## 5.6　モンテカルロシミュレーション ARCH Family(1)　　ver.0.1

ここでは、ARCH および GRACH に属するモデルのシミュレーションをプログラムします。列ごとの処理に優れた GAUSS においては、少なくとも、乱数設定から ARCH ファミリーのシミュレーションに持ちこむことは非常に簡単である。ARCH モデルの特質である過去の分散が大きいと当期の分散も大きい、過去の分散が小さいと当期の分散も小さくなるという自己回帰の不均一分散過程になっていることに十分に注意して、その初期値の影響を取り除くべくプログラムすることにしよう。

ARCH(q)過程

不均一分散の特殊な形として、過去の分散の変動を加味したモデルとしてこの ARCH モデルがある。すなわち、

$$Y = X\beta + u$$

$$h_t = a_0 + a_1 u_{t-1}^2 + \ldots + a_q u_{t-q}^2$$

$$u_t = \sqrt{h_t}\varepsilon_t \text{ここで} \varepsilon_t \sim NID(0,1)$$

ただし、$a_0 > 0$, $a_1, \ldots a_q \geqq 0$

であるから、まずｈを求めてから、そのルートをとったものに N(0,1)の標準正規乱数をかければｕが求まることになる。ここで、さらに定常において $var(u_t)=var(u_{t-1})$であるから

$$\text{unconditional variance は } s2 = \mathrm{var}(u_t) = \frac{a_0}{1-\sum_{k=1}^{q} a_k}, (q=1\text{の場合} s^2 = \frac{a_0}{1-a_1})$$

となる。これらの関係を使って、パラメータと s2 がわかっている場合の ARCH 過程をシミュレーションしてみよう。

**ARCH(1)のケース**

ここでは、ｑ＝１期前までの分散を考える ARCH(1)を扱う。すなわち、パラメータは次のように $a_0$ と $a_1$ の２つである。

$$h_t = a_0 + a_1 u_{t-1}^2$$

$$u_t = \sqrt{h_t}\varepsilon_t \text{ここで} \varepsilon_t \sim NID(0,1)$$

このうち、$a_0$ については、unconditional variance を設定すると一意的に決まってくる。

```
a0=(1-a1)*s2;
```

とﾌﾟﾛｸﾞﾗﾑ中は設定する。この関係をもとに、初期値 u[1]を定めた上で

```
h[t]=a0+a1*u[t-1]^2;
```

を繰り返し計算をする。また、εｔ～NID(0,1)の部分は標準正規乱数 rndn(1,1)を使って

```
u[t]=sqrt(h[t])*rndn(1,1);
```

を h[t]の値を代入して再帰的にｔが２からｎまで求めていく。ARCH(１)のシミュレーションの骨格は以上のようなものであるが、問題は初期値 u[1]をどうするかにある。ARCH 属の基本性質である「過去の分散が小さければ当期の分散も小さく、過去の分散が大きければ当期の分散も大きくなる」という性質が、無闇に u[0]=0 などと設定すると、如実に反映されてしまう。そこで、u[1]=の式の h[t-1]の部分には定常状態の unconditional variance の s2 を代入して、

```
u[1]=sqrt(s2)*rndn(1,1);
```

とする。なお、h も u も最初にゼロで領域を確保する必要があるが、h[1]の方は GARCH ではないので h[t-q]の項は存在しないので設定は不要である。この定常状態の設定に加えてさらに、この初期値の影響をシミュレーションに極力小さくするために、初期値を大幅にいくらかカットしてやる。例えば 100 個のシミュレーション系列がほしくて初期値を 10 個切り取るのならば、最初から 110 個のシミュレーションをするつもりで計算をしてやり、最初の 10 個を切り取り残りの 100 個を採用する。下のプログラムでは、cutn にその切り取る初期のデータ数、n に最終的にほしいデータ系列の数を設定してやって、まずｈと u を cutn＋n 個分領域をゼロで確保してやってから、その数だけ ARCH の繰り返し計算をする。そして、最終的にできあがった cutn＋n 個の系列のうち最初の cutn を切り取るため

```
u=u[cutn+1:cutn+n];
```

