3.19　VAR モデル(1)　　ver.0.1

共和分の後半でも少しふれたような、OLS で言うところの従属変数と独立変数の区別をつけない（内生変数と外生変数の区別がない）ようなベクトル自己回帰モデル（すべての変数のラグ項で現在の変数を説明するもの）を推定、予測、それにともなう諸々の分析と踏み込んで扱います。モデル自体の基本は、ラグがいくつになろうと極めてシンプルです。

基本線形モデル

まず、もとの系列には単位根や共和分が存在しないと仮定します。線形の OLS 回帰の応用として、従属変数側にｔ期の全変数の列、独立変数側に定数項とｔ－１期からｔ－lag 期までの全変数の列がくるものとして、従属変数の数が複数の線形の OLS 回帰をします。ラグ次数を２と前もって決めてやるとすると、ここでの３変数のモデル VAR(2)は、

$$Yt = \beta_{11} + \beta_{12} Yt\text{-}1 + \beta_{13} Lt\text{-}1 + \beta_{14} Kt\text{-}1 + \beta_{15} Yt\text{-}2 + \beta_{16} Lt\text{-}2 + \beta_{17} Kt\text{-}2 + u_1$$
$$Lt = \beta_{21} + \beta_{22} Yt\text{-}1 + \beta_{23} Lt\text{-}1 + \beta_{24} Kt\text{-}1 + \beta_{25} Yt\text{-}2 + \beta_{26} Lt\text{-}2 + \beta_{27} Kt\text{-}2 + u_2$$
$$Kt = \beta_{31} + \beta_{32} Yt\text{-}1 + \beta_{33} Lt\text{-}1 + \beta_{34} Kt\text{-}1 + \beta_{35} Yt\text{-}2 + \beta_{36} Lt\text{-}2 + \beta_{37} Kt\text{-}2 + u_3$$

となり、GAUSS の計算の列表示にすれば、ラグによって欠損する行を取り除いたものを考えて、従属変数側には現在のｔ期の(N－2)×3 の系列が、独立変数側には(N-2)×(1＋2×3)の系列がそれぞれくることになります。すなわち、

従属変数側			独立変数側						
Yt	Lt	Kt	1	Yt-1	Lt-1	Kt-1	Yt-2	Lt-2	Kt-2
:	:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:	:

というふうになります。行数は OLS をするために共通して、すべてＴ－2 となります。以下のプログラムは、もちろん、呼び出し部分で３列より大きな列の data 行列を入れれば、どんな大きなディメンションにも対応できるものです。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
data=Y~L~K;
call varmain(data,2);                    /* Lag # here is predetermined as 2. */

proc(6)=varmain(data,lag);
```

```
    local n,k,i,xlag,y,x,b,e,vcv,se,t,xgiven,projection,projectiont;
    local j,a,data1,n1,datahat,x1,y1;
    n=rows(data); k=cols(data);
    i=2; xlag=data[lag:n-1,.];
    do while i<=lag;
        xlag=xlag~data[lag+1-i:n-i,.];
        i=i+1;
    endo;
    y=data[lag+1:n,.];
    x=ones(n-lag,1)~xlag;
    b=inv(x'x)*x'y;
    print "b:" b;
    e=y-x*b;
    vcv=e'e/(n-lag);
    se=sqrt(diag(inv(x'x))*diag(vcv)');
    t=b./se;
/* Projection */
    i=0; xgiven=1;
    do while i<=lag-1;
        xgiven=xgiven~data[n-i,.];
        i=i+1;
    endo;
    projection=(b'xgiven')';
    print "Projection(T+1):";print projection;
/* Projection a periods ahead */      a=5;
    print/lz "Projections(T+a): a=2 through" a;
    data1=data|projection;
    j=2;
    do while j<=a;
        n1=rows(data1);
        i=0; xgiven=1;
        do while i<=lag-1;
            xgiven=xgiven~data1[n1-i,.];
            i=i+1;
        endo;
        projectiont=(b'xgiven')';
```

