5.8　モンテカルロシミュレーション GED　　ver.0.1

ここでは、GED とｔ分布、それに今まで GARCH で扱ってきた標準正規分布 N(0,1)についての分布とその乱数過程について簡単にふれて、それらを分散１に standardized した乱数過程を GARCH 等のシミュレーションの乱数部にあてはめてみます。さらにその推定を GED およびｔ分布に対応した Log-Likelihood でそれを最大化することによって、それぞれ推定をしてみます。

GED の PDF

通常の standardized されていない GED の density は、ν＞０に対して

$$f(x)=\frac{\exp(-|x|^{\nu})}{2\Gamma(1+\frac{1}{\nu})}$$

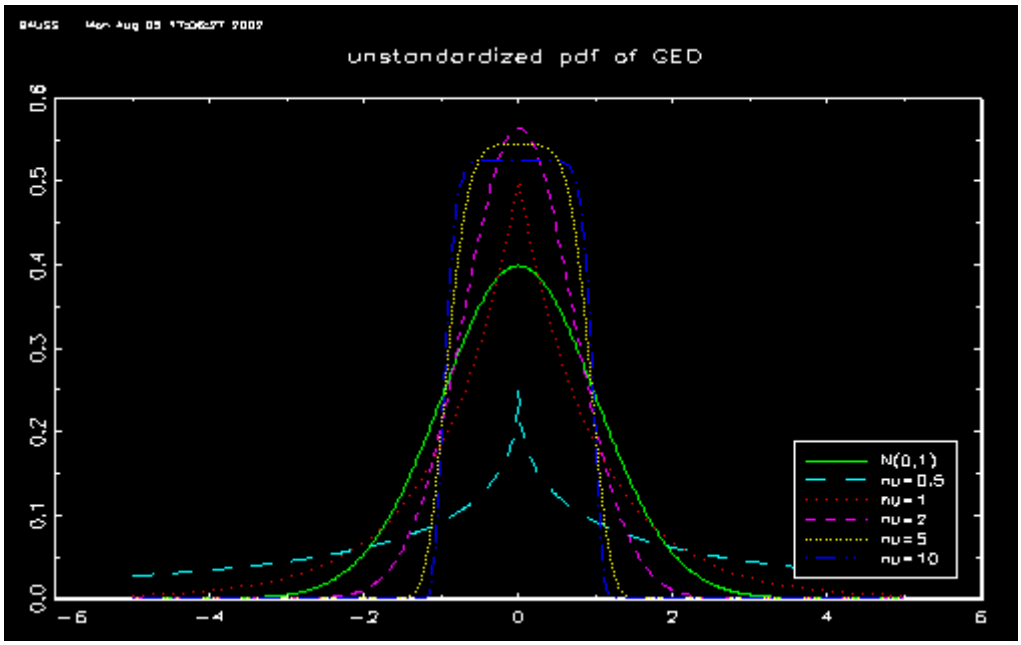
となります。これをｘとνをインプットに exp((-abs(x)^(nu)))/(2*gamma(1+(1/nu)))と procedure で書いてやってリターンで返すようにしたものを作ります。それを呼び出してνのところにいろいろな値を入れてやって PDF グラフを作成しましょう。下の場合には、ステップ幅を 0.02 に設定してやって、これをおおよそ－5 から５の範囲でｘを動かすことでグラフを描かせています。つまり、最後の端点を除く、おおよそ 10 の区間にｘの値を細かく取ることで－５からｎ(0.02)刻みで、10/n 個分のプロットを描かせます。

プログラム

```
new; cls;
library pgraph;
graphset;
n=0.02;
x=seqa(-5,n,10/n);
yn=pdfn(x); yged1=pdfged(x,0.5); yged2=pdfged(x,1);
yged3=pdfged(x,2); yged4=pdfged(x,5); yged5=pdfged(x,10);
title(" unstandardized pdf of GED");
_plegctl=1;
_plegstr="N(0,1)¥000nu=0.5¥000nu=1¥000nu=2¥000nu=5¥000nu=10";
xy(x,yn~yged1~yged2~yged3~yged4~yged5);

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

```
        end;
    endif ;
    retp( exp((-abs(x)^(nu)))/(2*gamma(1+(1/nu))) );
endp;
```

グラフ表示



上のように standardized されていない GED の PDF が各シェープパラメータνについて、標準正規分布と比べる形で書かれています。０より大きいνが大きくなるにつれて、中心がとがった形から先が丸まった形になり、さらに進むと台形のような形になります。なお統計のプログラムでよく出てくるΓ関数とは、実際には、ｘを関数の中味とする

$$\Gamma(x)=\int_0^{\infty} t^{x-1}e^{-t}dt$$

という積分値です。GAUSS では同じく gamma(x)となる。これを駆使してあらゆる density を作成することになります。今度は、分散を１に standardized した形の density を考えます。その際の PDF は、 $\nu>0$ に対して、

$$f(x)=\frac{\nu\exp(-\frac{1}{2}\left|\frac{x}{B}\right|^{\nu})}{B\Gamma(\frac{1}{\nu})2^{1+\frac{1}{\nu}}},\text{ここで}B=\left[2^{-\frac{2}{\nu}}\frac{\Gamma(\frac{1}{\nu})}{\Gamma(\frac{3}{\nu})}\right]^{\frac{1}{2}}$$

というΓ関数の複雑な形になります。すなわち、standardized されていない上のグラフの標準正規分布よりも内側にくるものはこれを横に伸ばして高さを押しつぶすことによって平たくする一方、標準正規乱数よりも外側に広がって先がとがっているものは横を圧縮して先をさらにとがらせます。これにより、分散が１になるようにします。なお、νが２の

ものが standardized された PDF では標準正規分布のものと一致します。これをプログラムに直すと、