として、cutn＋n－(cutn＋１)＋１＝ｎ個だけ残して、それをリターンにしている。

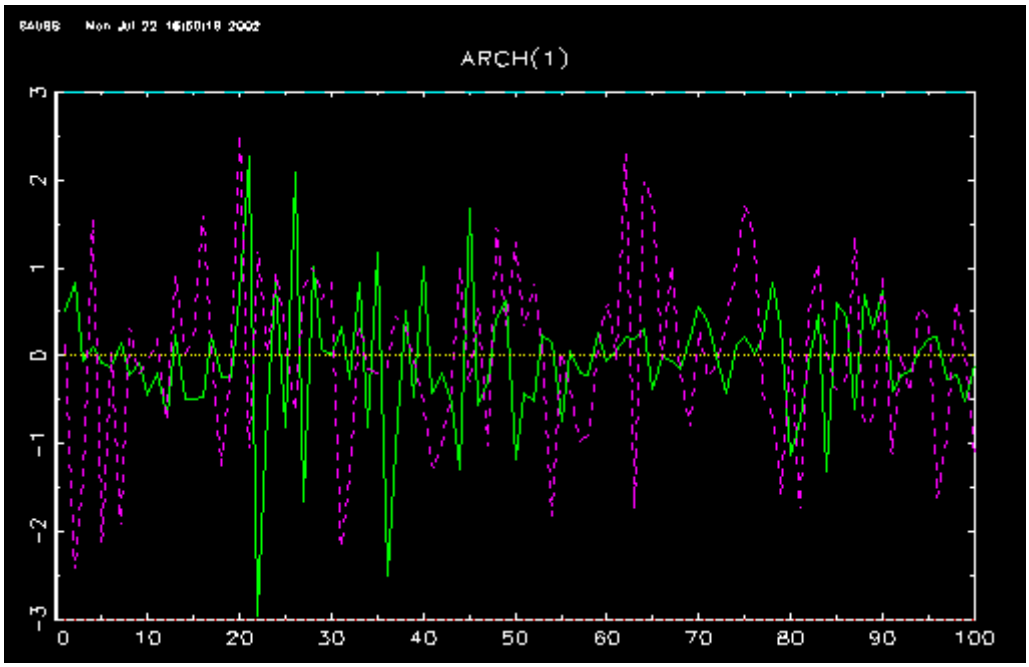
プログラム

```
new; cls;
a1=0.9; s2=1; cutn=10; n=100;
u=arch1(a1,s2,cutn,n);
library pgraph;
graphset;
title("ARCH(1)");
xy(seqa(1,1,n),u~(3*ones(n,1))~(-3*ones(n,1))~(sqrt(s2)*rndn(n,1))~zeros(n,1));

proc arch1(a1,s2,cutn,n);
local a0,u,h,t;
a0=(1-a1)*s2;
u=zeros(cutn+n,1);
h=zeros(cutn+n,1);
u[1]=sqrt(s2)*rndn(1,1);
t=2;
do while t<=cutn+n;
```

```
        h[t]=a0+a1*u[t-1]^2;
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

グラフ表示




上のプログラムでは、a1=0.9; s2=1; cutn=10; n=100;というふうにパラメータを設定してやってから、arch1(a1,s2,cutn,n)を呼び出し、そのリターンを u と置く。ここで、パラメータ a1 は１よりも小さいことが必要である。また s2 は分散であるから当然マイナスにはならない。これを（別系列ではあるが参考となる）分散が s2 である正規乱数のと一緒にグラフ表示させている。その際わかりやすいように-3 と３のラインも加えている。

ARCH(２)のケース

ここでは ARCH(1)と全く同様に初期値を設定してから再帰式を繰り返し計算させる。ただし、初期値は u[1]と u[2]の２つになる。ここではどちらも s2 に設定してあるが、別な方法に、h[1]=a0; および h[2]=a0+a1*u[1]としてやってもできる。（なお、後半で行なう推定については現実のデータの u にもとづいて計算するのでこの h[1]、h[2]、…の設定の方法を採用している。）

プログラム

```
    new; cls;
    a1=0.6; a2=0.3 s2=1; cutn=10; n=100;
    u=arch2(a1,a2,s2,cutn,n);
    library pgraph;
    graphset;
    title("ARCH(2)");
    xy(seqa(1,1,n),u~(3*ones(n,1))~(-3*ones(n,1))~(sqrt(s2)*rndn(n,1))~zeros(n,1));

proc arch2(a1,a2,s2,cutn,n);
  local a0,u,h,t;
  a0=(1-a1-a2)*s2;
  u=zeros(cutn+n,1);
  h=zeros(cutn+n,1);
  u[1]=sqrt(s2)*rndn(1,1); u[2]=sqrt(s2)*rndn(1,1);
  t=3;
  do while t<=cutn+n;
    h[t]=a0+a1*u[t-1]^2+a2*u[t-2]^2;
    u[t]=sqrt(h[t])*rndn(1,1);
    t=t+1;
  endo;
  u=u[cutn+1:cutn+n];
  retp(u);
endp;
```

ARCH(ｑ)の一般形

ARCH(1)および ARCH(2)のケースを一般化してｑ次までのケースをプログラムしてみます。骨格となるところの変更点は、まず初期値をｑ個確保する必要があるため、

```
u[1:q]=sqrt(s2)*rndn(q,1);
```

とします。その後で、h[t]を求めてからそれを u[t]=sqrt(h[t])*rndn(1,1);に代入して u[t]を求めていくことになります。そこで、

```
h[t]=a0+a1*u[t-1]^2+a2*u[t-2]^2 ．．． aq-1*u[t-q+1]^2+aq*u[t-q];
```

とでもなるのでしょうが、この方法では u[t-1]から u[t-q]を求めて各係数をかけるのは容易ではありません。そこで、