```
        print projectiont;
        data1=data1|projectiont;
        j=j+1;
    endo;
/* VAR Estimated Data in the past */
    j=lag; datahat=zeros(n,k);
    do while j<=n-1;
        i=0; xgiven=1;
        do while i<=lag-1;
            xgiven=xgiven~data[j-i,.];
            i=i+1;
        endo;
        datahat[j+1,.]=(b'xgiven')';
        j=j+1;
    endo;
    datahat=datahat[lag+1:n,.];
/* Graph real data vs. estimated data */
        library pgraph;
        k=1;          @ k=1) 1 only. 0) No graph. k) All graphs. @
        i=1;
        do while i<=1;
            graphset;
            x1=seqa(lag+1,1,n-lag);
            y1=data[lag+1:n,i]~datahat[.,i];
            _plegctl=1;
            _plegstr="Real¥000Estimated";
            xy(x1,y1);
            i=i+1;
        endo;
    retp(b,se,t,vcv,projection,datahat);
endp;
```

上のプログラムでは、まず３変数からなるデータ行列を従属変数独立変数のわけへだてなく変数 data としてまとめて、ラグ数２とともに、varmain という procedure を作ってそれを呼び出します。Proc varmain の中味は、通常の OLS をまとめて行なう部分、１期だけ先の Projection を計算する部分、a=5 期だけ先までの Projection を計算する部分、過去の

データの VAR による Estimated Data を求める部分、そしてそれをグラフにする部分からなっています。まず、最初の複数の OLS を一度に行なうのには、通常の OLS の行列計算と同じように、ｙ側に複数列をおいて、β=inv(x'x)x'y で計算できます。得られたβも複数列になります。この場合、３列になります。１列目が１本目の推定、２列目が２本目の推定、３列目が３本目の推定となっていて、上から順に定数項の係数、Ｙt-1、Ｌt-1、Ｋt-1 そして . . .最後にＫt-2 の係数になっています。同様にして、残差行列も複数列になり、この場合、３列になります。そのｅで残差の Variance Covariance 行列を求めるには、

vcv=ｅ'ｅ／(n－lag)

となります。ここでは、分母に自由度ではなくて、実質使われている系列の数が来るのが正当なやり方です。（EVIEWS 等は、さらに従属変数側の列数、この場合７を引いたものを分母にします。vcv=ｅ'ｅ／(n－lag－cols(x))となります。）βの各係数の SE は

se=sqrt(diag(inv(x'x))*diag(vcv)');

で求めます。βもこの se もともに、この場合７×３の行列になっています。これらを要素対要素で t=b./se;として割ってやれば、それぞれに対するｔ値が求まります。

次に、わかっているデータから１期先の projection を求めるには、推定されたβをもとに、推定式の右辺の t-1 期に最終 N 期、t-2 期に N－1 期のデータをそれぞれ代入して、

$Y_{N+1}=\beta_{11}+\beta_{12}Y_N+\beta_{13}L_N+\beta_{14}K_N+\beta_{15}Y_{N-1}+\beta_{16}L_{N-1}+\beta_{17}K_{N-1}$

$L_{N+1}=\beta_{21}+\beta_{22}Y_N+\beta_{23}L_N+\beta_{24}K_N+\beta_{25}Y_{N-1}+\beta_{26}L_{N-1}+\beta_{27}K_{N-1}$

$K_{N+1}=\beta_{31}+\beta_{32}Y_N+\beta_{33}L_N+\beta_{34}K_N+\beta_{35}Y_{N-1}+\beta_{36}L_{N-1}+\beta_{37}K_{N-1}$

をまとめて計算するとＹN+1 とＬN+1 とＫN+1 の３つの projection が得られます。なお、βは７×３というふうに立っていますから、ｂを転置されて b'として３×７に寝させてから、（定数項の１を最初に含む）わかっているデータ７×１のデータを与えます。７×１のわかっているデータを求めるには、xgiven=1 としてやって、それに３×１ずつのデータ系列を最終Ｎ期からラグだけ１期ごとずらして横にループでマージします。この xgiven をさらに転置させたもの xgiven'をさきほどのｂの転置行列 b'にかけて、３×１のＮ＋１期の projection が得られます。それを表示の便宜上、さらに転置させて１×３にしています。

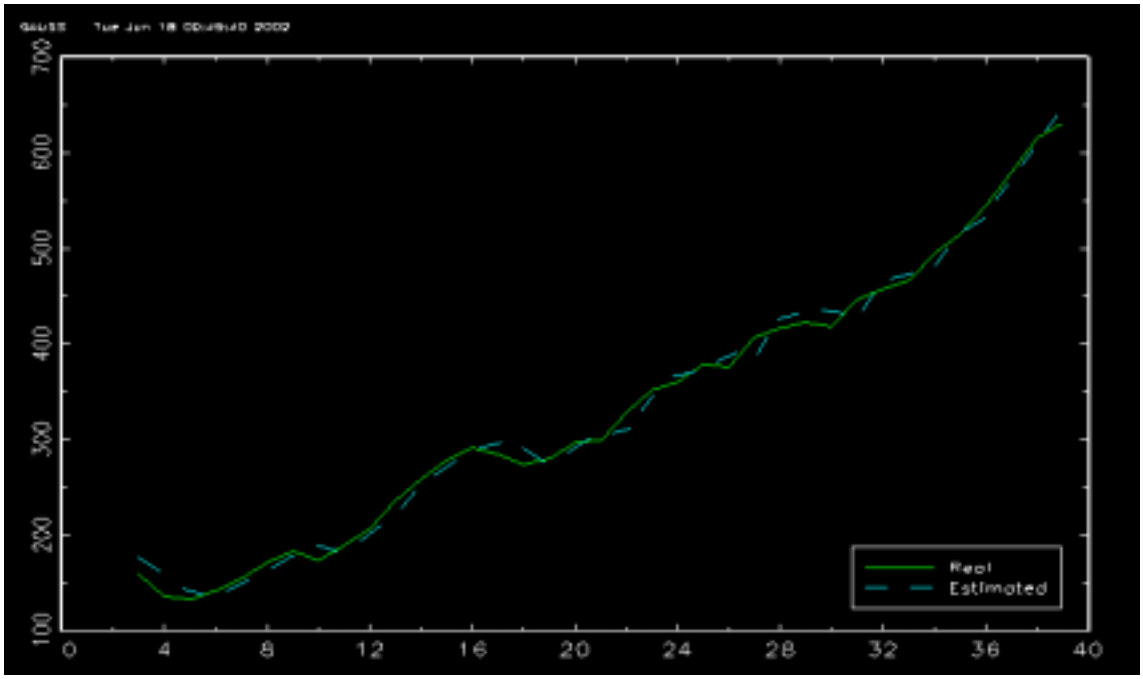
同様にして、さらにその先の projection t を求めるには、上で得られたＮ＋１期の projection を最終データとして、そこからラグをずらして同じように xgiven を与えてやって、その次の projhection を得ます。これを何回か繰り返すと、その分だけ projection が先にのばすことができます。内側の i ループは１期の projection のものと同じで、外側の j ループで j=2 から求める prujection 数 a=５まで同じことを最終に得られた projectuon を基準にずらしてもとめてやっています。

次に、これを過去のデータにあてはめると、Estimated Data すなわち、ちょうど VAR(2)でフィルタリングしたような値が求められます。VAR Estimated の内側の i ループは上のものと同じで、外側の j ループで lag 行から n-1 行までの既存のデータを用いて、lag+1 行からｎ行までの Estimated Data を求めています。ループの外で、ｎ×ｋの datahat を零行

列と初期化しておいてから、ここにループでデータを１行ずつ代入します。なお、できあがった Estimated Data は、最初の１行目から lag 行目までは欠損値になって０が入ったままになっていますから、datahat=datahat[lag+1:n,.];として最初の lag 行をカットします。

最後に、もとの Real Data とこの Estimated Data をマージしてｙ軸としてグラフを描きます。第 i 番目の変数から最終第ｋ番目の変数までのグラフを描かせるために単にループをまわしています。ｋに１を入れれば、第１番目の変数についてのみ描かれます。まず、グラフを描くには、pgraph のライブラリを呼び出して、そのグローバル変数を graphset によって初期化しますが、グラフごとに設定がことなりますから、ループの中に graphset; の文を置きます。_plegctl=1;は、凡例をデフォルトどおり右隅下に置くグローバル変数設定です。スケラーの非零であれば、すべて位置は右隅下になります。_plegstr=で文字列を凡例に入れます。区切りは、¥000 となります。（¥は英語半角のバックスラッシュに相当するもので、日本語では¥のしるしになります。）引用符の中に、これで区切って文字を入れます。そして、ｘ軸には lag+1 から１刻みの数列を設定して、xy グラフで描かせます。

グラフ表示



画面表示

b:

72.950335	98.852559	23.062726
1.2728367	0.20474814	-0.065690855

