4.5 特殊関数とライブラリ CML[要 full version]　　ver. 0.3

２回にわたって説明しましたMAXLIKは、制約条件なしにLog-Likelihood関数を最大化するような係数を求めることに特化したライブラリでした。それに対して、各係数パラメータに制約条件があるような条件のもとで、Log-Likelihood関数を最大化するような係数を求めるのがCMLライブラリです。名前のとおり、Constrained Maximum Likelihoodを求めるものです。もちろん、これには制約条件なしの計算も可能なので、MAXLIKで本来は行なう計算もCMLで行なわれていることも多々あります。文法は、ほとんどMAXLIKと同じで、グローバル変数の名前が_max何々となっていたところが、今度は_cml何々と変更になります。また、係数パラメータに制約を加える場合には、sqpSolveでやったような等号制約、不等号制約、それにバウンダリー条件を行列の形で表現して加えます。

まずは、前回MAXLIKで行なった計算をCMLで行なってみましょう。ｓのところに正値条件を加えてみます。Dドライブにdatafile1.txtというｙとｘのシリーズデータがあるものとします。これをCMLライブラリを呼び出して、CMLの制限条件つき最大化で計算させてみます。

プログラム

```
new; cls;
library cml;
cmlset;
load data[29,2]=d:datafile1.txt;
data=data[2:29,.];

start={0,0,1};
_cml_Bounds={-1e256 1e256,
                -1e256 1e256,
                 0.001 1e256};
_cml_Algorithm = 1;
_cml_ParNames = {"CONST","X","s"};
call cmlprt(cml(data,0,&ll,start));

proc ll(b,data);
    local beta, s, y, x, e;
    y=data[.,1];
    x=ones(28,1)~data[.,2];
    beta=b[1:2];
    s=b[3];
```

```
    e=y-x*beta;
    retp(-1/2*ln(2*pi)-1/2*ln(s^2)-1/2*e^2/s^2);
endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood         -5.33947
Number of cases     28

Covariance of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.    Gradient
-----------------------------------------------------------------
CONST           77.7952       21.2357      0.0000
X               52.0098        3.6923      0.0000
s               50.4274        6.7387     -0.0000

Number of iterations    93
Minutes to convergence     0.17933
```

上のプログラムは、MAXLIKのものとほとんど同じものです。まず、library cml;でCMLライブラリを呼び出して、そこにある関数群を使用可能にします。そして、cmlset;でCMLで使われる諸設定に関するグローバル変数を初期化します。スタート値を設定した後、バウンダリー条件を設定します。これは、spqSolveの時と全く同じように、この場合、３つのパラメータがありますから、３行２列の_cml_Boundsの行列値を設定することになります。１行目は、第１パラメータについての最小値と最大値を、２行目は第２パラメータの最小値と最大値を、そして３行目は、第３パラメータ（すなわちｓ）の最小値と最大値を設定しています。通常、慣習として、下天井は-1e256を、上天井は1e256を使用します。常に下限と上限を設定しなければなりません。第１パラメータおよび第２パラメータは制約がないので、-1e256から1e256までの値を設定しています。第３パラメータｓは０より大きい数なので、０からではなくて、０から有効数字程度分大きい、たとえば0.001などの数字から1e256までとしています。MAXLIKの際にアルゴリズム２であったBFGSは、CMLではアルゴリズム１となり、_cml_Algorithm = 1;　と設定します。なお、CMLでは、アルゴ

リズム１はBFGS、２はDEF、３はNEWTON、４はBHHHとなっています（ここでは、CMLバージョン1.0を用いています）。独立変数およびその他のパラメータの設定は、_cml_ParNamesで行ないます。MAXLINKで_max_ParNamesを設定したものを、cmlとするだけの変更となります。そして、Log-Likelihood関数llを最大化するために、call cmlprt(cml(data,0,&ll,start));とします。MAXLIKで、maxlikおよびmaxprtとなっていた関数名を、CMLではcmlおよびcmlprtに変更するだけです。結果は、若干アルゴリズムが違うため、係数の若干の違いは認められますが、MAXLIKのときとほぼ同じになっています。しかも、Ｓが負の数の場合にはiterationは回らなく、この設定の場合、Ｓが0.001以上の場合についてのみiterationが行なわれます。