```
h[t]=a0+rev(a)'(u[t-q:t-1]^2);
```

という具合に、列の古い上から順番に素直に、逆方向にかけ合わせることを考えます。すなわち、(u[t-q:t-1]^2)のところは、ｑ×１の列で上から順番に、

u[t-q]^2
u[t-q+1]^2
.
.
u[t-2]^2
u[t-1]^2

となっています。これに１×ｑの（aq, aq-1, .　.　.　, a2, a1）を前からかけてやれば、(1×ｑ)×(q×１)＝１×１ですべての要素がかけ合わさり、その和が計算できます。これにa0 をさらに加えれば h[t]が計算できることになります。なお、順番を逆にする関数 rev を使って列ベクトル a の中味の要素の順番を逆にして（aq, aq-1, .　.　.　, a2, a1）とします。これは列ですから行に直すために転置して rev(a)'をかけてやります。下のプログラムでは、この他に、procedure の冒頭で、a の列に入っているパラメータの合計が１を超えないようにエラールーチンをつけてやっていることと、初期値カットの数 cutn がパラメータの数よりも下回らないようにエラーを考えています。（もちろん、数倍程度、大幅に上回っていることが必要です。）

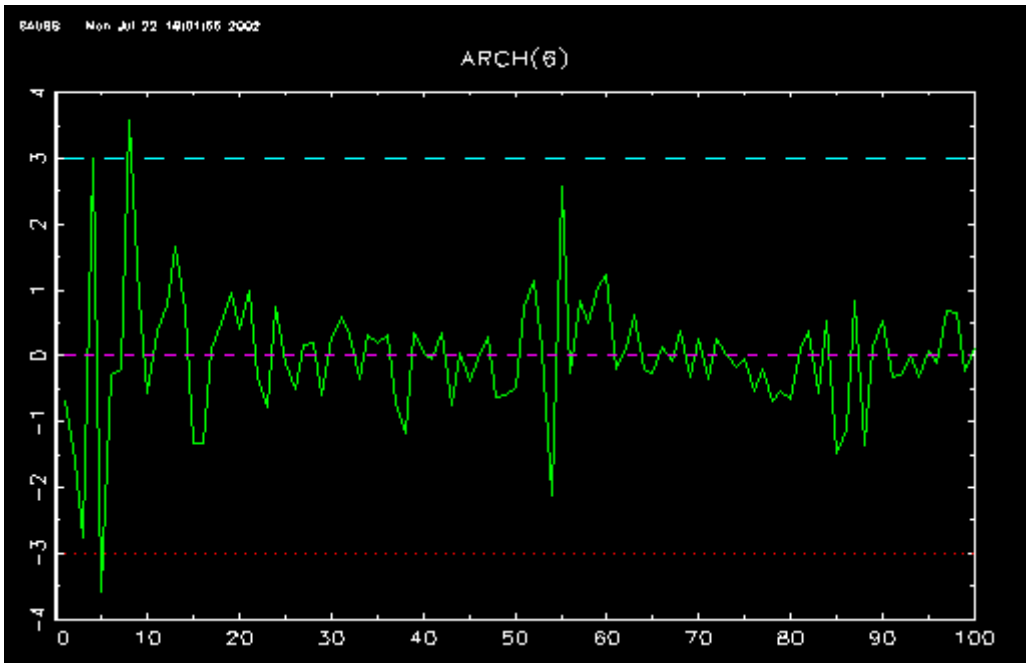
プログラム

```
new; cls;
a={0.3,0.2,0.1,0.1,0.1,0.1}; s2=1; cutn=10; n=100;
u=archq(a,s2,cutn,n);
library pgraph;
graphset;
title("ARCH(6)");
xy(seqa(1,1,n),u~(3*ones(n,1))~(-3*ones(n,1))~zeros(n,1));

proc archq(a,s2,cutn,n);
    local a0,q,u,h,t;
    if sumc(a)>=1;
        errorlog "ERROR:Parameter sum must be less than 1.";
        retp(-1);
    endif;
    if cutn<rows(a);
        errorlog "ERROR:Initial cut must be at least greater than order q.";
        retp(-1);
    endif;
    q=rows(a);
    a0=(1-sumc(a))*s2;
```

```
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    u[1:q]=sqrt(s2)*rndn(q,1);
    t=q+1;
    do while t<=cutn+n;
        h[t]=a0+rev(a)'(u[t-q:t-1]^2);
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

グラフ表示




aのところに６×１の６つのパラメータを設定してやって、ARCH(6)を描かせたのが上のプログラムです。プラスマイナス３を超えるところが発生しており、分散が大きくなればしばらくはその状態が続き、小さくなればまたその状態がしばらく続くといるARCH独特の動きをよく表せていると思います。技術的に、上のプログラムでは、u[1]からu[q]までの残差をどのように設定するかと初期カットの数cutnをどのように設定するかで理想的にシミュレーションができるかが決まります。なおcutnの数はデータの系列の数ｎとは無関係に理論上無限大まで設定できますから問題はありません。問題は、ｑ＋１期から始まる再帰過程の初期のu[1]からu[q]に何を入れるかで若干の影響が出てきます。ここではunconditional varianceのs2をｈとしたsqrt(s2)*N(0,1)の値を設定しています。工夫がある

とすれば、ここを変更することになります。ただし、cutnの値を可能な限り十分に大きくすれば、事実上u[1]からu[q]に入る数は理論的な範囲におさまっているならば何であってもかまいません。ＭＡやＡＲ過程でやったように０を設定しても問題はないでしょう。

**パラメータa0を設定する方法**

なお、unconditional variance s2は、aのベクトルとa0によって一意的に決まってきますから、s2の代わりにa0をprocのインプットとして代入してARCH過程を生成するには、下のように、インプットにa0をもってきて、s2=a0/(1-sumc(a));とすることになります。