```
    b=((2^(-2/nu))*(gamma(1/nu))/(gamma(3/nu)))^(0.5);
```

および

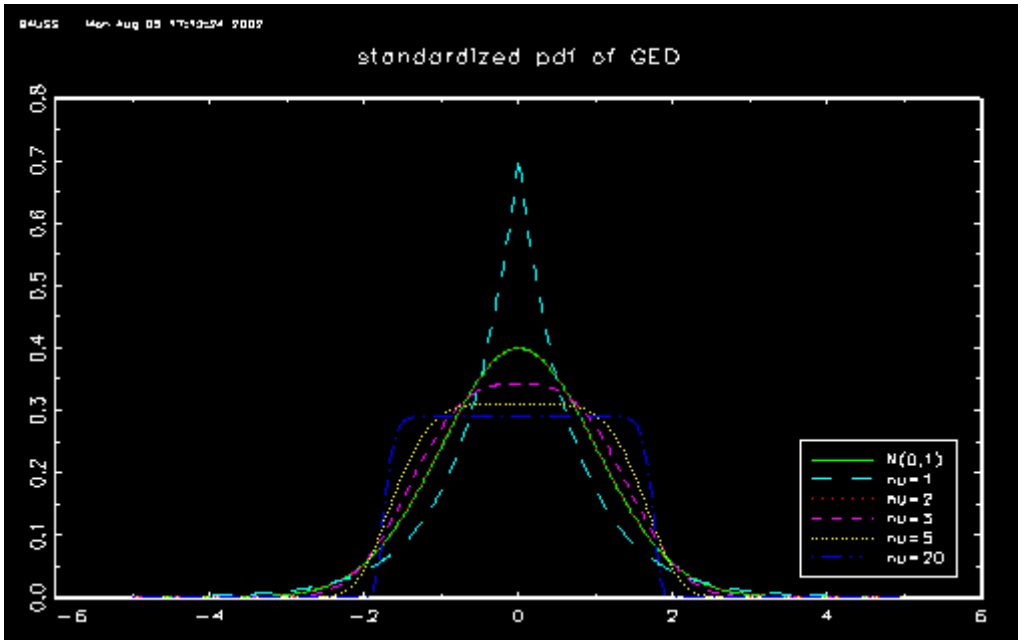
```
    nu*exp(-0.5*(abs(x/b)^(nu)))/(b*gamma(1/nu)*(2^(1+1/nu)))
```

となります。これをリターンとして上と同様のグラフを適当にνを変更して描いてみるには以下のプログラムになります。

プログラム

```
new; cls;
library pgraph;
graphset;
n=0.02;
x=seqa(-5,n,10/n);
yn=pdfn(x); yged1=pdfged(x,1); yged2=pdfged(x,2);
yged3=pdfged(x,3); yged4=pdfged(x,5); yged5=pdfged(x,20);
title("standardized pdf of GED");
_plegctl=1;
_plegstr="N(0,1)¥000nu=1¥000nu=2¥000nu=3¥000nu=5¥000nu=20";
xy(x,yn~yged1~yged2~yged3~yged4~yged5);

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



上のグラフからも明らかなように、standardized された PDF はν＝２のとき標準正規分布のものと全く同じになる。それよりもνが大きいと先が平たい分布になり究極では台形状になる。反対に、νが２よりも小さいと標準正規分布よりも先がとがった先細の分布となる。νが２のときの GED と N(0,1)の一致を数値的に確かめると次のようになる。

プログラム

```
new; cls;
print pdfged(1.5,2);
print pdfn(1.5);

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;
```

画面表示

```
        0.12951760
        0.12951760
```

上のように 1.5 のときの PDF はν＝２のときの GED も N(0,1)も全く同じ値になる。数値をかえてみても結果は同じである。

PDF of t-分布

同様に便宜上ｔ分布の PDF も作成しておこう。既に standardized されたものを示す。

プログラム

```
proc pdft(x,df);
    if df<=0;
        errorlog "ERROR: d.f. is not positive" ;
        end;
    endif;
    if df>300;
        errorlog "ERROR: d.f. is to set under 300 here." ;
        end;
    endif ;
    retp((pi*(df-2))^(-0.5)*gamma((df+1)/2)/gamma(df/2)*(1+x^2/(df-2))^(-(df+1)/2));
endp;
```

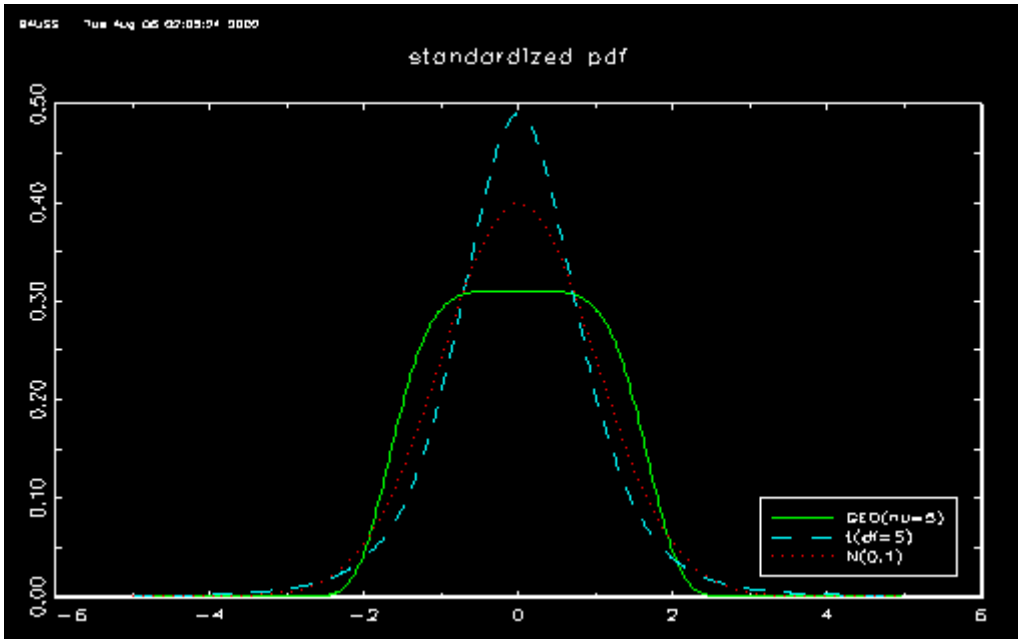
これを用いて、df＝５の場合のグラフと GED のν＝５のグラフと N(0,1)のグラフを一同に描いて比べてみよう。なお、ここで df とνは無関係である。