ポイント　_cml_Bounds = { **第１パラメータの下限　　第１パラメータの上限,**
第２パラメータの下限　　第２パラメータの上限,
……
第Ｋパラメータの下限　　第Ｋパラメータの上限　};

ただし、慣習上－∞には-e256、＋∞には1e256を使う。
等号をともなわない下限または上限には、たとえば、0.0001を０＜または＜０の意味で使う。小数の大きさは任意。

次に、前章で扱った標準TOBITモデルについて、上の計算と同様に、Ｓのところに正値の制約を加えた計算をしてみます。

プログラム

```
new; cls;
library cml;
cmlset;
load data[51,7]=d:datafile6.txt;
data=data[2:51,.];

hour=data[.,2];
one=ones(50,1);
c18=data[.,4];
age=data[.,5];
age2=age^2;
ed1=data[.,6];
hi=data[.,7];
data=hour~one~c18~age~age2~ed1~hi;
```

```
_cml_Bounds = {-1e256   1e256,
               -1e256   1e256,
               -1e256   1e256,
               -1e256   1e256,
               -1e256   1e256,
               -1e256   1e256,
               0.0001   1e256};
_cml_ParNames = {"CONST","C18","AGE","AGE2","ED","HI","SIGMA"};
_cml_Algorithm = 4;
start={-1,0,0,0,0,0,1};
call cmlprt(cml(data,0,&ll,start));

proc ll(b,data);
    local y,x,s,beta;
    y=data[.,1];
    x=data[.,2:7];
    s=b[7];
    beta=b[1:6];
    retp( (y.==0).*ln(1-cdfn(x*beta/s))+ (y.>0).*lnpdfmvn(y-x*beta,s^2) );
endp;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood          -3.16591
Number of cases      50

Covariance of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.    Gradient
-------------------------------------------------------------------
CONST          -47.4330       45.8176     -0.0000
C18             -7.1220        3.1963     -0.0000
```

```
AGE              2.8162        2.3676      -0.0000
AGE2            -0.0432        0.0273      -0.0000
ED               2.5322        1.3753      -0.0000
HI               0.0001        0.0002      -0.0000
SIGMA           22.4303        3.0444       0.0000

Number of iterations    1074
Minutes to convergence     3.26883
```

MAXLIKからの変更点は、CMLライブラリを呼び出して、cmlset;でそのグローバル変数設定を初期化するほか、いままで_max何々となっていたグローバル変数を_cml何々に変更します。そして、この場合、_cml_Boundsというパラメータのバウンダリーを設定するところで、７番目のパラメータ、すなわちｓを0.0001から上天井の1e256までとしています。その他のパラメータについては、-1e256から1e256までとして事実上無制約にします。アルゴリズムの番号は、MAXLIKとはずれていますから、BHHHの場合は_cml_Algorithm = 4;というように４に変更します。そして、call cmlprt(cml(data,0,&ll,start));というcmlとその値を表示するcmlprtとが２重に入れ子にした関数をcallで呼んでいます。ここでも結果はMAXLIKと若干iterationの数、係数ともに若干の違いはありますが、ほとんど同じ結果になっています。正値制限を加えてやると、ほかのアルゴリズムでもｓがマイナスになって計算が続くということはなくなりますが、依然としてMAXLIK同様にアルゴリズムとスタートベクトルの組み合わせによっては収束しないという点は残ります。

ポイント　CMLの書式

```
{x,f,g,cov,retcode} = cmlprt(cml(dataset,0,&ll,start));
```

または、

```
call cmlprt(cml(dataset,0,&ll,start));
```

また、MAXLIKと同じように、収束の許容桁数、最大iteration数、最大iteration分の時間数を次のようなグローバル値のデフォルト値を変更することによって設定できます。

```
_cml_DirTol = 1e-5;        （収束の許容桁数）
_cml_MaxIters = 1e+5;      （最大iteration数）
_cml_MaxTime = 1e+5;       （最大iterationに要する分数）
```