プログラム

```
proc archq(a,a0,cutn,n);
    local s2,q,u,h,t;
    if sumc(a)>=1;
        errorlog "ERROR:Parameter sum must be less than 1.";
        retp(-1);
    endif;
    if cutn<rows(a);
        errorlog "ERROR:Initial cut must be at least greater than order q.";
        retp(-1);
    endif;
    q=rows(a);
    s2=a0/(1-sumc(a));
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    u[1:q]=sqrt(s2)*rndn(q,1);
    t=q+1;
    do while t<=cutn+n;
        h[t]=a0+rev(a)'(u[t-q:t-1]^2);
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

これにより、aのパラメータを設定することによりシミュレーションができるようになります。なお、unconditional varianceは１からa1からaqまでのパラメータ合計を引いた残りの数に対するa0の割合で表せますから、この割合が大きくなるとs2は結果的に小さくなり、

反対に引いた残りが大きく相対的にa0が小さい場合にはs2は結果的に大きくなり、シミュレーションに反映されます。

GARCH(p,ｑ)過程

ここでは、いままで「ｈに対して」過去の分散に関係があったMA(q)のような過程であったのですが、さらにこれにｈ自身のAR過程を加えます。すなわち、

$$h_t = a_0 + a_1 u_{t-1}^2 + ... + a_q u_{t-q}^2 + b_1 h_{t-1} + ... + b_p h_{t-p}$$

$$u_t = \sqrt{h_t}\varepsilon_t \text{ここで} \varepsilon_t \sim NID(0,1)$$

を考えます。プログラムはARCHの時と全く同様です。ただし、h[1],...などの初期値を設定する必要が新たに出てきます。ｈの再帰式はh[t-1]...のタームが新たに加わります。

GARCH(1,1)のケース

ここではh[t-1]の項を再帰式にどう加えるかがARCH(1)からの主な変更点となります。定常状態を初期には考えて、u[1]の設定に加えて、新たにh[1]=s2と設定します。これら２つの初期値をもとにして、h[t]=a0+a1*u[t-1]^2+b1*h[t-1];の再帰式でh[t]を生成してからu[t]の式に代入します。これをt=2からnまで繰り返します。ARCHと同様、初期カットの数だけ余計に過程を生成してやりますから、全体の数はcutn＋nになります。最終的にできたものをcutn＋1から最後まで採用して最終的にｎ個の系列にします。以降、GARCH(2,2)となろうとも一般的なGARCH(p,q)になろうと原理は同じです。なお、分散安定のためには、aおよびｂのパラメータと1-sumc(a)-sumc(b)が０よりも大きい必要があります。また、a0=(1-sumc(a)-sumc(b))*s2;というふうにs2からa0は一意的に決まってきます。

プログラム

```
proc garch11(a1,b1,s2,cutn,n);
    local a0,u,h,t;
    a0=(1-a1-b1)*s2;
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    u[1]=sqrt(s2)*rndn(1,1); h[1]=s2;
    t=2;
    do while t<=cutn+n;
        h[t]=a0+a1*u[t-1]^2+b1*h[t-1];
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

GRACH(2,2)のケース

プログラム

```
proc garch22(a1,a2,b1,b2,s2,cutn,n);
    local a0,u,h,t;
    a0=(1-a1-a2-b1-b2)*s2;
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    u[1]=sqrt(s2)*rndn(1,1); u[2]=sqrt(s2)*rndn(1,1);
    h[1]=s2; h[2]=s2;
    t=3;
    do while t<=cutn+n;
        h[t]=a0+a1*u[t-1]^2+a2*u[t-2]^2+b1*h[t-1]+b2*h[t-2];
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

GRACH(p,q)の一般形

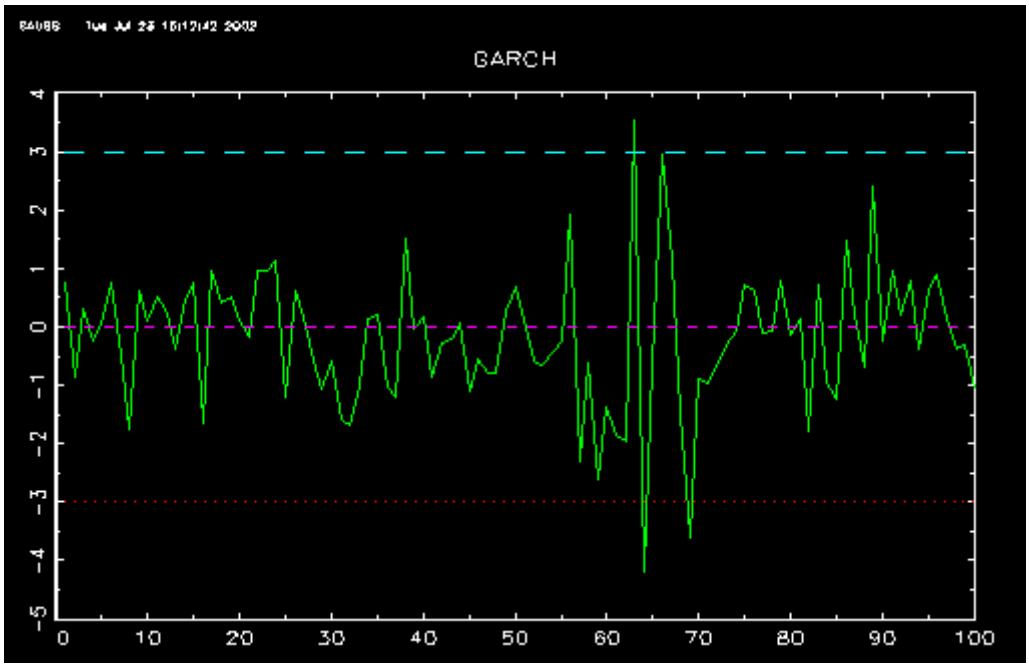
プログラム

```
    new; cls;
    a={0.2,0.1,0.1}; b={0.2,0.1,0.1,0.1}; s2=1; cutn=20; n=100;
    u=garchpq(a,b,s2,cutn,n);
    library pgraph;
    graphset;
    title("GARCH");
    xy(seqa(1,1,n),u~(3*ones(n,1))~(-3*ones(n,1))~zeros(n,1));

proc garchpq(a,b,s2,cutn,n);
    local a0,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);
    a0=(1-sumc(a)-sumc(b))*s2;
    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];
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

グラフ表示




上の一般的なGARCHのプログラムでは、ｐとｑの次数は必ずしも同じとは限らないケースを想定する必要があります。そのため、max=maxc(q|p)でもって、まずどちらか大きい方の次数を調べます。この最大次数をもとにuとhの初期値をunconditional variance s2を用いて設定します。そして、それぞれの次数が違うｈの再帰式（maxではなく、それぞれpお