```
      -0.64742976        0.52189797        0.015346008
        1.1757222        0.29065623          1.5348256
      0.066626261        0.29232077          0.21806960
      0.087756574       -0.26734743         -0.20352127
       -1.5861833        -1.0159535         -0.74958780
Projection(T+1):
        662.69046         319.77700          263.43280
Projections(T+a): a=2 through 5
        694.73647         329.08436          278.48765
        725.96392         334.38234          291.87020
        757.08759         339.01523          304.24911
        789.57597         345.51962          316.97525
```

上の１列目はすべて、１番目の変数、すなわちＹに対するもの、２列目は２番目の変数Ｌ、３列目は３番目の変数Ｋに対するものです。ｂのところは上述のモデルのβの係数が行列形で転置した形になっています。

最適ラグ数の設定

ここでは、Akaike、Schwarz および Hannan-Quinn といった基本的な Information Criteria に基づいてラグ次数を決定します。前章の共和分のテストのところで、マニュアルで前もって定めていたラグ数も、これによって決定します。

$$LL = -\frac{kT}{2}(1+\ln 2\pi) - \frac{T}{2}\ln|\Omega| \qquad \text{ここで}\Omega\text{は}\det(VCV),T\text{は実際の行数}$$

$$AIC = -\frac{2LL}{T} + \frac{2n}{T}, SC = -\frac{2LL}{T} + \frac{n\ln T}{T}, HQ = -\frac{2LL}{T} + \frac{2n\ln(\ln T)}{T}$$

ここで n はパラメータ数。AIC または SC または HQ を最小にするモデルのラグ数を最適のラグ次数に選びます。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
data=Y~L~K;
call varlag(data,1);


proc varlag(data,opt);
  local maxlag,crit,aic,sc,hq,fpe,j,i,n,k,t,np,xlag,y,x,b,e,s2mat,detvcov,ll,lag;
  /* opt= 1 for min AIC,   2 for min SC,   3 for min HQ. */
```

```
if opt==1;
    print "Min AIC criterion:";
elseif opt==2;
    print "Min Schwarz information criterion:";
elseif opt==3;
    print "Min Hannan-Quinn information criterion:";
else;
    errorlog "ERROR:Opt # must be 1:AIC or 2:SC or 3:HQ.";
    retp(-1);
endif;
maxlag=1;
do while maxlag<=24;
    if rows(data)-maxlag-cols(data)*maxlag-1<cols(data);
    @   d.f. is set to be more than # of variables. @
        break;
    endif;
    maxlag=maxlag+1;
endo;
maxlag=maxlag-1;
crit=zeros(maxlag,3);
j=1;
do while j<=maxlag;
    n=rows(data); k=cols(data); t=n-j; np=k*(1+k*j);
    i=2; xlag=data[j:n-1,.];
    do while i<=j;
        xlag=xlag~data[j+1-i:n-i,.];
        i=i+1;
    endo;
    y=data[j+1:n,.];
    x=ones(n-j,1)~xlag;
    b=inv(x'x)*x'y;
    e=y-x*b;
    s2mat=e'e/(n-j);
    detvcov=abs(det(s2mat));
    if detvcov<1e-16;   /* Avoid 0 determinant. */
        detvcov=1e-16;
```

```
        endif;
        ll=-k*t/2*(1+ln(2*pi))-t/2*ln(detvcov);
        aic=-2*ll/t+2*np/t;
        sc=-2*ll/t+np*ln(t)/t;
        hq=-2*ll/t+2*np*ln(ln(t))/t;
        crit[j,.]=aic~sc~hq;
        j=j+1;
    endo;
    print "       LAG                AIC                SC                HQ";;
    print seqa(1,1,maxlag)~crit;
    lag=minindc(crit[.,opt]);
    print;
    print/rz "Choice of LAG #" lag;
    call varmain(data,lag);
    retp(lag);
endp;
```

（　この後に、proc varmainを置く　）

画面表示

```
Min AIC criterion:
     LAG                AIC                SC                HQ
       1.0000000        18.414644        18.931776        18.598636
       2.0000000        18.305424        19.219729        18.627759
       3.0000000        18.374404        19.694003        18.834979
       4.0000000        18.655474        20.388576        19.253741
       5.0000000        18.643364        20.798226        19.378233
       6.0000000        18.627329        21.212206        19.497061
       7.0000000        18.764165        21.787246        19.766231
       8.0000000        16.406679        19.876003        17.537592

Choice of LAG #                8
```

上のプログラムで重要なことは、ラグ数が、自由度を計算したときに変数の個数よりも小さくならないようにしているところです。もし小さくなるか、マイナスの値に自由度がなってしまうと det(vcv)が非常に 0 に近い数になって、一時的に情報値がマイナスか或いは相対的に小さな値になります。-INF が計算に含まれて、計算が止まってしまうこともありま

す。それを防止するアルゴリズムを入れています。この proc varlag の第１引数にはデータ行列が入り、第２引数にはオプションナンバーが入ります。１には AIC が、２には SC が、３には HQ が割り当てられ、それ以外の数がくるとエラーを返してそこで retp で抜けて正常終了して戻り値-1 を返すようにしてあります。その後は、要するに、vcv の計算と各パラメータ数から得られる情報値を計算させて、考えうるラグ数×情報値の種類３の行列 crit=zeros(maxlag,3);の中に値を１つ１つ入れていき、それをオプションナンバーの列の中で最小の値になるところの行数を最適ラグ次数としているのです。ｊループでラグｊ＝１から（自由度が変数の個数より下回らない）maxlag まで１つずつラグ数を増やしていってそれに対する３つの情報値を計算し、それを水平方向にマージして、crit[j,.]=aic~sc~hq;にってゼロと１行ずつ置き換えます。その crit の行列をラベルとともに画面表示させた後、lag=minindc(crit[.,opt]);によってオプションナンバーの列に対して最小値を与える行の行数を lag として、それを選択するべきラグ次数としてメッセージとともに画面表示しています。最後にリターンとして lag を返しています。途中、vcv の determinant が０に近づきすぎて計算が止まらないように、1e-16 より小さい値はすべて 1e-16 に置きかえる IF 文を挿入しています。これは maxlag のループで最大考えられるラグ数を設定してやっているかぎりでは、通常は引っかかるものではありません。Maxlag のループは、データ数が不足して deteminant が０に近づくのを防止するとともに、singular になることも防止する役割を果たしています。

Impulse Response Function

ここでは、VAR の係数β行列から定数項を除いたものを用いて、ある一定の期まで先の MA 表現に変換した後に、残差の Variance Covariance 行列 vcv を Cholesky 分解したものをかけ合わせることによって、Impulse Response を計算します。これまでは、簡単化のために、単位根の存在などを無視して（そういう意味でいい加減でしたが）、元データを階差などをとらずにそのまま使ってきましたが、ここでは 1 階の階差をとったデータ dDATA を使うことにします。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
data=Y~L~K;
ddata=data[2:rows(data),.]-data[1:rows(data)-1,.];
{b,se,t,vcv,projection,datahat}=varmain(ddata,2);
call impulse(b,vcv,2,10); /* Lag 2 and period 10 here are predetermined. */
```

```
proc impulse(b,vcv,lag,period);
    local i,j,k,ma,a0,irf,malag;
    malag=period-1;
/* MA Representation */
    k=cols(b);
    ma=zeros(malag*k,k);
    ma[1:k*lag,.]=b[2:k*lag+1,.];
    i=1;
    do while i<=malag-1;
        j=k*minc((i+lag)|malag);
        ma[i*k+1:j,.]=ma[i*k+1:j,.]+b[2:(j-i*k)+1,.]*ma[i*k-k+1:i*k,.];
        i=i+1;
    endo;
/* Cholesky Decomposition */
    a0=chol(vcv);
    print "A0=" a0;
/* Impulse Response Function */
    i=1;
    do while i<=malag;
        ma[(i-1)*k+1:i*k,.]=a0*ma[(i-1)*k+1:i*k,.];
        i=i+1;
    endo;
    irf=reshape((a0|ma),malag+1,k*k);
/* Display Results */
    print; "Impulse Responses";
    print "============================";
    print "Response      Period     Shock to";
    print " of                  ";;print/rz seqa(1,1,k)';
    print "============================";
    i=1;
    do while i<=k;
        print/lz i;;
        print/rz seqa(1,1,malag+1)~irf[.,(i-1)*k+1:i*k];
        i=i+1;
    endo;
```