ここで、ｓが０よりも大きいというバウンダリー条件だけではなくて、さらにいくつかの不等号、等号条件をもっているよく使われる例について説明します。

GARCH(1,1)

CMLの応用例としては、最も有力なのが、金融やマクロの時系列データにおいて残差項に一定の関係があって、そのパラメータに一定の条件関係があるGARCHファミリーのケースがあります。まずここで挙げるGARCH(p,q)は、

$$Y_t = X_t\beta + u_t$$

$$where \qquad u_t = \sqrt{h_t}\cdot v_t$$

ここでv_tはunit normal density

$$h_t = a_0 + b_1 h_{t-1} + b_2 h_{t-2} + \ \ldots\ . + b_p h_{t-p} + a_1 u_{t-1}^2 + a_2 u_{t-2}^2 + \ \ldots\ . + a_q u_{t-q}^2$$

としたときに、次のような条件がパラメータにあるものをGARCHと呼びます。

$a_0 > 0$

$b >= 0$

$a >= 0$

$$\sum_{i=1}^{p} b_i + \sum_{j=1}^{q} a_j < 1$$

ここでは、ｐ＝１、ｑ＝１の単純な例についてのプログラムをしてみます。この場合、δおよびθは１個ずつで、バウンダリー条件のほかに、b＋a＜１という不等号制約が加わります。今、Ｄドライブに、datafile14.txtというｙ、ｘの２シリーズの時系列データがそれぞれ100個の行だけラベルなしにあるものとします。

プログラム

```
new; cls;
library cml;
cmlset;
load data[100,2]=d:datafile14.txt;
data=data~ones(100,1);
_cml_Bounds={-1e256   1e256,
             -1e256   1e256,
              0.001   1e256,
              0              1,
              0              1};
_cml_c={0 0 0 -1 -1}; _cml_d= -0.9999;
_cml_ParNames = {"const","beta","a0","a1","b1"};
_cml_Algorithm=4;
b={0,0,1,1,1};
call cmlprt(cml(data,0,&ll,b));
```

```
proc ll(b,data);
    local y,x,e2,beta,a0,a1,b1,h;
    y=data[.,1]; x=data[.,3]~data[.,2]; 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;
```

画面表示

```
return code =    0
normal convergence

Mean log-likelihood        -0.482068
Number of cases      100

Covariance of the parameters computed by the following method:
Inverse of computed Hessian

Parameters    Estimates     Std. err.    Gradient
------------------------------------------------------------------
const            0.4772        0.0377     -0.0000
beta             1.9985        0.0130     -0.0000
a0               0.0611        0.0409      0.0000
a1               0.2089        0.1670      0.0000
b1               0.4134        0.2513      0.0000

Number of iterations    29
Minutes to convergence      0.06133
```

上のように、Log-Likelihood関数を作るのは、残差項の関係にＡＲＭＡのようなプログラムを必要とするので容易ではありませんが、その部分をまず上のように与えられたものとして、そのLog-Likelihood関数を条件付の最大化をしてみます。ｂには、その１、２番目にはｘの係数がきて、３番目以降、a0,a1,b1の順で設定されていて、それぞれに制約を課します。ここで、_cml_Bounds=のところは、第１、第２パラメータは無制約、第３パラメータのa0は０よりも大きいですから、下限は０ではなくて0.001にしています。第４、第５パラメータは０を含んでそれ以上ですから、ともに下限は０に設定されます。上限はGARCHの定常

性から足し合わせたものが１より小さくなりますから、それぞれも１が上限になります。最適値問題の章のsqpSolveのところでやったのと同様に、不等号の制約は

```
_cml_C * b >= _cml_D
```