よびqになっている点に注意）

h[t]=a0+rev(a)'(u[t-q:t-1]^2)+rev(b)'h[t-p:t-1]

を考えて、t＝max+1から考えているデータの系列分だけ過程を生成しています。

**パラメータa0を設定する方法**

同様に、s2ではなくてa0を設定する方法も簡単にできます。

プログラム

```
proc garchpq(a,b,a0,cutn,n);
    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];
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

上のように、ここでも以前と同様に、s2=a0/(1-sumc(a)-sumc(b));として一意的に決まってくるunconditional varianceをもとに、初期値を設定してから再帰式を回します。

IGARCH(p,q)過程

ここでは、以下のようなGARCH(p,q)の設定に加えて、a0を除くパラメータ間にΣa＋Σｂ＝１の関係が成り立っているときの過程をIGARCHと呼ぶ。

$$h_t = a_0 + a_1 u_{t-1}^2 + ... + a_q u_{t-q}^2 + b_1 h_{t-1} + ... + b_p h_{t-p}$$

$$u_t = \sqrt{h_t}\varepsilon_t \text{ここで} \varepsilon_t \sim NID(0,1)$$

IGARCH(p,q)の一般形

以下では、a0および各パラメータが与えられているケースのIGARCH過程のシミュレーション方法について扱う。方法は、GARCHの一般形と同じ計算を、パラメータ設定の時にその和がちょうど１になるように設定する。そうでなければ、エラーを返すようにすればよい。

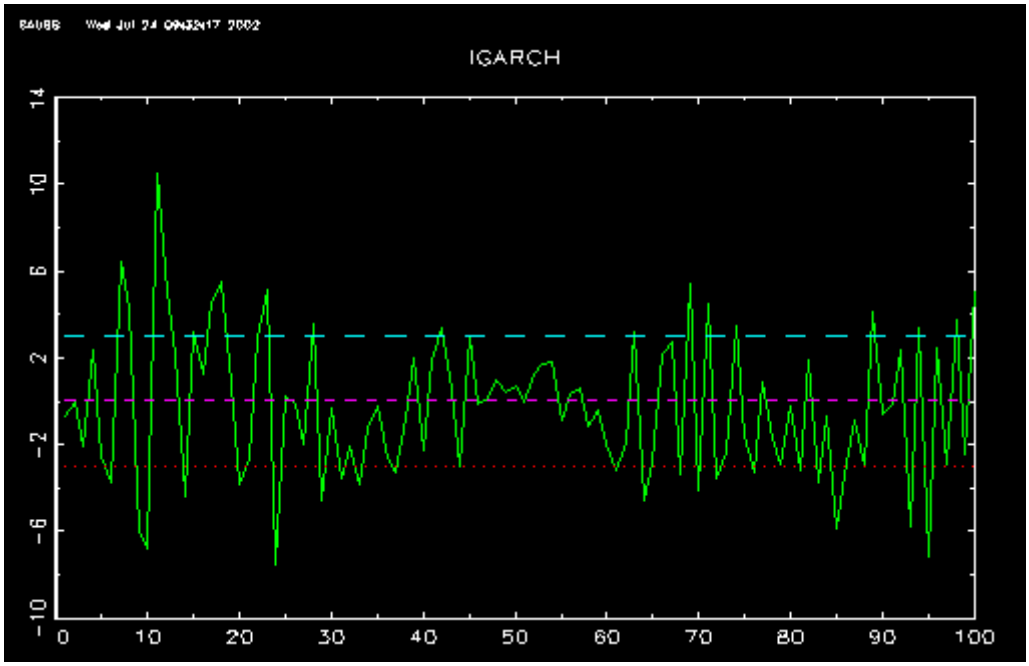
プログラム

```
new; cls;
a={0.3,0.1,0.1}; b={0.2,0.1,0.1,0.1}; a0=1; cutn=20; n=100;
u=igarchpq(a,b,a0,cutn,n);
library pgraph;
graphset;
title("IGARCH");
xy(seqa(1,1,n),u~(3*ones(n,1))~(-3*ones(n,1))~zeros(n,1));