プログラム

```
new; cls;
library pgraph;
graphset;
n=0.02; /* Step to draw graph. */
x=seqa(-5,n,10/n); /* From -5 to 5. */
yged=pdfged(x,5);
yt=pdft(x,5);
yn=pdfn(x);
title("standardized pdf");
_plegctl=1;
_plegstr="GED(nu=5)¥000t(df=5)¥000N(0,1)"; /* ¥000 is separation. */
xy(x,yged~yt~yn);

proc pdft(x,df);
    if df<=0;
        errorlog "ERROR: d.f. is not positive" ;
```

```
        end;
    endif;
    if df>300;
        errorlog "ERROR: d.f. is to set under 300 here." ;
        end;
    endif ;
    retp((pi*(df-2))^(-0.5)*gamma((df+1)/2)/gamma(df/2)*(1+x^2/(df-2))^(-(df+1)/2));
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



上にも示したように GED は N(0,1)に比べてν＞2では台形状に先が平たくなってくるのであるが、t 分布の PDF は standardized したものではテイルが N(0,1)よりも若干分厚く中心はとがった形になる。一方、自由度 df が十分に大きいと N(0,1)にほぼ一致する。

GED 乱数

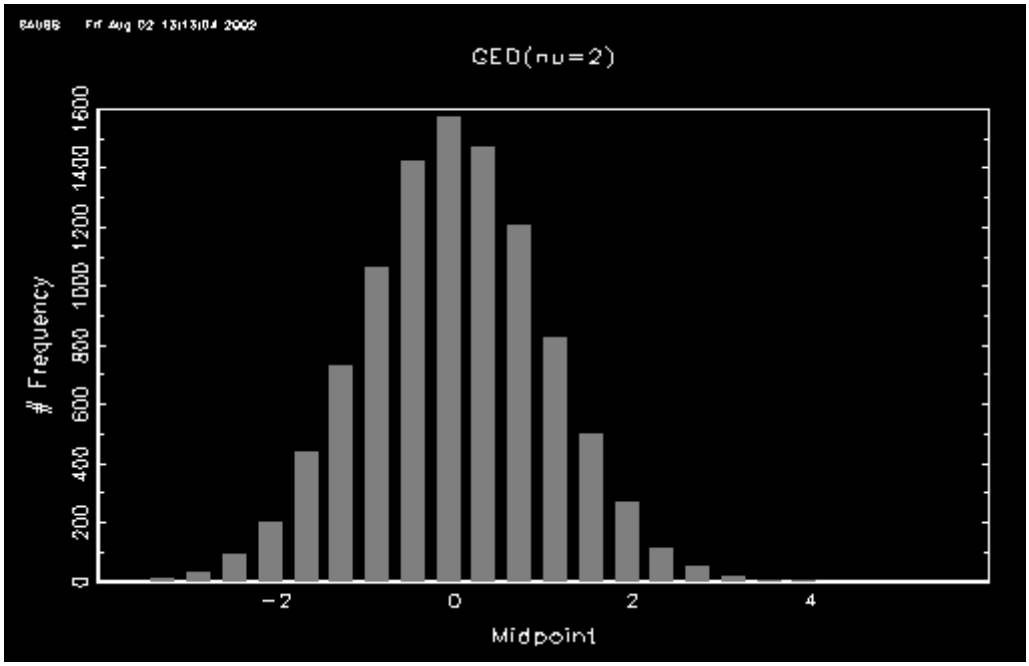
標準正規乱数をパラメタライズして伸ばしてテイルの範囲を圧縮して分厚くする簡易的な方法で乱数を作ると次のようになる。

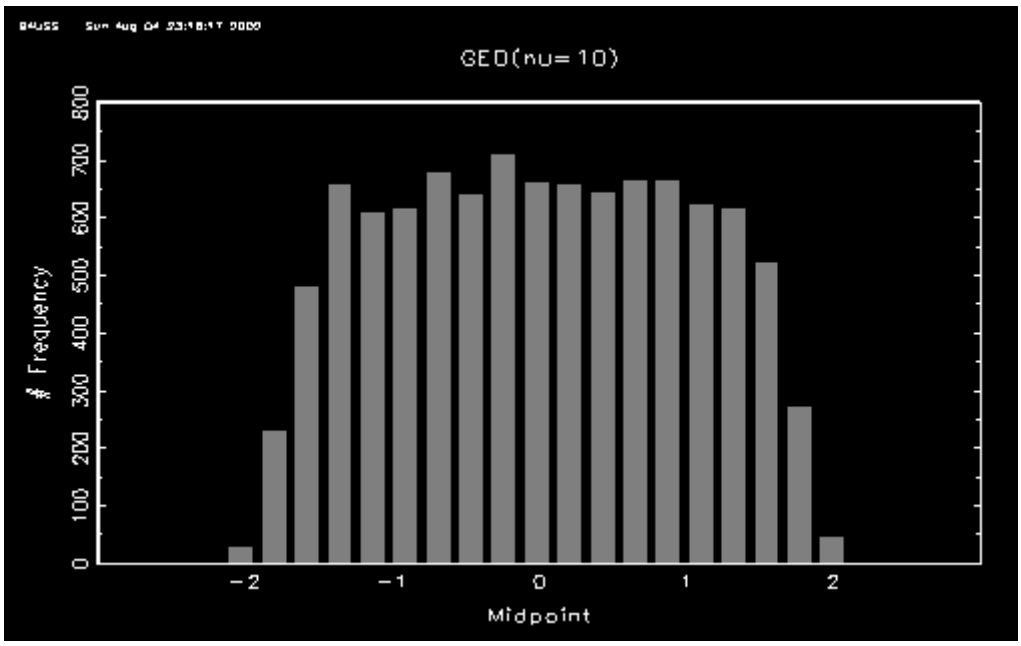
プログラム