```
    retp(irf);
endp;
```
（　この後に、proc varmain を置く。必須　）

プログラムでは、proc impulse の引数には MA 表現に直すときに使う係数βであるｂ、そして Cholesky 分解に使用する vcv、そして、ラグ数 lag、そして最後に当期も含めていくつ先までのショックを考えるかの period をもってきます。引数 period は当期も含んでいるので、マイナス１をしたものが MA 表現のラグ数になります。ここで、ma[1:k*lag,.]=b[2:k*lag+1,.];として、係数ｂの最初の行を取り除いたものを考えます。この場合、10 期先までのことを考えますから、あらかじめ ma=zeros(malag*k,k);として MA 表現の領域を確保します。そして、MA 表現での１期先のものから順に i=1 から maxlag-1 まで、j= j=k*minc((i+lag)|malag);のもとに、 ma[i*k+1:j,.]を設定してあるところに繰り返し足し合わせていきます。ma[i*k+1:j,.]=ma[i*k+1:j,.]+b[2:(j-i*k)+1,.]*ma[i*k-k+1:i*k,.]; によって足し合わせていきます。ここで、縦に行の３つごとに期が変わっています。最終的にこれを MA 表現とします。これに、vcv を Cholesky 分解したものを a0 として、３行ごとに同じ変数の系列の MA 表現がきていることに注意して、１期から maxlag 期まで順にかけ合わせずらして MA 行列に格納します。そして最終的に、当期の動きの a0 の行列を MA 行列の上に加えて、当期、そしてその次の期、…というふうな形にします。最初の部分も含めて、３行ごとに系列のショックは格納されていますから、reshape で３行ごとに横に展開するように、irf=reshape((a0|ma),malag+1,k*k);で形をかえてやります。この場合、ちょうど 10×３のブロックが横に３つ水平に並ぶことになります。最終的に、IRF を画面表示するには、seqa(1,1,malag+1)~irf[.,(i-1)*k+1:i*k]としてやって、１から３列目、４から６列目、７から９列目に I が１から k(この場合３)まで動く時、１から 10 までの数列とともに水平方向にマージして print 文でメッセージとともに画面表示されます。FED に使うために Impulse Response の入ったの行列 irf をリターンとして返しています。

なお、Cholesky 分解性質上、上の結果は変数の順番に依存して決まってきます。場合によっては、変数の順序を入れ替えると大きく結果がことなることもありますし、そうでないこともあります。複数の組合せを試してみる必要があります。なお、下の結果は、これまでの変数の順番と同じケースについての結果をそのまま表示しています。

画面表示

```
A0=
        12.009129         5.0685771         3.0582129
       0.00000000         3.1373825        0.45123564
       0.00000000        0.00000000         2.1571806
```

Impulse Responses

Response of	Period	Shock to 1	2	3
1				
	1	12.009129	5.0685771	3.0582129
	2	3.262761	1.3391903	1.0448884
	3	2.6706386	0.43676015	0.8690101
	4	1.5432079	0.15347411	0.41080624
	5	0.93928453	0.20545964	0.36341844
	6	0.81790153	0.35037202	0.37969389
	7	0.51601531	0.09124529	0.30526584
	8	0.3998483	-0.073973554	0.19231126
	9	0.27523921	-0.080844964	0.12007197
	10	0.25200225	0.039006665	0.12223269
	11	0.21499471	0.072582106	0.13172511
	12	0.17469265	0.015675432	0.10973525
	13	0.12938835	-0.038480014	0.07059106
	14	0.10468807	-0.026165426	0.049227693
	15	0.093400058	0.012654636	0.049432454
	16	0.082812336	0.022839526	0.051134964
	17	0.066631298	0.0023739879	0.041642683
	18	0.051166303	-0.013540354	0.027829781
	19	0.041916397	-0.0078460433	0.020566043
	20	0.037306072	0.0049761379	0.020277406
2				
	1	0	3.1373825	0.45123564
	2	-0.83168136	0.091781409	0.81126566
	3	0.14819014	-1.3469567	-0.10270766
	4	-0.79971368	-1.5642433	-0.61986806
	5	0.15529043	0.47134206	-0.14504711
	6	0.13552553	0.93709943	0.31283872
	7	0.20090861	0.22724721	0.29028032
	8	-0.082007748	-0.56841253	-0.053860136
	9	-0.076818281	-0.38417257	-0.17959897

10	0.020558692	0.16875632	-0.031439178
11	0.1003944	0.32855174	0.11973241
12	0.051846932	0.039435682	0.089294372
13	-0.016036022	-0.19103836	-0.022559742
14	-0.02376689	-0.11424119	-0.057283579
15	0.015151807	0.069688953	-0.0031845473
16	0.034687601	0.10498965	0.042398207
17	0.017092951	0.0038335011	0.027652741
18	-0.0054634083	-0.065455965	-0.0093435853
19	-0.0060122514	-0.032642011	-0.017800544
20	0.006889375	0.026801649	0.0012587001

3

1	0	0	2.1571806
2	3.6082502	1.2812402	2.0501556
3	0.60409045	-1.8827998	1.0187871
4	1.9081186	-0.78002006	0.62724597
5	1.0864187	0.020088263	0.66123954
6	1.5368863	0.7848684	0.94353023
7	1.0590116	0.12686029	0.78696205
8	0.85635134	-0.37704386	0.46415712
9	0.63105261	-0.31015256	0.27123365
10	0.61977025	0.11162952	0.31027517
11	0.56329997	0.23627029	0.36441935
12	0.45637088	0.039400491	0.30178324
13	0.33325176	-0.13665696	0.18106056
14	0.26942788	-0.088446067	0.12009615
15	0.24816978	0.045388448	0.13034313
16	0.22376539	0.077052939	0.14219456
17	0.17838036	0.0059076477	0.11434811
18	0.13401128	-0.046931559	0.071762337
19	0.10976787	-0.02575801	0.051450839
20	0.099794268	0.017887832	0.053962908

Forecast Error Decomposition

各変数のインパルス反応でのショックが予測誤差の分散にパーセンテージでどれだけ寄与しているのかを見るのが、Forecast Error Variance Decomposition です。基本的には、

上で求めた誤差の分散共分散行列のCholesky分解から得たショックの伝播をそのまま用います。

プログラム

```
    new; cls;
    load data[40,3]=d:datafile13.txt;
    Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
    data=Y~L~K;
    ddata=data[2:rows(data),.]-data[1:rows(data)-1,.];
    {b,se,t,vcv,projection,datahat}=varmain(ddata,2);
    irf=impulse(b,vcv,2,10);
    call fed(irf);

proc fed(irf);
  local n,k,i,irf2,decomp,s2,ratio,r,se;
  n=rows(irf); k=sqrt(cols(irf));
  irf2=reshape((irf.*irf),n*k,k);
  decomp=zeros(k,k); s2=zeros(n,k); ratio=zeros(n,k*k);
  i=1;
  do while i<=n;
    decomp=decomp+irf2[(i-1)*k+1:i*k,.];
    s2[i,.]=sumc(decomp)';
    r=decomp./s2[i,.]*100;
    ratio[i,.]=reshape(r',1,k*k);
    i=i+1;
  endo;
  se=sqrt(s2);
/* Display Results */
  print; "Forecast Error Decomposition";
  print "============================";
  print " For          Period";
  print "Variable                    Std.Err. ";;print/rz seqa(1,1,k)';
  print "============================";
  i=1;
  do while i<=k;
    print/lz i;;
```