の関係が成り立ちますから、符号の向きが逆の_cml_C * b ＜ _cml_D　の場合には、両辺に－１をかけて、しかも等号を含まない不等号ですから、－１ではなくて、-0.9999を採用しています。第３、第４パラメータにかかる係数はマイナスの符号をつけて-1と-1になっていて、その他の部分は０が入ります。すなわち、

$$\begin{bmatrix}0 & 0 & 0 & -1 & -1\end{bmatrix}\begin{bmatrix}const\\ beta\\ a0\\ a1\\ b1\end{bmatrix} \geq -0.9999$$

とプログラムでは表していることになり、それは－a1－b1≧－0.9999のことであって、これは両辺に－1をかけるとa1＋b1≦0.9999となって、近似的にa1＋b1＜1を表している。これらのパラメータ制約のもとで、アルゴリズム４のBHHHでLog-Likelihood関数を最大化するベクトルｂをCMLで求めて、cmlprtで画面表示させています。

さて、Log-Likelihood関数のprocedureのところですが、これは基本的に２つの式の連立になっています。まず、$h_t = a1 + b_1 h_{t-1} + a_1 u_{t-1}^2$ というパラメータの制約があります。これに対して、v_tはunit normalなdesityですから、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ます。

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

したがって、Log-Likelihoodのprocedureのリターンが -1/2*ln(2*pi)-1/2*ln(h)-1/2*e2./h というふうに足し合わせる前の列ベクトルになっています。ここまでは、容易にプログラムをたてられると思います。問題は、

$$h_t = a0 + b_1 h_{t-1} + a_1 u_{t-1}^2$$

の部分です。これはＡＲプロセスを作る組込み関数recserarと、v_tがunit normalなdesityであることを利用してe2の部分の列ベクトルを１つずらす方法で、e2(t-1)、すなわちここでは、u_{t-1}^2のシリーズを作り出すという２つの合わせ技でｈの式を書きます。まず、前半部分のp=1に相当する部分は元のe2の列ベクトルを１つずらします。これにはlagnの組込み関数を用います。たとえば、lagn(a,1)とすると、aという列ベクトルを下方向に１だけずらし

て、ずれてなくなった先頭のところを０で埋め合わせる働きをします。そこで、aをたとえば１から20までの数列として、これを１つ下にずらして、一番最初に０で穴埋めすると、0から19までの列ベクトルの数列が生成されます。この１つずらす箇所を利用して、ｑ＝１の部分をまず作ります。この１つずらしたシリーズにa1をかけて、さらにこれにa0を加えることによって、ＡＲプロセスの組込み関数で動かない定数項とします。それが、ＡＲプロセスを生成する組込み関数recserarの第１要素になります。そして、その第２要素をもとにしてＡＲ項を作り、第３要素のb1がスケラーの場合にはｑ＝１の場合の$b_1 h_{t-1}$が生成されます。この場合の第２要素には、meanc(e2)がきています。これらをまとめて、h=recserar(a0+lagn(e2,1)*a1, meanc(e2),b1);ということになります。

ポイント　recserar(x, y0, δ) で、δがスケラーの場合、$y_t = x + \delta y_{t-1}$ が作られる
さらに、δが列ベクトルの場合、$y_t = x + \delta_1 y_{t-1} + \delta_2 y_{t-2} + \ldots + \delta_p y_{t-p}$

この場合、ｘのところにa0＋$a_1 u_{t-1}^2$　がきていて、この部分が固定されてＡＲプロセスがmeanc(e2)をもとに生成されます。ただし、これでは第１行目のシフトしたあとの０の入っているところもはいっているわけで、厳密ではありません。

さらに、0.95 confidence limits、すなわち９５％の信頼区間の下限と上限を計算したものをパラメータの係数とともに表示させるのには、cmlprtを使うのではなくて、次のようにしてやります。