```
new; cls;
x=rndged(10000,1,2);
library pgraph;
graphset;
title("GED(nu=2)");
hist(x,19);
print "MEAN:" meanc(x); print "    S2:" stdc(x)^2;

x=rndged(10000,1,10);
graphset;
title("GED(nu=10)");
hist(x,19);
print "MEAN:" meanc(x); print "    S2:" stdc(x)^2;

proc rndged(r,c,nu);
    local x,rnd,i,j;
    if nu<2;
        errorlog "ERROR: nu should be greater than or equal to 2: Fat tail case only" ;
        end;
    endif ;
    rnd=zeros(r,c);
    i=1;
    do while i<=r;
        j=1;
        do while j<=c;
            x=((1/nu)^(1/nu))*rndn(1,1);
            if ln(rndu(1,1))<=(-abs(x)^nu + x^2/(2*(((1/nu)^(1/nu))^2))-0.5+(1/nu));
                rnd[i,j]=(1/((gamma(3/nu)/gamma(1/nu))^(0.5)))*x;
                break;
            else;
                continue;
            endif;
```

```
            j=j+1;
        endo;
        i=i+1;
    endo;
    retp(rnd);
endp;
```

グラフ表示





画面表示(ν＝2のケース)

MEAN:　　-0.012215396

　S2:　　1.0055565

画面表示(ν=10のケース)

MEAN:　-0.0034690218

　S2:　　1.0040025

上のどちらも standardized されているので、当然のことながら、S2 はほぼ1になっている。グラフは、PDF からわかるように、ν＝2のときは標準正規分布に、νがそれよりも大きくて、ν＝10 くらいになると、ほぼ台形になる。それが乱数過程でも確かめられる。なお、上の GED 乱数発生のプログラムはν≧2だけに対応したものである。ν＜2については、先がとがるので、場合分けをして、別のプログラムにする必要がある。

t 分布乱数

　同様にして standardized されたｔ分布の乱数過程のプログラムは以下のようになる。

プログラム

```
proc rndt(r,c,df);
    local rndx2,i,x,x2;
    rndx2=zeros(r,c);
    i=1;
    do while i<=c;
        x=rndn(r,df);
        x2=x.*x;
        rndx2[.,i]=sumc(x2');
        i=i+1;
    endo;
    retp( (rndn(r,c)*sqrt(df))./(sqrt(rndx2)*sqrt(df/(df-2))) );
endp;
```

GARCH(p,q)-(N(0,1), t-Dist ,GED)シミュレーション

　いままで、NID(0,1)で標準正規乱数過程で作り出していた乱数部を自由度 df のｔ分布乱数および shape パラメータνの GED にそれぞれ取り替えることによってシミュレーションしてみよう。

プログラム

```
    new; cls;
    rndseed 777;
    a={0.2}; b={0.3,0.2,0.1}; a0=0.1; cutn=20; n=100;
```

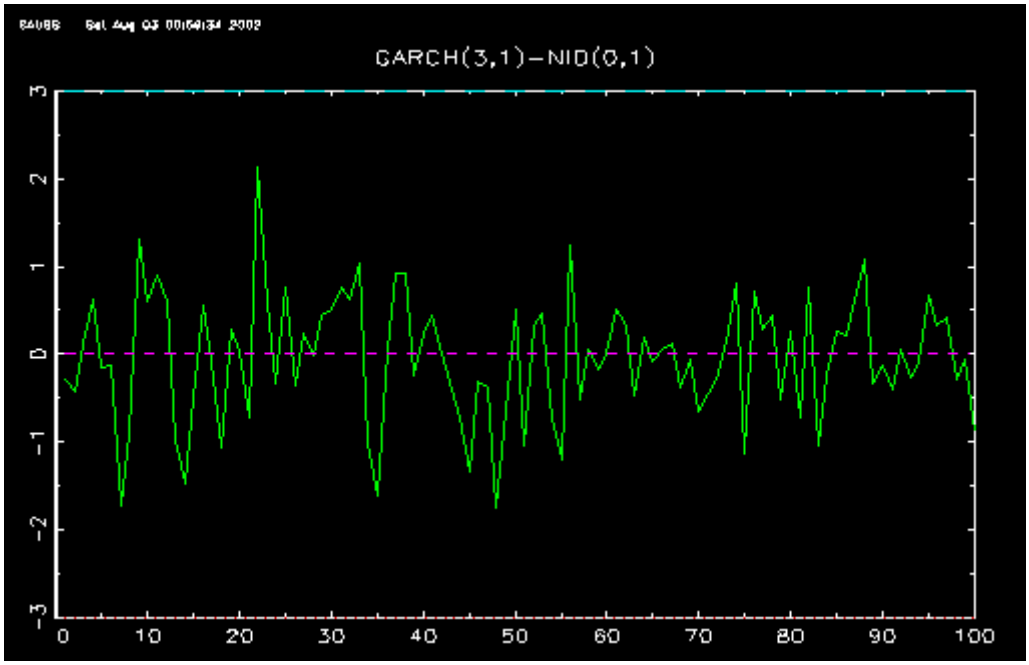
```
    opt="ged"; parameter=5;
    u=garchpq(a,b,a0,cutn,n,opt,parameter);
    library pgraph;
    graphset;
    title("GARCH(3,1)-GED(nu=5)");
    xy(seqa(1,1,n),u~(3*ones(n,1))~(-3*ones(n,1))~zeros(n,1));