```
        print/rz seqa(1,1,n)~se[.,i]~ratio[.,(i-1)*k+1:i*k];
        i=i+1;
    endo;
    retp(ratio);
endp;
```

（　この後に、proc impulse および proc varmain を置く。必須。　）

プログラムでは、IRF で得たもとの結果行列 irf をインプットとして使って、まずそれを要素ごとに２乗してやって分散の形にします。それを、(i-1)*k+1:i*k ごと、すなわちここでは k=3 ですから i を１から順に増やすことになりますから、この場合３つ飛ばしに足し合わせていきます。そのおのおのの段階までの累積和を分散として s2 に格納していきます。各列の累積和は１列でててきますから、それを転置させることも忘れずにします。それぞれの値をこの累積和で要素対要素で割ると、分散に対する寄与率がでてきます。パーセンテージ表示にするのが FED の慣習ですから、100 をかけています。ｋ×ｋで出てくる結果をそれぞれの分散に対する値に整列させるために、もともとの結果を転置させたものを１×(k×k)に変換て、変数 ratio それぞれの行に格納していき完成です。これを各行の（各ｎ期先の）分散とマージして最後画面表示させています。なお、IRF のときと同様、この場合の ratio もｎ×３のそれぞれの結果のかたまりが水平方向にくっついててでてきますから、それらのおのおのを取り出して表示させています。

画面表示

```
Forecast Error Decomposition
============================
 For        Period
Variable                Std.Err.                1                2                3
============================
 1
                 1       12.009129              100                0                0
                 2        12.98368          91.866477       0.41031588        7.7232072
                 3       13.270085          91.994073       0.40526626         7.600661
                 4       13.518768          89.943751       0.74043428        9.3158143
                 5       13.595726          89.405687       0.74512184        9.8491916
                 6       13.707411           88.31074       0.74280446        10.946455
                 7       13.759406          87.785215       0.75852164        11.456263
                 8        13.79207           87.45395       0.75846856        11.787582
                 9       13.809456          87.273604       0.75965432        11.966742
```

	10	13.825669	87.102262	0.75809485	12.139643
	11	13.839174	86.956484	0.76187859	12.281638
	12	13.847896	86.862897	0.76232097	12.374782
	13	13.852519	86.813656	0.76194626	12.424398
	14	13.855555	86.781326	0.76190664	12.456767
	15	13.8581	86.753993	0.76174633	12.484261
	16	13.860197	86.731311	0.76214216	12.506547
	17	13.861516	86.717122	0.76214923	12.520729
	18	13.862259	86.709185	0.76208304	12.528732
	19	13.862758	86.703854	0.76204696	12.534099
	20	13.863169	86.699436	0.76202646	12.538538
2					
	1	5.9610102	72.299015	27.700985	0
	2	6.2431622	70.513001	25.275353	4.2116463
	3	6.67286	62.15246	26.199541	11.647999
	4	6.8997031	58.182318	29.644942	12.17274
	5	6.9188644	57.948685	29.945062	12.106253
	6	7.0347436	56.303361	30.741147	12.955492
	7	7.0401476	56.233756	30.798163	12.968081
	8	7.0735002	55.715641	31.154153	13.130205
	9	7.0911723	55.451285	31.292572	13.256143
	10	7.0941656	55.407523	31.322758	13.269719
	11	7.1060695	55.232477	31.431675	13.335848
	12	7.1063055	55.229296	31.432667	13.338036
	13	7.1102904	55.170337	31.469633	13.36003
	14	7.1118062	55.148175	31.482023	13.369802
	15	7.1123037	55.140776	31.48722	13.372004
	16	7.1135326	55.122758	31.498125	13.379117
	17	7.1135365	55.122709	31.49812	13.379172
	18	7.1140053	55.115806	31.502434	13.38176
	19	7.1141311	55.113977	31.503425	13.382598
	20	7.1142059	55.112869	31.504182	13.382949
3					
	1	3.7695766	65.81885	1.432919	32.748231
	2	4.4902995	51.800659	4.2740394	43.925301
	3	4.6868373	50.985205	3.9711226	45.043673

4	4.78674	49.615755	5.4840372	44.900208
5	4.8480129	48.931452	5.4358042	45.632744
6	4.9634176	47.267691	5.5832307	47.149078
7	5.0430421	46.153271	5.7396373	48.107092
8	5.0682936	45.838497	5.6938802	48.467623
9	5.0801418	45.680796	5.7923364	48.526868
10	5.0911728	45.5407	5.7710765	48.688224
11	5.1073016	45.320041	5.7896434	48.890316
12	5.1181655	45.17382	5.7955293	49.030651
13	5.1219032	45.126906	5.7890137	49.08408
14	5.1238677	45.10154	5.7970742	49.101386
15	5.1257647	45.077464	5.7928228	49.129713
16	5.1281668	45.045186	5.7942326	49.160582
17	5.1296851	45.025115	5.7937092	49.181176
18	5.130271	45.017774	5.7927175	49.189509
19	5.1306011	45.013588	5.7931759	49.193236
20	5.1309251	45.009465	5.7924503	49.198085

表の見方は、ｎ期先の予測誤差の分散が２列目に示されていて（通常は徐々に大きくなる傾向にある）その後に 100％のうち、その変数の予測誤差の分散の寄与のパーセンテージが示されている。この場合、３変数で横に３つ足し合わせると常に合計は 100％になります。

Impulse Response Graph

ここでは、上で計算した IRF と FED の一般形のグラフ化を試みます。基本的には、

```
xy(seqa(1,1,rows(irf)),zeros(rows(irf),1)~irf[.,(i-1)*k+1:i*k]);
```

によって、ｘ軸側に１からのシークエンスの座標の数列を代入し、ｙ軸側には基準線となる零ベクトルと irf の行列のうちのブロックごとのｋ列の値を代入し、ｘｙグラフを作成します。IRF のところで説明したように、irf[.,(i-1)*k+1:i*k])とは、k=3 のとき、irf[.,1:3]、irf[.,4:6]、irf[.,7:9]をそれぞれ一般形にした形です。これ以外の部分は、変数の実際の名前を含んだファイル名で保存し、また、変数の実際の名前を含んだグラフタイトルをつけ、凡例を設定している箇所です。文字列の演算には、文字自身は引用符で囲んで代入することと、足し算の場合、文字という意味で＄のしるしをつけた＄＋という演算記号を使うことに注意してください。下のプログラムでは、irfgraph の proc の最終引数のところに文字列 name を設定するようにしています。これにより、name={"Y","L","K"};などというように proc の外部で設定された変数の実際の名前の入っている文字列の列変数を扱えるように

しています。なお、第２引数のｋは変数の個数です。名前の文字列の列の行数と一致している必要があります。

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
data=Y~L~K;
{b,se,t,vcv,projection,datahat}=varmain(data,2);
irf=impulse(b,vcv,2,20);
name={"Y","L","K"};
call irfgraph(irf,3,name);

proc(0)=irfgraph(irf,k,name);
    local i,j,st;
    library pgraph;
    i=1;
    do while i<=k;
        graphset;
        _ptek="d:irf"$+name[i]$+".tkf";
        j=1; st="Zero Line";
        do while j<=k;
            st=st$+"¥000"$+"to "$+name[j];
            j=j+1;
        endo;
        _plegctl=1;
        _plegstr=st;
        title("Impulse Response of "$+name[i]);
        xy(seqa(1,1,rows(irf)),zeros(rows(irf),1)~irf[.,(i-1)*k+1:i*k]);
        i=i+1;
    endo;
endp;
```

（　この後に、proc impulse および proc varmain を置く。必須。　）

プログラムの proc 内部の i ループでは、それぞれの変数に対するグラフを１枚ずつ作成して、同時に_ptek="d:irf"$+name[i]$+".tkf";で、irf 変数名.tkf というそれぞれ独自の名前を