proc igarchpq(a,b,a0,cutn,n);
    local a0,q,p,u,h,max,t;
    if sumc(a)+sumc(b)/=1;
        errorlog "ERROR:Parameter sum must be 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);
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    max=maxc(q|p);
    u[1:max]=sqrt(a0)*rndn(max,1); h[1:max]=a0*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];
        u[t]=sqrt(h[t])*rndn(1,1);
        t=t+1;
    endo;
    u=u[cutn+1:cutn+n];
    retp(u);
endp;
```

グラフ表示




ARCH型回帰モデルの設定とその推定[要ＣＭＬライブラリ]

後半では、シミュレーションされた残差uをもとにデータ系列を作成してやり、そのARCHおよびGARCHのパラメータ推定を行なう。

**ARCH(3)の推定**

乱数設定は以下のような

| n | $\beta_0$ | $\beta_1$ | $X_1$ | a0 | a1 | a2 | a3 |
|---|---|---|---|---|---|---|---|
| 100 | 0.5 | 2 | [-5,5] | 0.1 | 0.5 | 0.3 | 0.1 |

$y_t=\beta_0+\beta_1 X_t+u_t$についてのARCH(3)について推定することにします。まず、これによりuのARCH(3)過程を作り、$y_t=\beta_0+\beta_1 X_t+u_t$でｙの系列を導き出す。このｙとｘによって、あらためてARCH(3)のパラメータとβのパラメータを同時にLikelihoodを最大化することで推定してやることにする。以下のプログラムでは、archsimでは、uの計算はいままでどおりARCHの過程で求め、定数項の１の列ベクトルを含むＸの系列を乱数設定した

ものに、βをかけてy=x*beta+uでもってｙの系列を作成し、このｙとＸをリターンとして返してやっている。これを冒頭で、dataとして水平方向にマージして、通常のLog-Likelihood関数のσ$_2$のところに、ルートのｈ$_t$を２乗して１をかけたもの、すなわち、ｈ$_t$自身を代わりに代入した下のLog-Likelihoodのベクトル形

$$LL = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(h) - \frac{e^2}{2h} \qquad where \quad e = y - x\beta$$

を最大化します。ここで、ｈにはARCHまたはGARCHのｈの式をそのまま代入設定してやればよいことになります。なお、初期値については、シミュレーションの時のように定常下のunconditional varianceは所与ではないので、残差ｕを求めてやってから、０期以前のわからないパラメータはすべて０として、１期になった時点からｈ式にあてはめ、それをｈの式で再帰的にパラメータに応じて再帰的に計算しています。（パラメータの間の関係式からs2を求めて初期値にそれを用いる方法もありますが、推定の場合、最適化アルゴリズムの検索過程でパラメータは頻繁に変化してｈが０になって回らなくなってしまうケースもでてくるため、ここでは０期以前のわからないパラメータをすべて０と置く方法を採用しています。）なお、CMLのLog-Likelihoodのベクトル形の最適化では、_cml_Bounds=によって、a0は0.001以上でa0＞０を近似的に表し、その他のβ以外のパラメータはすべて０と１の間にくるように設定してあります。また、定常条件のa1+a2+a3＜1を表すのに、

_cml_c={0 0 0 -1 -1 -1}; _cml_d=-0.9999;

でもって、不等号関係を表すグローバル変数_cml_cと_cml_dを設定します。（なお、_cml_aと_cml_bは等号関係の行列設定です。）すなわち、この場合の不等号の向きは＞で一定で

$$\begin{bmatrix} 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \geq 0.9999$$

となりますから、－１を両辺にかけて符号を逆転させると、a1+a2+a3≦0.9999となります。これでもって、近似的に、定常条件のa1+a2+a3＜1を表します。ｌここでのアルゴリズムは４のBHHHです。Startベクトルはｈが０にならないように適当に０でない部分を作ると同時に、上の不等号条件にもかからないように、a1+a2+a3＜1を満たす点から始める必要があります。したがって、上では0.1を入れています。

プログラム

```
new; cls;
library cml;
cmlset;
```

```
    rndseed 4;
    a={0.5,0.3,0.1}; a0=0.1; cutn=10; n=100; beta={0.5,2};
    {y,x}=archqsim(a,a0,cutn,n,beta);
        data=y~x;
        _cml_Bounds={ -1e256     1e256,
                      -1e256     1e256,
                        0.001    1e256,
                        0              1,
                        0              1,
                        0              1};
      _cml_c={0 0 0 -1 -1 -1}; _cml_d=-0.9999;      /* a1+ a2+a3<=0.9999 */
      _cml_ParNames = {"const","beta","a0","a1","a2","a3"};
      _cml_Algorithm=4;
      start={0,0,1,0.1,0.1,0.1};
      {x,f,g,cov,retcode}=cml(data,0,&ll,start);
      cl=cmltlimits(x,cov);
      call cmlclprt(x,f,g,cl,retcode);

proc ll(b,data);
    local y,x,e2,a0,a1,a2,a3,h,t;
    y=data[.,1]; x=data[.,2:3]; beta=b[1:2];
    e2=(y-x*beta)^2;
    a0=b[3]; a1=b[4]; a2=b[5]; a3=b[6];
    h=zeros(rows(data),1);
    h[1]=a0;
    h[2]=a0+a1*e2[1];
    h[3]=a0+a1*e2[2]+a2*e2[1];
    t=4;
    do while t<=rows(data);
        h[t]=a0+a1*e2[t-1]+a2*e2[t-2]+a3*e2[t-3];
        t=t+1;
    endo;
    retp(-1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h);
endp;

proc(2)=archqsim(a,a0,cutn,n,beta);
```

```
    local s2,q,u,h,t;
    if sumc(a)>=1;
        errorlog "ERROR:Parameter sum must be less than 1.";
        retp(-1);
    endif;
    if cutn<rows(a);
        errorlog "ERROR:Initial cut must be at least greater than order q.";
        retp(-1);
    endif;
    q=rows(a);
    s2=a0/(1-sumc(a));
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    u[1:q]=sqrt(s2)*rndn(q,1);
    t=q+1;
    do while t<=cutn+n;
        h[t]=a0+rev(a)'(u[t-q:t-1]^2);
        u[t]=sqrt(h[t])*rndn(1,1);
        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;
```

画面表示

```
Mean log-likelihood        -0.650752
Number of cases      100
                                0.95 confidence limits
Parameters     Estimates      Lower Limit    Upper Limit    Gradient
-------------------------------------------------------------------
const             0.5077          0.4391         0.5764       0.0000
beta              1.9918          1.9680         2.0157      -0.0000
a0                0.0654         -0.0114         0.1422      -0.0000
a1                0.5662         -0.0055         1.1379      -0.0000
a2                0.3112         -0.0895         0.7118      -0.0000
```

```
a3                0.0362        -0.2628         0.3352       -0.0000
Number of iterations    17
Minutes to convergence      0.06033
```

**GARCH(1,1)の推定**

乱数設定は以下のような

| n | $\beta_0$ | $\beta_1$ | $X_1$ | a0 | a1 | b1 |
|---|---|---|---|---|---|---|
| 100 | 0.5 | 2 | [-5,5] | 0.1 | 0.5 | 0.3 |

$y_t = \beta_0 + \beta_1 X_t + u_t$についてのGARCH(1,1)について推定することにする。以下では、上のARCHとほぼ同じように乱数設定からｙとｘの系列を出してから、それをもとにもう一回GARCHの推定をしている。なお、