proc garchpq(a,b,a0,cutn,n,opt,parameter);
    local s2,q,p,u,h,max,t;
    if sumc(a)+sumc(b)>=1;
        errorlog "ERROR:Parameter sum must be less than 1.";
        retp(-1);
    endif;
    if cutn<maxc(rows(a)|rows(b));
        errorlog "ERROR:Initial cut must be at least greater than order p or q.";
        retp(-1);
    endif;
    q=rows(a); p=rows(b);
    s2=a0/(1-sumc(a)-sumc(b));
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    max=maxc(q|p);
    u[1:max]=sqrt(s2)*rndn(max,1); h[1:max]=s2*ones(max,1);
    t=max+1;
    do while t<=cutn+n;
        h[t]=a0+rev(a)'(u[t-q:t-1]^2)+rev(b)'h[t-p:t-1];
        if      opt$=="n";
            u[t]=sqrt(h[t])*rndn(1,1);
        elseif opt$=="t";
            u[t]=sqrt(h[t])*rndt(1,1,parameter);
        elseif opt$=="ged";
            u[t]=sqrt(h[t])*rndged(1,1,parameter);
        else;
            errorlog "ERROR: Check back the option(n: N(0,1), t: t-dist, ged: GED)";
            end;
        endif;
```

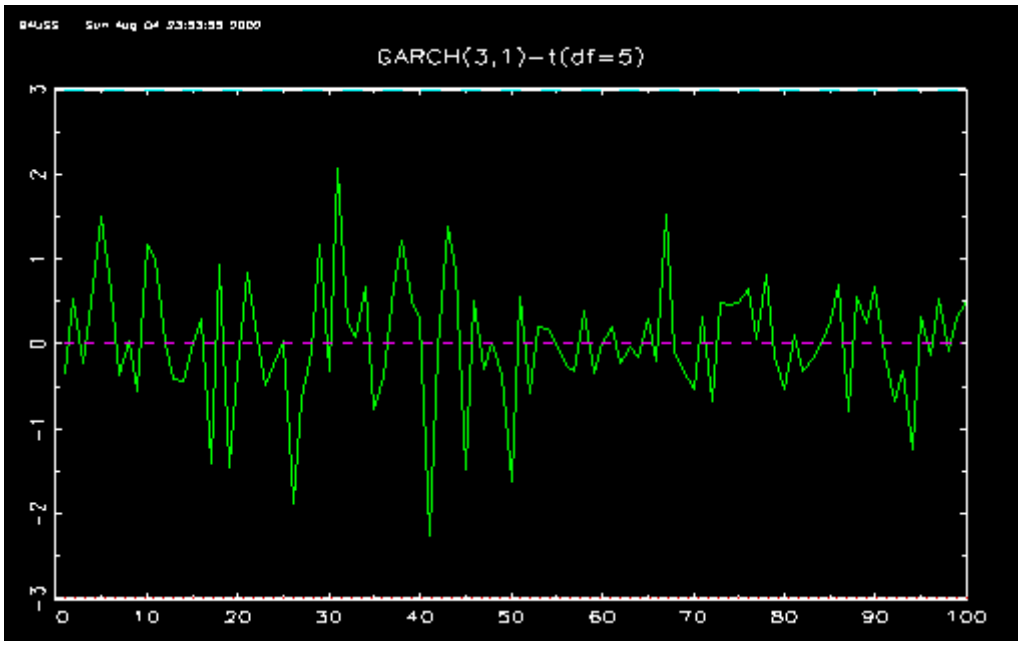
```
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;

proc rndt(r,c,df);
    local rndx2,i,x,x2;
    rndx2=zeros(r,c);
    i=1;
    do while i<=c;
        x=rndn(r,df);
        x2=x.*x;
        rndx2[.,i]=sumc(x2');
        i=i+1;
    endo;
    retp( (rndn(r,c)*sqrt(df))./(sqrt(rndx2)*sqrt(df/(df-2))) );
endp;