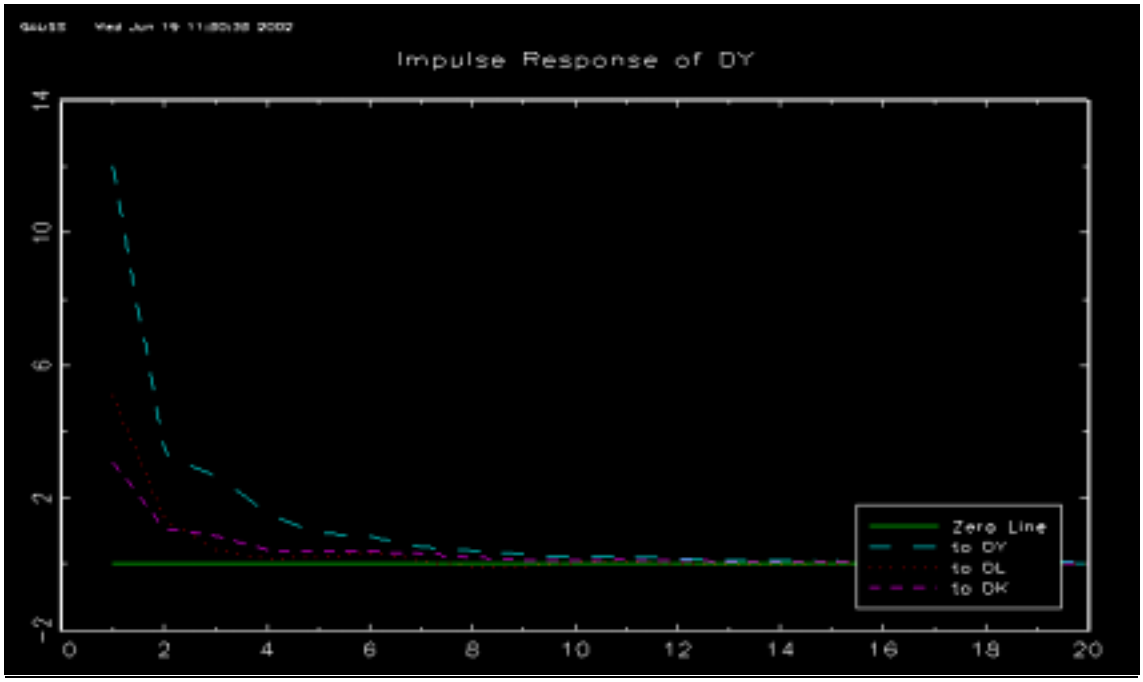
つけて、GAUSS のグラフィックファイルとして保存させています。ここで、

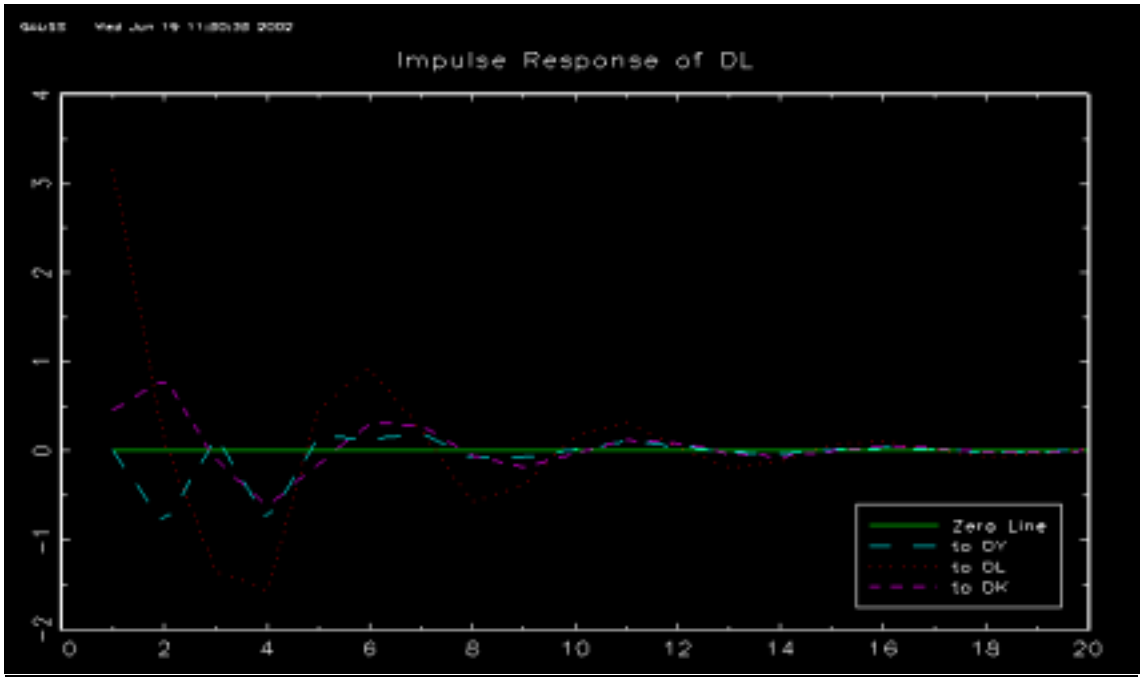
_ptek=”ファイル名”;

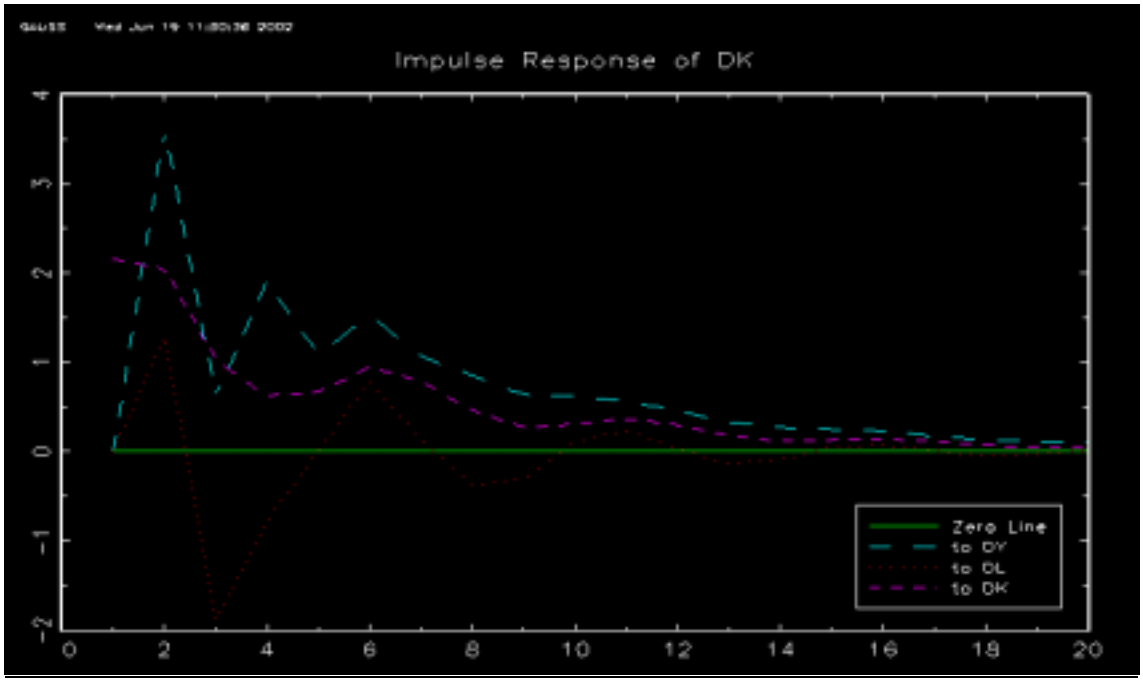
はグラフをファイルとして保存する方法です。これは pgraph ライブラリのグローバル設定の形をとるので複数のグラフを作成して、上のようにそれぞれ異なる名前をつけるのには、その都度、graphset;でもってグローバル変数を初期化する必要があります。そういう意味で、必ず graphset;の命令はループの内側に入れる必要があります。引用符の中の d:はディレクトリ名、その後ファイル名を指定するときに、irf"$+name[i]$+".tkf というように、文字列の足し算をしてファイル名の一部を name という文字列の列変数の i 行目（番目）から取り出して、前の irf と後ろの.tkf と合成しています。繰り返しになりますが、＄のマークは文字列の演算を表す記号です。