```
__weight=(rows(data)/(rows(data)-1))*ones(rows(data),1); __weight[1]=zeros(1,1);
```

の行によって、推定のデータの１番目のウエイトを０とするやり方で１番目のデータの影響を殺している（なくても可)。なお、コメントとして無効にしてあるところのように、

```
h=recserar(a0+lagn(e2,1)*a1,meanc(e2),b1);
```

としてAR過程を生成する組込み関数recserarを用いて、その第１要素にMAに相当する残差のラグ項計算結果を設定してやり、第２要素に、ARとなるｈのもととなるmeanc(e2)を、そして第３要素にその係数を入れてｈの項を計算させてもよい。なお、MAに相当する項の１番目にはラグをとっているので０が入っている。

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};
{y,x}=garchpqsim(a,b,a0,cutn,n,beta);
data=y~x;
_cml_Bounds={ -1e256     1e256,
              -1e256     1e256,
              0.001      1e256,
              0                1,
              0                1};
_cml_c={0 0 0 -1 -1}; _cml_d=-0.9999;      /* a1+ b1<=0.9999 */
_cml_ParNames = {"const","beta","a0","a1","b1"};
_cml_Algorithm=4;
start={0,0,0.1,0.1,0.1};
```

```
    __weight=(rows(data)/(rows(data)-1))*ones(rows(data),1); __weight[1]=zeros(1,1);
    {x,f,g,cov,retcode}=cml(data,0,&ll,start);
    cl=cmltlimits(x,cov);
    call cmlclprt(x,f,g,cl,retcode);

proc ll(b,data);
    local y,x,e2,a0,b1,a1,h,t;
    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];
    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(-1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h);
endp;
/*
proc ll(b,data);
    local y,x,e2,a0,b1,a1,h;
    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];
    h=recserar(a0+lagn(e2,1)*a1,meanc(e2),b1);
    retp(-1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h);
endp;
*/
proc(2)=garchpqsim(a,b,a0,cutn,n,beta);
    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];
        u[t]=sqrt(h[t])*rndn(1,1);
        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;
```

画面表示

```
Mean log-likelihood        -0.877694
Number of cases      100
                                0.95 confidence limits
Parameters    Estimates     Lower Limit   Upper Limit   Gradient
-------------------------------------------------------------------
const            0.4845         0.3913        0.5778       0.0000
beta             1.9793         1.9495        2.0090       0.0000
a0               0.0793        -0.0811        0.2397      -0.0000
a1               0.5041         0.0908        0.9174      -0.0000
b1               0.3517        -0.1994        0.9028      -0.0000
Number of iterations     142
Minutes to convergence      0.40817
```

GARCH(1,3)の推定

乱数設定は以下のような

| n | $\beta_0$ | $\beta_1$ | $X_1$ | a0 | a1 | a2 | a3 | b1 |
|---|---|---|---|---|---|---|---|---|
| 100 | 0.5 | 2 | [-5,5] | 0.1 | 0.3 | 0.2 | 0.1 | 0.1 |

$y_t = \beta_0 + \beta_1 X_t + u_t$についてのGARCH(3,1)について推定することにする。シードを適当にとってやらないと、ｐ、ｑどちらかの次数が２を超えるとパラメータの１つ以上が０に張りついてしまうようになる（その場合のgradientは０ではない）。

プログラム

```
new; cls;
library cml;
cmlset;
rndseed 55;
a={0.3,0.2,0.1}; b={0.1}; a0=0.1; cutn=20; n=100; beta={0.5,2};
{y,x}=garchpqsim(a,b,a0,cutn,n,beta);
data=y~x;
_cml_Bounds={ -1e256     1e256,
              -1e256     1e256,
              0.001      1e256,
              0                1,
              0                1,
              0                1,
              0                1};
_cml_c={0 0 0 -1 -1 -1 -1}; _cml_d=-0.9999;      /* a1+a2+a3+b1<=0.9999 */
_cml_ParNames = {"const","beta","a0","a1","a2","a3","b1"};
_cml_Algorithm=4;
start={0,0,1,0.1,0.1,0.1,0.1};
__weight=(rows(data)/(rows(data)-3))*ones(rows(data),1); __weight[1:3]=zeros(3,1);
{x,f,g,cov,retcode}=cml(data,0,&ll,start);
cl=cmltlimits(x,cov);
call cmlclprt(x,f,g,cl,retcode);

proc ll(b,data);
local y,x,e2,a0,a1,a2,a3,b1,h,t;
y=data[.,1]; x=data[.,2:3]; beta=b[1:2];
e2=(y-x*beta)^2;
a0=b[3]; a1=b[4]; a2=b[5]; a3=b[6]; b1=b[7];
h=zeros(rows(data),1);
h[1]=a0;
```

```
    h[2]=a0+a1*e2[1]+b1*h[1];
    h[3]=a0+a1*e2[2]+a2*e2[1]+b1*h[2];
    t=4;
    do while t<=rows(data);
        h[t]=a0+a1*e2[t-1]+a2*e2[t-2]+a3*e2[t-3]+b1*h[t-1];
        t=t+1;
    endo;
    retp(-1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h);
endp;

proc(2)=garchpqsim(a,b,a0,cutn,n,beta);
    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];
        u[t]=sqrt(h[t])*rndn(1,1);
        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;
画面表示

