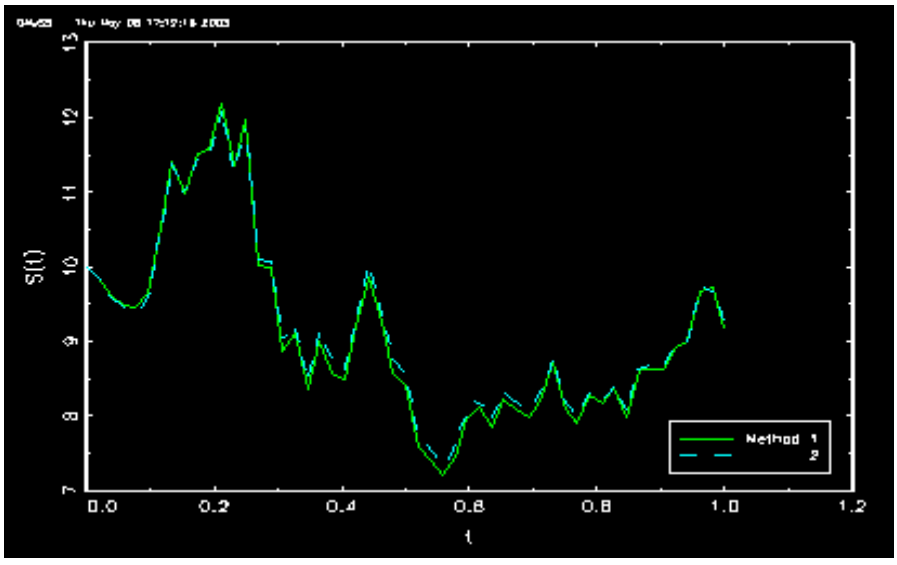
```
proc sim2ways(S0,nu,sig,delt,n);
    local e,t,mu,dz,c2,c4,St1,St2,i;
    e=rndn(n,1);
    t=seqa(0,delt,n+1);
    mu=nu+0.5*sig^2;
    dz=e*sqrt(delt);
    c2=miss(0,0)|(mu*delt+sig*dz);
    c4=miss(0,0)|(nu*delt+sig*dz);
    dz=miss(0,0)|dz;
    /* Method 1 */
    St1=zeros(n+1,1);
    St1[1]=S0;
    i=1;
    do while i<=n;
        St1[i+1]=(1+mu*delt+sig*e[i]*sqrt(delt))*St1[i];
        i=i+1;
    endo;
    /* Method 2 */
    St2=zeros(n+1,1);
    St2[1]=S0;
    i=1;
    do while i<=n;
        St2[i+1]=exp(nu*delt+sig*e[i]*sqrt(delt))*St2[i];
        i=i+1;
    endo;
    /* graph */
    library pgraph;
    graphset;
        _plegctl=1;
        _plegstr="Method 1¥000            2";
        xlabel("t");
        ylabel("S(t)");
        xy(t,St1~St2);
    retp(seqa(0,1,n+1)~dz~c2~St1~c4~St2);
endp;
```