proc rndged(r,c,nu);
    local x,rnd,i,j;
    if nu<2;
        errorlog "ERROR: nu should be greater than or equal to 2: Fat tail case only" ;
        end;
    endif ;
    rnd=zeros(r,c);
    i=1;
    do while i<=r;
        j=1;
        do while j<=c;
            x=((1/nu)^(1/nu))*rndn(1,1);
            if ln(rndu(1,1))<=(-abs(x)^nu + x^2/(2*(((1/nu)^(1/nu))^2))-0.5+(1/nu));
                rnd[i,j]=(1/((gamma(3/nu)/gamma(1/nu))^(0.5)))*x;
                break;
            else;
                continue;
```

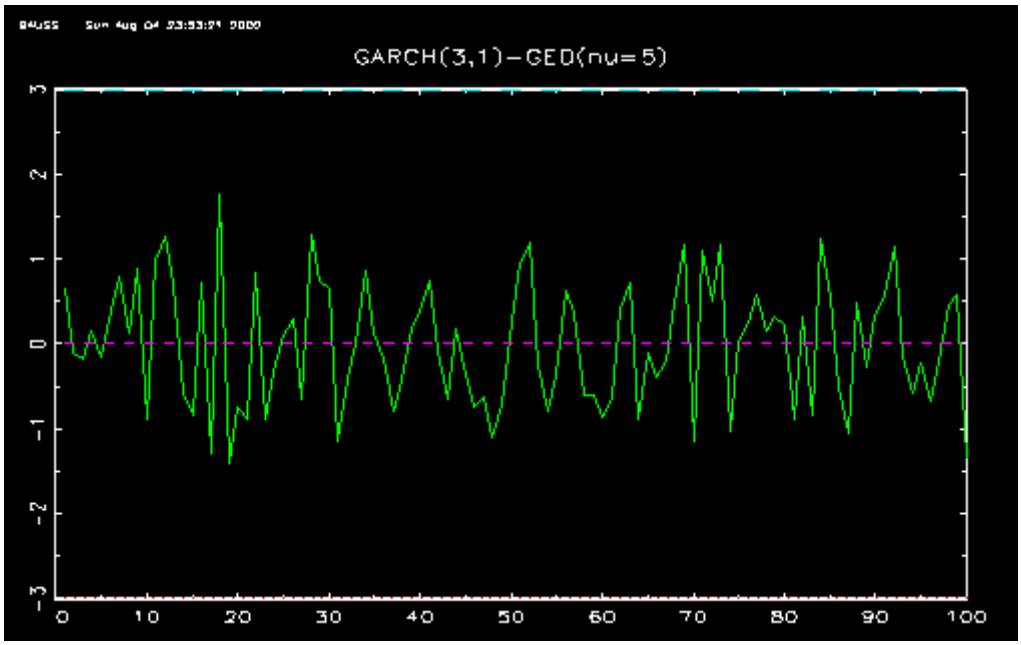
```
            endif;
            j=j+1;
        endo;
        i=i+1;
    endo;
    retp(rnd);
endp;
```

プログラム的には、GARCH のシミュレーションのプログラムをそのまま用いて、その乱数発生部を場合分けをして”ged”を入力した場合には rndged の procedure を rndn の代わりに呼び出すものにします。また、”t”と入力した場合には rndt の procedure を呼び出すようにします。これに対応してインプットは２つ増やして、opt と parameter を加えます。Opt には”n”または”t”または”ged”が入ります。それ以外であるとエラーを返してそこで計算をやめます。インプット変数 parameter には、GED の場合にはνの数値が、t の場合には自由度 df が入ります。なお、opt が”n”のときには、parameter 値は任意で無視されるようにします。上のプログラムでは、末尾に rndt と rndged の２つの procedure を置いて、上で呼び出せるようにしています。なお、言い忘れましたがΓ関数の中味の数は、正でありかつ 169 までの数でなければならないと決まっています。例えば、t 分布の自由度にとてつもない大きい数を入れるとブローアップしてしまいます。それを避けるために、技術上プログラムの一部には df を 300 くらいまでにするよう制限をかけているところがあります。

グラフ表示







同じシードに対して、GARCH 過程を描いてみると、この場合は、ｔ分布のケースでは標準正規乱数のケースよりも敏感に反応しているところは敏感に反応している一方、GED のケースでは若干狭い範囲の変動になっています。それぞれの乱数に対してシードの持つ意味は異なりますが、この場合のグラフのスケールは - 3 から 3 の間で同じにしています。

最後に、GED 乱数に対してとｔ分布に対して、GARCH と EGARCH について実際に推定をしてみましょう。これまでの GARCH 系の Log-Likelihood では、GED やｔなどの圧縮しり伸ばしたりした分布の推定には対応できません。特有の Log-Likelihood をそれぞれ使って推定することになります。

GARCH(1,1)-GED の推定

プログラム

```
new; cls;
library cml;
cmlset;
rndseed 1111;
a={0.5}; b={0.3}; a0=0.1; cutn=20; n=100; beta={0.5,2}; opt="ged"; parameter=2.5;
{y,x}=garchpqsim(a,b,a0,cutn,n,beta,opt,parameter);
data=y~x;
_cml_Bounds={ -1e256     1e256,
              -1e256     1e256,
              0.001      1e256,
              0                1,
              0                1,
              2              200};
_cml_c={0 0 0 -1 -1 0}; _cml_d=-0.9999;      /* a1+ b1<=0.9999 */
_cml_ParNames = {"const","beta","a0","a1","b1","nu"};
_cml_Algorithm=4;
start={0,0,0.1,0.1,0.1,3};
__weight=(rows(data)/(rows(data)-1))*ones(rows(data),1); __weight[1]=zeros(1,1);
{x,f,g,cov,retcode}=cml(data,0,&llged,start);
cl=cmltlimits(x,cov);
call cmlclprt(x,f,g,cl,retcode);

proc llged(b,data);
    local y,x,e2,a0,b1,a1,h,t,nu,e,bb;
    y=data[.,1]; x=data[.,2:3]; beta=b[1:2];
    e2=(y-x*beta)^2;
    e=y-x*beta;
    a0=b[3]; a1=b[4]; b1=b[5]; nu=b[6];
    h=zeros(rows(data),1);
    h[1]=a0/(1-a1-b1);
```

```
    t=2;
    do while t<=rows(data);
        h[t]=a0+a1*e2[t-1]+b1*h[t-1];
        t=t+1;
    endo;
    bb=(2^(-2/nu)*gamma(1/nu)/gamma(3/nu))^(0.5);
    retp(ln(nu)-ln(bb)-ln(2)*(1+1/nu)-ln(gamma(1/nu))-0.5*ln(h)-
0.5*((abs(e./(sqrt(h)*bb)))^nu) );
endp;

proc(2)=garchpqsim(a,b,a0,cutn,n,beta,opt,parameter);
    local s2,q,p,u,h,max,t;
    if sumc(a)+sumc(b)>=1;
        errorlog "ERROR:Parameter sum must be less than 1.";
        retp(-1);
    endif;
    if cutn<maxc(rows(a)|rows(b));
        errorlog "ERROR:Initial cut must be at least greater than order p or q.";
        retp(-1);
    endif;
    q=rows(a); p=rows(b);
    s2=a0/(1-sumc(a)-sumc(b));
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    max=maxc(q|p);
    u[1:max]=sqrt(s2)*rndn(max,1); h[1:max]=s2*ones(max,1);
    t=max+1;
    do while t<=cutn+n;
        h[t]=a0+rev(a)'(u[t-q:t-1]^2)+rev(b)'h[t-p:t-1];
        if      opt$=="n";
            u[t]=sqrt(h[t])*rndn(1,1);
        elseif opt$=="t";
            u[t]=sqrt(h[t])*rndt(1,1,parameter);
        elseif opt$=="ged";
            u[t]=sqrt(h[t])*rndged(1,1,parameter);
        else;
```