グローバル変数の_plegctlのところは、任意の非零のスケラーの値を設定すると（ここでは１）凡例が右隅下に出ます。_plegstr=は文字列を凡例として設定するものです。なお、この場合のそれぞれの凡例の区切りには、¥000を用います。（なお、ここで¥の記号は英語のバックスラッシュに相当する部分の日本語システムでの半角表示に相当します。）つまり凡例ごとに入れる文字を¥000で区切ればよいことになります。その_plegstr=に設定する文字列を設定するのに j ループを用いて、最初にZero Lineという文字列を j ループの外に置いて、区切りの制御文字列と文字列の列変数nameの j 番目の名前の文字列をその都度足し合わせて一般化しています。同様に、タイトルのところも既存のメッセージ部分と文字列の列変数を＄＋で足し合わせて一続きの文字列にしてタイトルとしています。

グラフ表示


GAUSS Wed Jun 19 11:80:38 2002
Impulse Response of DY
14
10
6
2
-2
0
2
4
6
8
10
12
14
16
18
20
Zero Line
to DY
to DL
to DK


GAUSS Wed Jun 19 11:80:38 2002
Impulse Response of DL
4
3
2
1
0
-1
-2
0
2
4
6
8
10
12
14
16
18
20
Zero Line
to DY
to DL
to DK



グラフの見方は、ある変数に与えられたショックが時とともにどのようにそれぞれの変数に伝播していっているかを示しています。なお、重要な点ですが、このデータの場合、階差をとったものではなくて、もとの系列Y、L、Kなどをそのまま使うとＩＲＦは発散してしまいます。

Forecast Error Decomposition Graph

プログラム

```
new; cls;
load data[40,3]=d:datafile13.txt;
Y=data[2:40,1]; L=data[2:40,2]; K=data[2:40,3];
data=Y~L~K;
ddata=data[2:rows(data),.]-data[1:rows(data)-1,.];
{b,se,t,vcv,projection,datahat}=varmain(ddata,2);
irf=impulse(b,vcv,2,20);
ratio=fed(irf);
name={"DY","DL","DK"};
call fedgraph(ratio,3,name);

proc(0)=fedgraph(ratio,k,name);
    local i,j,st;
    library pgraph;
```

```
    i=1;
    do while i<=k;
        graphset;
        _ptek="d:fed"$+name[i]$+".tkf";
        j=1; st="100%";
        do while j<=k;
            st=st$+"¥000"$+name[j];
            j=j+1;
        endo;
        _plegctl={2 5 3 2};
        _plegstr=st;
        title("Forecast Error Decomposition of "$+name[i]);
        xy(seqa(1,1,rows(ratio)),100*ones(rows(ratio),1)~ratio[.,(i-1)*k+1:i*k]);
        i=i+1;
    endo;
endp;
```

FED のグラフ化のプログラムと上のプログラムの違いは、_plegctl={2 5 3 2};というふうに、

スケラー１ではなくて、行ベクトルを設定しているところにあります。これにより右下隅以外のどこにでも凡例を入れられます。すなわち、

_plegctl={第１変数　第２変数　第３変数　第４変数};