画面表示

week	dz	mudt+sdz	P1	nudt+sdz	P2
0.00000000	.	.	10.00000000	.	10.00000000
1.00000000	-0.04649656	-0.01417555	9.85824454	-0.01571401	9.84408813
2.00000000	-0.07634035	-0.02611306	9.60081559	-0.02765152	9.57561308
3.00000000	-0.04147963	-0.01216877	9.48398544	-0.01370724	9.44525337
4.00000000	-0.02102271	-0.00398601	9.44618222	-0.00552447	9.39321725
5.00000000	0.04353781	0.02183820	9.65246985	0.02029974	9.58584566
6.00000000	0.21406651	0.09004968	10.52167169	0.08851122	10.47298223
7.00000000	0.19892945	0.08399486	11.40543798	0.08245639	11.37314886
8.00000000	-0.10219310	-0.03645416	10.98966231	-0.03799262	10.94915836
9.00000000	0.10729211	0.04733992	11.50991205	0.04580146	11.46230758
10.00000000	-0.00160756	0.00378005	11.55342011	0.00224159	11.48803019
11.00000000	0.12514541	0.05448124	12.18286478	0.05294278	12.11262649
12.00000000	-0.18237370	-0.06852640	11.34801689	-0.07006486	11.29300555
13.00000000	0.12760258	0.05546411	11.97742455	0.05392565	11.91870727
14.00000000	-0.42215265	-0.16443798	10.00788103	-0.16597644	10.09593393
15.00000000	-0.01638298	-0.00213011	9.98656309	-0.00366858	10.05896408
16.00000000	-0.29363224	-0.11302982	8.85778368	-0.11456828	8.97009176
17.00000000	0.05760971	0.02746696	9.10108009	0.02592850	9.20571425
18.00000000	-0.21379310	-0.08109416	8.36303560	-0.08263263	8.47560282
19.00000000	0.17796184	0.07560781	8.99534643	0.07406935	9.12721976
20.00000000	-0.12608713	-0.04601177	8.58145459	-0.04755023	8.70337513
21.00000000	-0.03595777	-0.00996003	8.49598305	-0.01149849	8.60387261
22.00000000	0.20293055	0.08559530	9.22319925	0.08405684	9.35835232
23.00000000	0.16320834	0.06970641	9.86611539	0.06816795	10.01853817
24.00000000	-0.16255607	-0.06059935	9.26823519	-0.06213781	9.41495497
25.00000000	-0.19850913	-0.07498057	8.57329758	-0.07651904	8.72140495
26.00000000	-0.05754113	-0.01859338	8.41389104	-0.02013184	8.54758259
27.00000000	-0.25601343	-0.09798230	7.58947868	-0.09952076	7.73788000
28.00000000	-0.06740881	-0.02254045	7.41840842	-0.02407891	7.55378557
29.00000000	-0.08242754	-0.02854794	7.20662816	-0.03008640	7.32990414
30.00000000	0.06882770	0.03195416	7.43690989	0.03041570	7.55627342
31.00000000	0.16191348	0.06918847	7.95145831	0.06765001	8.08514271
32.00000000	0.04360060	0.02186332	8.12530358	0.02032486	8.25115343

33.00000000	-0.10004252	-0.03559393	7.83609209	-0.03713239	7.95038702
34.00000000	0.11040652	0.04858569	8.21681400	0.04704722	8.33336914
35.00000000	-0.05028607	-0.01569135	8.08788109	-0.01722981	8.19101663
36.00000000	-0.04421426	-0.01326263	7.98061454	-0.01480109	8.07067347
37.00000000	0.05908851	0.02805848	8.20453847	0.02652002	8.28757126
38.00000000	0.13733726	0.05935798	8.69154331	0.05781952	8.78087861
39.00000000	-0.18800335	-0.07077826	8.07637098	-0.07231672	8.16829135
40.00000000	-0.06317679	-0.02084764	7.90799771	-0.02238610	7.98746668
41.00000000	0.10428609	0.04613751	8.27285306	0.04459905	8.35176338
42.00000000	-0.04375899	-0.01308052	8.16463985	-0.01461898	8.23055723
43.00000000	0.05194792	0.02520224	8.37040710	0.02366378	8.42764609
44.00000000	-0.12486613	-0.04552338	7.98935791	-0.04706184	8.04021372
45.00000000	0.19014079	0.08047939	8.63233659	0.07894093	8.70064009
46.00000000	-0.01073557	0.00012885	8.63344885	-0.00140961	8.68838420
47.00000000	-0.01524710	-0.00167576	8.61898125	-0.00321422	8.66050262
48.00000000	0.07823759	0.03571811	8.92683499	0.03417965	8.96163252
49.00000000	0.00587283	0.00677221	8.98728939	0.00523375	9.00865840
50.00000000	0.18080446	0.07674486	9.67701766	0.07520640	9.71229452
51.00000000	0.00025656	0.00452570	9.72081296	0.00298724	9.74135085
52.00000000	-0.15302176	-0.05678563	9.16881050	-0.05832409	9.18944654

グラフ表示



上では実線の Method 1 と破線の Method 2 はほぼ一致していますが、微妙に違うことがわかると思います。ここでは共通の標準正規乱数を各期間で使っています。