```
            errorlog "ERROR: Check back the option(n: N(0,1), t: t-dist, ged: GED)";
            end;
        endif;
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    x=ones(n,1)~(10*rndu(n,1)-5); /* Set a constant and X of [-5,5] here. */
    y=x*beta+u;
    retp(y,x);
endp;

proc rndt(r,c,df);
    local rndx2,i,x,x2;
    rndx2=zeros(r,c);
    i=1;
    do while i<=c;
        x=rndn(r,df);
        x2=x.*x;
        rndx2[.,i]=sumc(x2');
        i=i+1;
    endo;
    retp( (rndn(r,c)*sqrt(df))./(sqrt(rndx2)*sqrt(df/(df-2))) );
endp;

proc rndged(r,c,nu);
    local x,rnd,i,j;
    if nu<2;
        errorlog "ERROR: nu should be greater than or equal to 2: Fat tail case only" ;
        end;
    endif ;
    rnd=zeros(r,c);
    i=1;
    do while i<=r;
        j=1;
        do while j<=c;
            x=((1/nu)^(1/nu))*rndn(1,1);
```

```
            if ln(rndu(1,1))<=(-abs(x)^nu + x^2/(2*(((1/nu)^(1/nu))^2))-0.5+(1/nu));
                rnd[i,j]=(1/((gamma(3/nu)/gamma(1/nu))^(0.5)))*x;
                break;
            else;
                continue;
            endif;
            j=j+1;
        endo;
        i=i+1;
    endo;
    retp(rnd);
endp;
```

画面表示

```
Mean log-likelihood         -1.12000
Number of cases      100
                                0.95 confidence limits
Parameters     Estimates      Lower Limit    Upper Limit    Gradient
-------------------------------------------------------------------
const             0.4869         0.3843         0.5895      -0.0000
beta              1.9676         1.9378         1.9975       0.0000
a0                0.0749         0.0056         0.1443      -0.0000
a1                0.6005         0.3299         0.8711       0.0000
b1                0.3182         0.1015         0.5348      -0.0000
nu                2.3372         1.1492         3.5252      -0.0000
Number of iterations     426
Minutes to convergence      1.24950
```

GARCH(1,1)-ｔの推定

プログラム

```
    new; cls;
    library cml;
    cmlset;
    rndseed 777;
    a={0.5}; b={0.3}; a0=0.1; cutn=20; n=100; beta={0.5,2}; opt="t"; parameter=5;
    {y,x}=garchpqsim(a,b,a0,cutn,n,beta,opt,parameter);
    data=y~x;
```

```
    _cml_Bounds={ -1e256     1e256,
                  -1e256     1e256,
                  0.001      1e256,
                  0                1,
                  0                1,
                  2.001        200};
    _cml_c={0 0 0 -1 -1 0}; _cml_d=-0.9999;      /* a1+ b1<=0.9999 */
    _cml_ParNames = {"const","beta","a0","a1","b1","nu"};
    _cml_Algorithm=4;
    start={0,0,0.1,0.1,0.1,3};
    __weight=(rows(data)/(rows(data)-1))*ones(rows(data),1); __weight[1]=zeros(1,1);
    {x,f,g,cov,retcode}=cml(data,0,&llt,start);
    cl=cmltlimits(x,cov);
    call cmlclprt(x,f,g,cl,retcode);

proc llt(b,data);
    local y,x,e2,a0,b1,a1,h,t,nu;
    y=data[.,1]; x=data[.,2:3]; beta=b[1:2];
    e2=(y-x*beta)^2;
    a0=b[3]; a1=b[4]; b1=b[5]; nu=b[6];
    h=zeros(rows(data),1);
    h[1]=a0/(1-a1-b1);
    t=2;
    do while t<=rows(data);
        h[t]=a0+a1*e2[t-1]+b1*h[t-1];
        t=t+1;
    endo;
    retp(ln(gamma(0.5*(nu+1)))-ln(gamma(nu/2))-0.5*ln(pi*(nu-2))-0.5*ln(h)-
((nu+1)/2)*(ln(1+e2./(h*(nu-2)))) );
endp;

proc(2)=garchpqsim(a,b,a0,cutn,n,beta,opt,parameter);
    local s2,q,p,u,h,max,t;
    if sumc(a)+sumc(b)>=1;
        errorlog "ERROR:Parameter sum must be less than 1.";
        retp(-1);
```

```
    endif;
    if cutn<maxc(rows(a)|rows(b));
        errorlog "ERROR:Initial cut must be at least greater than order p or q.";
        retp(-1);
    endif;
    q=rows(a); p=rows(b);
    s2=a0/(1-sumc(a)-sumc(b));
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    max=maxc(q|p);
    u[1:max]=sqrt(s2)*rndn(max,1); h[1:max]=s2*ones(max,1);
    t=max+1;
    do while t<=cutn+n;
        h[t]=a0+rev(a)'(u[t-q:t-1]^2)+rev(b)'h[t-p:t-1];
        if      opt$=="n";
            u[t]=sqrt(h[t])*rndn(1,1);
        elseif opt$=="t";
            u[t]=sqrt(h[t])*rndt(1,1,parameter);
        elseif opt$=="ged";
            u[t]=sqrt(h[t])*rndged(1,1,parameter);
        else;
            errorlog "ERROR: Check back the option(n: N(0,1), t: t-dist, ged: GED)";
            end;
        endif;
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    x=ones(n,1)~(10*rndu(n,1)-5); /* Set a constant and X of [-5,5] here. */
    y=x*beta+u;
    retp(y,x);
endp;

proc rndt(r,c,df);
    local rndx2,i,x,x2;
    rndx2=zeros(r,c);
    i=1;
```

```
    do while i<=c;
        x=rndn(r,df);
        x2=x.*x;
        rndx2[.,i]=sumc(x2');
        i=i+1;
    endo;
    retp( (rndn(r,c)*sqrt(df))./(sqrt(rndx2)*sqrt(df/(df-2))) );
endp;

proc rndged(r,c,nu);
    local x,rnd,i,j;
    if nu<2;
        errorlog "ERROR: nu should be greater than or equal to 2: Fat tail case only" ;
        end;
    endif ;
    rnd=zeros(r,c);
    i=1;
    do while i<=r;
        j=1;
        do while j<=c;
            x=((1/nu)^(1/nu))*rndn(1,1);
            if ln(rndu(1,1))<=(-abs(x)^nu + x^2/(2*(((1/nu)^(1/nu))^2))-0.5+(1/nu));
                rnd[i,j]=(1/((gamma(3/nu)/gamma(1/nu))^(0.5)))*x;
                break;
            else;
                continue;
            endif;
            j=j+1;
        endo;
        i=i+1;
    endo;
    retp(rnd);
endp;
```

画面表示