プログラム

```
new; cls;
library cml;
cmlset;
load data[100,2]=d:datafile14.txt;
data=data~ones(100,1);

_cml_Bounds={-1e256   1e256,
             -1e256   1e256,
              0.001   1e256,
                  0        1,
                  0        1};
_cml_c={0 0 0 -1 -1}; _cml_d= -0.9999;
_cml_ParNames = {"const","beta","a0","a1","b1"};
_cml_Algorithm=4;
b={0,0,1,1,1};
```

```
{x,f,g,cov,retcode}=cml(data,0,&ll,b);
cl=cmltlimits(x,cov);
call cmlclprt(x,f,g,cl,retcode);

proc ll(b,data);
    local y,x,e2,beta,a0,a1,b1,h;
    y=data[.,1]; x=data[.,3]~data[.,2]; 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;
```

画面表示

```
Mean log-likelihood        -0.482068
Number of cases      100
                                0.95 confidence limits
Parameters     Estimates     Lower Limit    Upper Limit    Gradient
-----------------------------------------------------------------
const             0.4772         0.4024         0.5520      -0.0000
beta              1.9985         1.9727         2.0244      -0.0000
a0                0.0611        -0.0201         0.1423       0.0000
a1                0.2089        -0.1226         0.5404       0.0000
b1                0.4134        -0.0854         0.9123       0.0000
```

プログラムの中の太字の部分のように、cmlはcmlで別に計算します。そしてそのリターンを出すように書きます。すなわち、

```
{x,f,g,cov,retcode}=cml(data,0,&ll,b);
```

とします。５つあるリターンのうち、第１番目のxと第３番目のcovを使って、CMLのライブラリの中にある関数のconfidence limitsを求めるcmltlimitsを用いて、それぞれのパラメータの信頼区間を求めます。このcmltlimitsでは、グローバル変数_cml_NumObsでデータのそれぞれのシリーズの個数を設定しなければなりません。そして、xとcovをインプットして、clとして信頼区間がリターンされたものを、cmlclprtで表示します。その際、cmlprtとは違って、その第４インプットにはclがきていることに注意してください。このようにVarianceCovariance行列ではなくて信頼区間が第４要素にくる場合には、通常のcmlprtで

はなくて、cmlclprtを用います。なお、confidence limitsのデフォルトは、1-0.05の0.95、すなわち、９５%信頼区間がデフォルトですが、これを、たとえば、

_cml_Alpha=0.01;

とすると、1-0.01の0.99、すなわち99%信頼区間が得られます。_cml_Alphaのグローバル変数の設定には、小数が使われることとともに、１からその数を引いたものが実際の信頼区間になるので注意が必要です。

ポイント　cl=cmltlimits(x,cov);　**パラメータの列ベクトルx、VarCov行列がcovの時の各パラメータの95%信頼区間を求める。**

その際には、cmlprtではなく、cmlclprt(x,f,g,cl,retcode)で受けて表示する。

ただし、上のプログラムも一見うまくできているようですが、GAUSSのCMLデータハンドリングの仕様の視点からは、なおも不正確さが残ります。_cml_NumObsの設定で=99とすることによって100個の系列ではなくて99個で分析もできますが、これではウエイトをかけて初期の１データを殺していることにはなりません。そこで、各行にウエイトをかけて全99行のLog-Likelihoodを最大化します。正しくは、

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

の行を加えて、各行のウエイトをデフォルトの１から変更します。100/99にしてやって、最初のデータのウエイトを０にします。（下線はそれぞれ２つ分です。）

画面表示

```
Mean log-likelihood        -0.485442
Number of cases      100
                                0.95 confidence limits
Parameters     Estimates     Lower Limit    Upper Limit    Gradient
-----------------------------------------------------------------
const             0.4754         0.4007         0.5500      -0.0000
beta              1.9979         1.9721         2.0237      -0.0000
a0                0.0605        -0.0194         0.1404      -0.0000
a1                0.2115        -0.1199         0.5429      -0.0000
b1                0.4152        -0.0756         0.9059      -0.0000
Number of iterations     45
Minutes to convergence      0.09783
```

結果は、前のものと比べて若干変ってきます。このように、各行のウエイトをデフォルトの１から、「全行数／(全行数－１)」に若干増やしてやって、さらにオーバーライトする形