```
Mean log-likelihood        -0.630869
Number of cases      100
                                  0.95 confidence limits
Parameters     Estimates      Lower Limit    Upper Limit    Gradient
------------------------------------------------------------------
const             0.5186          0.4399         0.5974      -0.0000
beta              2.0063          1.9835         2.0292      -0.0000
a0                0.0559         -0.0112         0.1230      -0.0000
a1                0.4902          0.0334         0.9469      -0.0000
a2                0.1921         -0.4047         0.7889       0.0000
a3                0.1700         -0.1979         0.5380      -0.0000
b1                0.0620         -0.6945         0.8184      -0.0000
Number of iterations     42
Minutes to convergence      0.14000
```

**IGARCH(1,1)の推定**

乱数設定は以下のような

| n | $\beta_0$ | $\beta_1$ | $X_1$ | a0 | a1 | b1 | (a1+b1=1) |
|---|---|---|---|---|---|---|---|
| 100 | 0.5 | 2 | [-5,5] | 1 | 0.7 | 0.3 | |

$y_t=\beta_0+\beta_1 X_t+u_t$についてのIGARCH(1,1)について推定することにする。設定の仕方は全くGARCHと同じで、パラメータ合計を１にする。ただし、推定の仕方は、等号制約で

```
_cml_a={0 0 0 1 1}; _cml_b=1;
```

によって、

$$\begin{bmatrix}0 & 0 & 0 & 1 & 1\end{bmatrix}\begin{bmatrix}\beta_0\\ \beta_1\\ a_0\\ a_1\\ b_1\end{bmatrix}=1, \text{すなわち} a_1+b_1=1$$

の制限を加えてやる。なお、推定の段階で、この設定を加えないで推定してもデータがIGARCHにしたがっていれば、おおよそパラメータ合計が１になる。以下のプログラムではh[1]＝a0にして収束しやすくしている。もちろん、収束するのならばGARCHのようにstart値を変更してからlikelihoodのなかでh[1]＝a0/(1-a1-b1)を設定してもよい。

プログラム

```
new; cls;
library cml;
cmlset;
rndseed 1;
a={0.7}; b={0.3}; a0=1; cutn=100; n=100; beta={0.5,2};
{y,x}=igarchpqsim(a,b,a0,cutn,n,beta);
      data=y~x;
      _cml_Bounds={ -1e256     1e256,
                    -1e256     1e256,
                      0.001    1e256,
                      0              1,
                      0              1};
      @ _cml_a={0 0 0 1 1}; _cml_b=1; @
      _cml_ParNames = {"const","beta","a0","a1","b1"};
      _cml_Algorithm=4;
      start={0,0,1,1,1};
      __weight=(rows(data)/(rows(data)-1))*ones(rows(data),1); __weight[1]=zeros(1,1);
      {x,f,g,cov,retcode}=cml(data,0,&ll,start);
      cl=cmltlimits(x,cov);
      call cmlclprt(x,f,g,cl,retcode);

proc ll(b,data);
    local y,x,e2,a0,b1,a1,h,t;
    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];
    h=zeros(rows(data),1);
    h[1]=a0;
    t=2;
    do while t<=rows(data);
        h[t]=a0+a1*e2[t-1]+b1*h[t-1];
        t=t+1;
    endo;
    retp(-1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h);
endp;
/*
```

```
proc ll(b,data);
    local y,x,e2,a0,b1,a1,h;
    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];
    h=recserar(a0+lagn(e2,1)*a1,meanc(e2),b1);
    retp(-1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h);
endp;
*/
proc(2)=igarchpqsim(a,b,a0,cutn,n,beta);
    local q,p,u,h,max,t;
    if sumc(a)+sumc(b)/=1;
        errorlog "ERROR:Parameter sum must be 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);
    u=zeros(cutn+n,1);
    h=zeros(cutn+n,1);
    max=maxc(q|p);
    u[1:max]=sqrt(a0)*rndn(max,1); h[1:max]=a0*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];
        u[t]=sqrt(h[t])*rndn(1,1);
        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;
```

画面表示

Mean log-likelihood -2.54579
Number of cases 100

| Parameters | Estimates | 0.95 confidence limits Lower Limit | Upper Limit | Gradient |
|---|---|---|---|---|
| const | 0.5825 | 0.1682 | 0.9968 | -0.0000 |
| beta | 1.9511 | 1.7744 | 2.1278 | -0.0000 |
| a0 | 1.5913 | -0.0379 | 3.2204 | 0.0000 |
| a1 | 0.6905 | 0.2270 | 1.1541 | -0.0000 |
| b1 | 0.3179 | 0.0262 | 0.6095 | 0.0000 |

Number of iterations 31
Minutes to convergence 0.07967

画面表示（a1+b1=1の制限をかけたケース）
Mean log-likelihood -2.54580
Number of cases 100

| Parameters | Estimates | 0.95 confidence limits Lower Limit | Upper Limit | Gradient |
|---|---|---|---|---|
| const | 0.5846 | 0.1780 | 0.9912 | -0.0000 |
| beta | 1.9510 | 1.7744 | 2.1276 | -0.0000 |
| a0 | 1.6087 | 0.1354 | 3.0820 | 0.0000 |
| a1 | 0.6819 | 0.3905 | 0.9733 | -0.0028 |
| b1 | 0.3181 | 0.0267 | 0.6095 | -0.0028 |

Number of iterations 48
Minutes to convergence 0.10533
仮想的にIGARCHをシミュレートしたデータを用いても、係数の合計が１の合計をかけて推定をすると、その部分のgradientは完全には０にはならない。しかしながら、現実のデータを使った時よりも０に近い値になるのは確かである。

*/