```
Mean log-likelihood        -0.803731
Number of cases      100
                                0.95 confidence limits
Parameters     Estimates      Lower Limit    Upper Limit    Gradient
-----------------------------------------------------------------
const             0.5660         0.4729         0.6592      -0.0000
beta              1.9755         1.9419         2.0092       0.0000
a0                0.0889        -0.0217         0.1995       0.0000
a1                0.3702        -0.0958         0.8361       0.0000
b1                0.4341         0.0240         0.8442       0.0000
nu                5.2876        -0.9156        11.4907      -0.0000
Number of iterations     56
Minutes to convergence      0.15850
```

EGARCH(1,1)-GED の推定

プログラム

```
new; cls;
library cml;
cmlset;
rndseed 777;
a=0.45; b=0.5; a0=0.2; cutn=20; n=100; beta={0.5,2}; opt="ged"; parameter=3;
{y,x}=egarchsim(a,b,a0,cutn,n,beta,opt,parameter);
data=y~x;
_cml_Bounds={ -1e256     1e256,
              -1e256     1e256,
              0.00001    1e256,
              0                1,
              0                1,
              2              200};
_cml_c={0 0 0 -1 -1 0}; _cml_d=-0.9999;
_cml_ParNames = {"const","beta","a0","a1","b1","nu"};
_cml_Algorithm=4;
start={0,0,0.1,0.1,0.1,3};
__weight=(rows(data)/(rows(data)-1))*ones(rows(data),1); __weight[1]=zeros(1,1);
{x,f,g,cov,retcode}=cml(data,0,&llged,start);
```

```
cl=cmltlimits(x,cov);
call cmlclprt(x,f,g,cl,retcode);

proc llged(b,data);
    local y,x,e2,a0,a1,b1,h,t,nu,bb,e;
    y=data[.,1]; x=data[.,2:3]; beta=b[1:2];
    e2=(y-x*beta)^2;
    e=(y-x*beta);
    a0=b[3]; a1=b[4]; b1=b[5]; nu=b[6];
    h=zeros(rows(data),1);
    h[1]=exp(a0/(1-a1-b1));
    t=2;
    do while t<=rows(data);
        h[t]=exp(a0+a1*sqrt(e2[t-1])/sqrt(h[t-1])+b1*ln(h[t-1]));
        t=t+1;
    endo;
    bb=(2^(-2/nu)*gamma(1/nu)/gamma(3/nu))^(0.5);
    retp(ln(nu)-ln(bb)-ln(2)*(1+1/nu)-ln(gamma(1/nu))-0.5*ln(h)-
0.5*((abs(e./(sqrt(h)*bb)))^nu) );
endp;

proc(2)=egarchsim(a1,b1,a0,cutn,n,beta,opt,parameter);
    local s2,u,h,t,x,y;
    s2=a0/(1-a1-b1);
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    u[1]=exp(ln(s2)/2)*rndn(1,1); h[1]=exp(s2);
    t=2;
    do while t<=cutn+n;
        h[t]=exp(a0+a1*abs(u[t-1])/sqrt(h[t-1])+b1*ln(h[t-1]));
        if      opt$=="n";
            u[t]=sqrt(h[t])*rndn(1,1);
        elseif opt$=="t";
            u[t]=sqrt(h[t])*rndt(1,1,parameter);
        elseif opt$=="ged";
            u[t]=sqrt(h[t])*rndged(1,1,parameter);
```

```
        else;
            errorlog "ERROR: Check back the option(n: N(0,1), t: t-dist, ged: GED)";
            end;
        endif;
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    h=h[cutn+1:cutn+n];
    x=ones(n,1)~(10*rndu(n,1)-5); /* Set a constant and X of [-5,5] here. */
    y=x*beta+u;
    retp(y,x);
endp;

proc rndt(r,c,df);
    local rndx2,i,x,x2;
    rndx2=zeros(r,c);
    i=1;
    do while i<=c;
        x=rndn(r,df);
        x2=x.*x;
        rndx2[.,i]=sumc(x2');
        i=i+1;
    endo;
    retp( (rndn(r,c)*sqrt(df))./(sqrt(rndx2)*sqrt(df/(df-2))) );
endp;

proc rndged(r,c,nu);
    local x,rnd,i,j;
    if nu<2;
        errorlog "ERROR: nu should be greater than or equal to 2: Fat tail case only" ;
        end;
    endif ;
    rnd=zeros(r,c);
    i=1;
    do while i<=r;
        j=1;
```

```
        do while j<=c;
            x=((1/nu)^(1/nu))*rndn(1,1);
            if ln(rndu(1,1))<=(-abs(x)^nu + x^2/(2*(((1/nu)^(1/nu))^2))-0.5+(1/nu));
                rnd[i,j]=(1/((gamma(3/nu)/gamma(1/nu))^(0.5)))*x;
                break;
            else;
                continue;
            endif;
            j=j+1;
        endo;
        i=i+1;
    endo;
    retp(rnd);
endp;
```

画面表示

```
Mean log-likelihood         -2.00715
Number of cases      100
                                0.95 confidence limits
Parameters     Estimates      Lower Limit    Upper Limit    Gradient
-------------------------------------------------------------------
const             0.5891         0.3182         0.8600       0.0000
beta              1.9012         1.7994         2.0030      -0.0000
a0                0.2717        -0.0111         0.5545       0.0000
a1                0.4443         0.0328         0.8558      -0.0000
b1                0.4901         0.0983         0.8818      -0.0000
nu                4.8239         1.0155         8.6323       0.0000
Number of iterations     168
Minutes to convergence      0.56117
```