で、１行目だけのウエイトを０に再定義します。具体的には、下線２つにweightの__weightのグローバル変数を変更します。なお、ｎ行分だけ１行目からウエイト０にするには、

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

というふうに、分母で１を引くのではなくてｎを引いてウエイトを増して、さらに、１行目からｎ行目までのウエイトにｎ行１列のすべてが０の要素を代入します。ここで、信頼区間を計算するグローバル変数_cml_NumObsの扱いですが、これは全100行ではなくて、計算上、実質上の行数の１減らした99とした数で計算した方が、より正確であるように思われます。係数には影響はありませんが、信頼区間の計算で値が変ってきます。非常に微妙な問題ですが、GARCH(p,q)のｐまたはｑのどちらか大きい方の一方の数が１ではなくて大きな数の場合に（上からカットする行数はｐまたはｑのどちらか大きい方の数)、計算結果に無視できない影響を及ぼしてきます。通常は考えなくてもかまわないでしょう。

IGARCH(1,1)

さらに、GARCHの場合の最後の制約条件が等号条件、すなわち、

$$\sum_{i=1}^{p} b_i + \sum_{j=1}^{q} a_j = 1$$

の場合をIGARCHと言います。この場合のp=1,q=1のケースについて同じような計算をさせてみましょう。今度は、この場合、

$$b + a = 1$$

というかなりきつい制約になります。変更箇所は、不等号制約ではなくsqpSolveでやったように、等号制約の_cml_Ａと_cml_Ｂの２つの行列を設定することになります。今、Dドライブにdatafile15.txtのファイルがあるものとします。

プログラム

```
new; cls;
library cml;
cmlset;
load data[100,2]=d:datafile15.txt;
data=data~ones(100,1);

_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;
b={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,b);
_cml_NumObs=99;
cl=cmltlimits(x,cov);
call cmlclprt(x,f,g,cl,retcode);

proc ll(b,data);
    local y,x,e2,beta,a0,a1,b1,h;
    y=data[.,1]; x=data[.,3]~data[.,2]; 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;
```

画面表示

```
Mean log-likelihood         -3.07455
Number of cases      99
                                0.95 confidence limits
Parameters     Estimates     Lower Limit    Upper Limit    Gradient
-------------------------------------------------------------------
const             0.5736        -0.1625         1.3098       0.0000
beta              1.8396         1.5693         2.1098       0.0000
a0                1.4381        -0.8037         3.6798       0.0000
a1                0.3668         0.0809         0.6526      -0.0574
b1                0.6332         0.3474         0.9191      -0.0574
Number of iterations     37
Minutes to convergence      0.07600
```

上のように、等号制約の行列ＡおよびＢを用いて計算すると、

_cml_Ａ＊ｂ＝_cml_B

すなわち、この場合、_cml_Aは１×５、ｂは５×１、_cml_Bは１×１のスケラーで、

$$\begin{bmatrix}0 & 0 & 0 & 1 & 1\end{bmatrix}\begin{bmatrix}const\\ beta\\ a0\\ a1\\ b1\end{bmatrix}=1$$

1×a1＋１×b1の制約を課すようになるように、_cml_a={0 0 0 1 1};　_cml_b=1; というふうになっています。IGARCHへの変更箇所は、これら２行だけです。

ポイント　等号制約　　_cml_Ａ＊ｂ　＝　_cml_B
不等号制約　　_cml_Ｃ＊ｂ＞＝_cml_Ｄ

(例)　ｂが４つのパラメータβ，κ，δ，θの４×１の列ベクトルとすると、

２β＋κ＋３δ　　＝１
　３κ＋δ－４θ＝０　の場合には

```
_cml_A= {2   1   3   0,          _cml_B = {1,
         0   3   1   -4  }                   0 }
```

β＋κ＋δ＋θ＜＝１
２β　　＞＝１
　κ－２δ　　＜１の場合には

```
_cml_C= {-1   -1   -1   -1,      _cml_D= {-1,
          2    0    0    0,                1,
          0   -1    2    0 }           -0.9999 }
```