第１変数　１：プロット軸座標　２：インチ座標　３：ピクセル座標

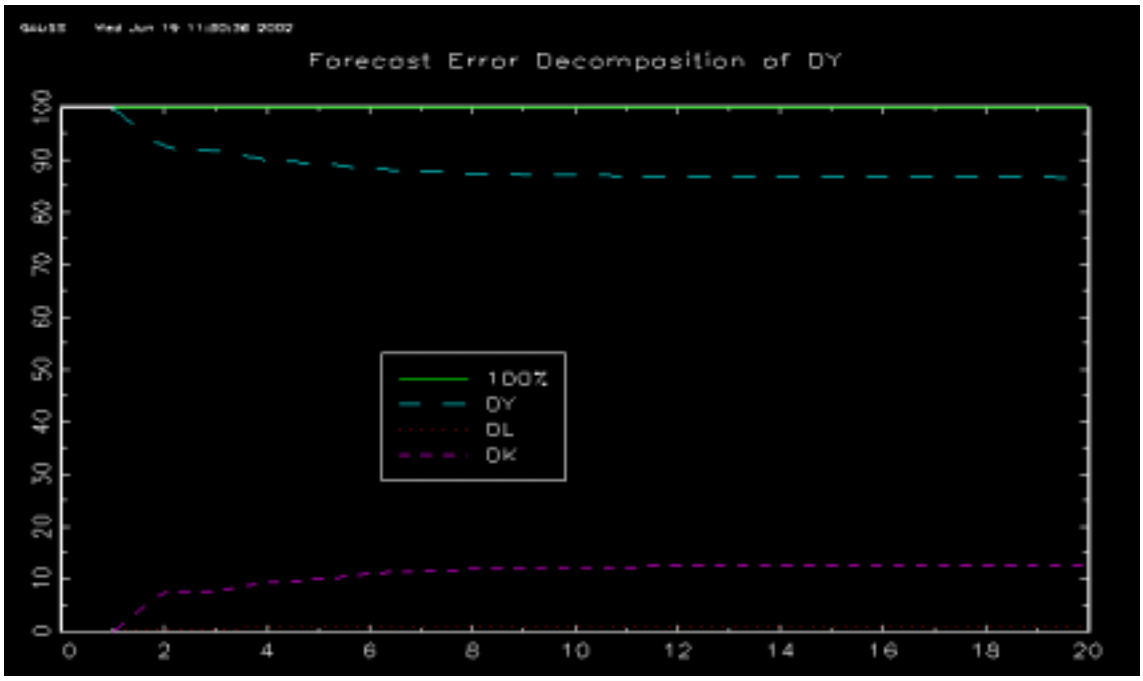
第２変数　１から９までのフォントサイズ（デフォルトは５）

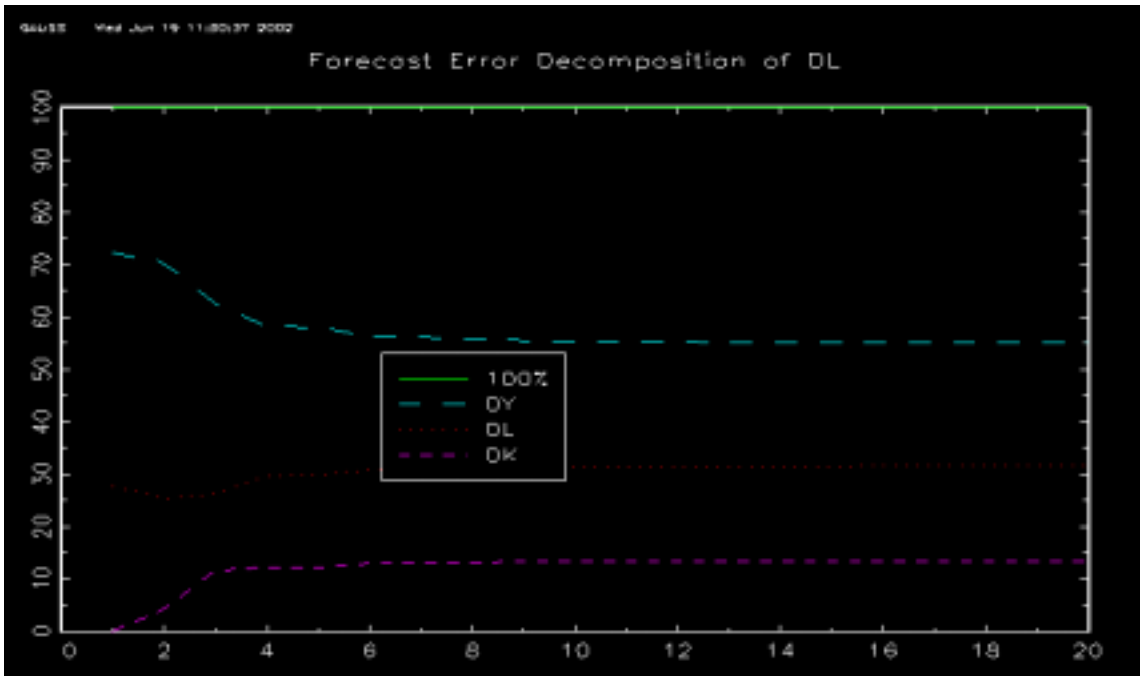
第３変数　凡例左下のｘ座標

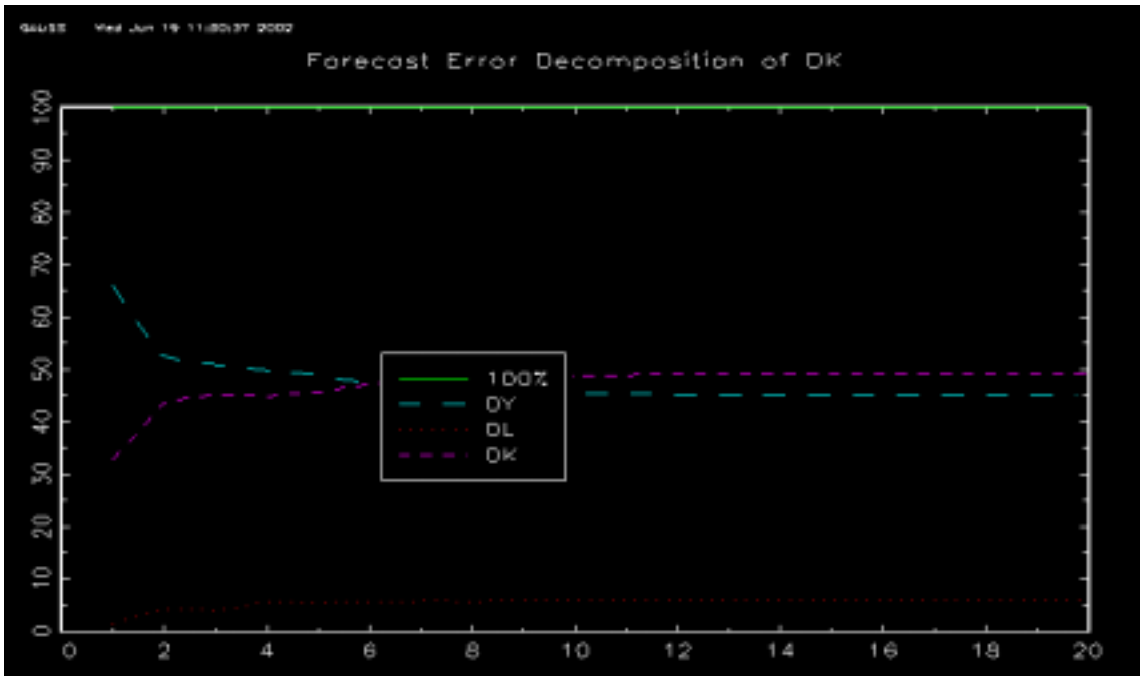
第４変数　凡例左下のｙ座標

したがって、_plegctl={2 5 3 2};の意味するところは、インチ座標で、フォントサイズ５、凡例の左下の座標が(3,2)ということになる。

グラフ表示


Forecast Error Decomposition of DY
100
90
80
70
60
50
40
30
20
10
0
0
2
4
6
8
10
12
14
16
18
20
100%
DY
DL
DK


Forecast Error Decomposition of DL
100
90
80
70
60
50
40
30
20
10
0
0
2
4
6
8
10
12
14
16
18
20
100%
DY
DL
DK



ここで、本章で扱った行列の reshape と print の際に横方向にブロックごとに分割して print した際に用いた行列整形のテクニックを小さなプログラムを使ってまとめておこう。

プログラム

```
new; cls;
m={1 1 1,
   2 2 2,
   3 3 3,
   1 1 1,
   2 2 2,
   3 3 3};
format /rz 2,1; print "m=" m;
n=reshape(m,2,3*3);
print "RESHAPE m to 2 by 9:      n=reshape(m,2,3*3)";
format /rz 2,1; print "n=" n;

k=3;
print "Here k=3 and i=1,2,3";
print "BLCOK BY BLOCK of          n[.,(i-1)*k+1:i*k]";
i=1;
print "i=1" n[.,(i-1)*k+1:i*k];
i=2;
```

```
    print "i=2" n[.,(i-1)*k+1:i*k];
    i=3;
    print "i=3" n[.,(i-1)*k+1:i*k];
```

画面表示

```
m=
 1   1   1
 2   2   2
 3   3   3
 1   1   1
 2   2   2
 3   3   3
RESHAPE m to 2 by 9:      n=reshape(m,2,3*3)
n=
 1   1   1   2   2   2   3   3   3
 1   1   1   2   2   2   3   3   3

Here k=3 and i=1,2,3
BLCOK BY BLOCK of          n[.,(i-1)*k+1:i*k]
i=1
 1   1   1
 1   1   1
i=2
 2   2   2
 2   2   2
i=3
 3   3   3
 3   3   3
```

上のプログラムでは、前半では３行ごとに異なる係数とされるグループを横に展開するために、T×(3×3)の行列に変換するものです。T の部分は、ここでは２になりますが、実際にもっと長い場合には、もとの行列のディメンションの掛け算の答えをこの場合の３×３の９で割ったものが T のところにきます。これにより、何行おきかに入り組んでいる行列を横方向に展開します。後半は、そうして展開された行列が３つのブロックに分割できると見たてて、i=1 から順番に３までケースについて一般化して分割して表示するものです。なお、実際のプログラムでは、i ループで変数の個数（この場合３）だけまわすことになり

ます。