付録２　EViews　プログラムについて

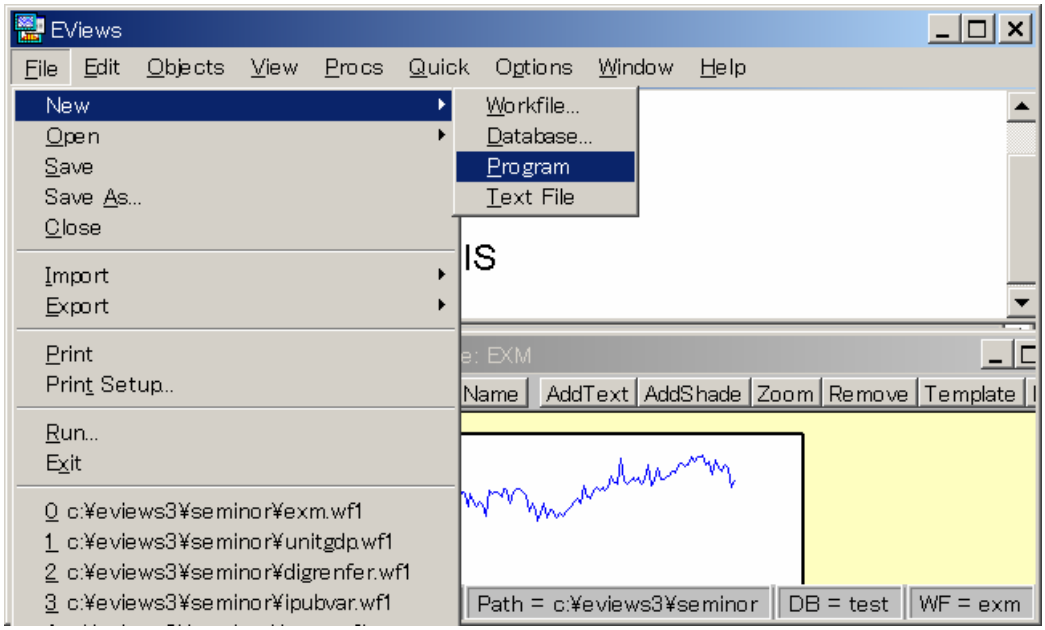
プログラムの操作法

　ＥViews では、プログラムを実行することによってもさまざまな推計ができる。プログラムは一度作っておけば、推計期間や変数を変えるのも楽だし、一つのプログラムで表の作成や数値の出力まで終えることもできる。

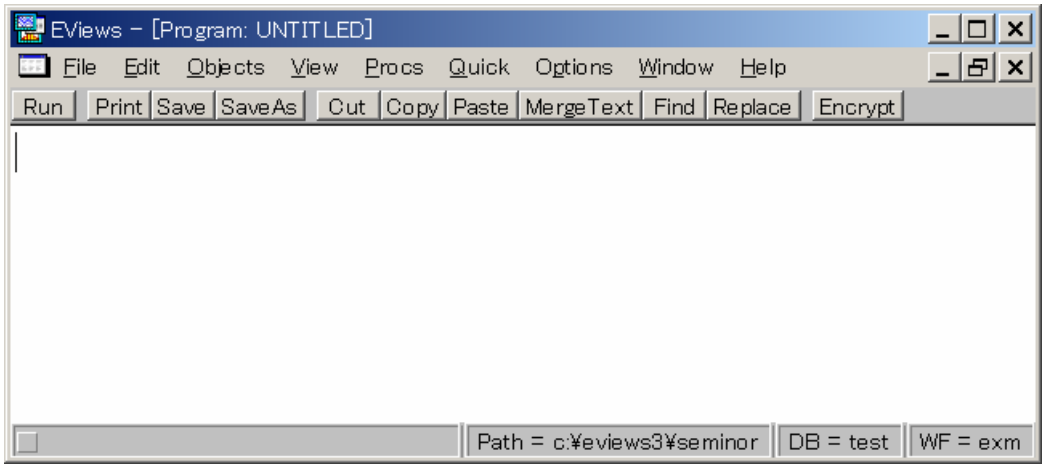
　プログラムファイルを作って、「Run」という支持を出せばプロラムが走る。ワークファイルの呼び出しなどをプログラムに含めることもできる。

プログラムファイルの作成

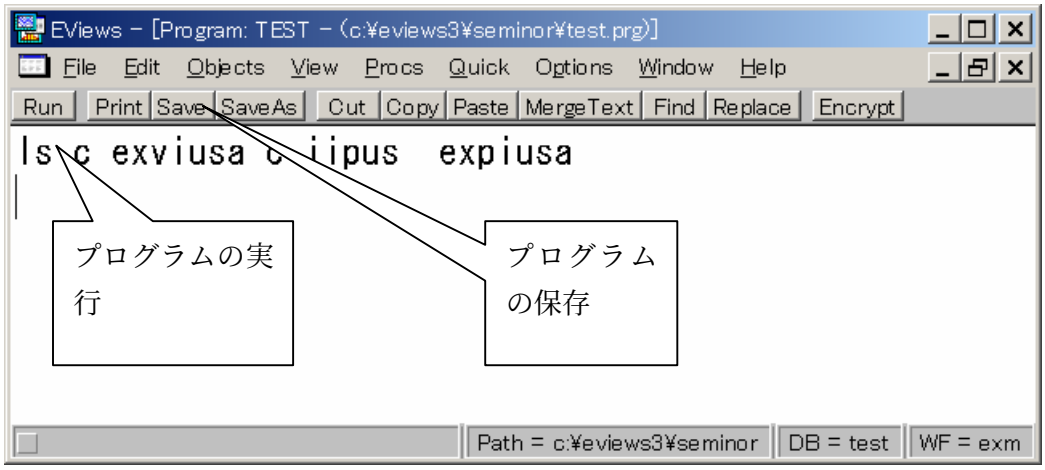
[File]→[New]→[Program]を選ぶ。



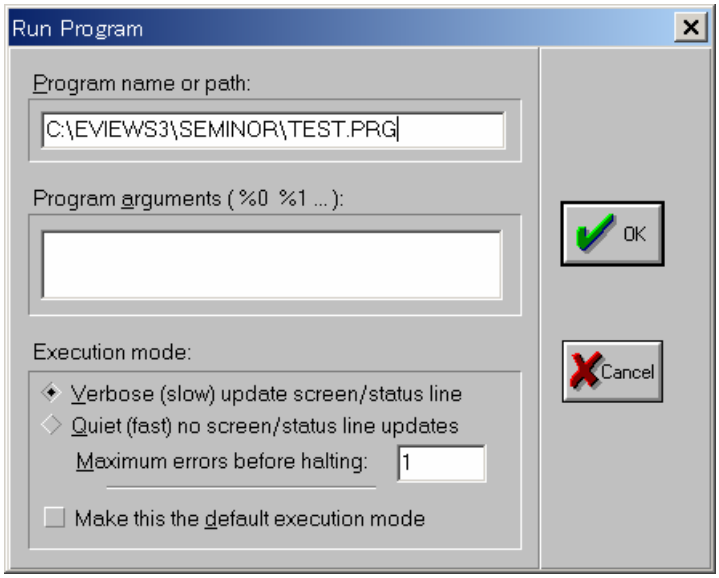
プログラムファイルが開くので、ここにコマンドを書いていく。



プログラムファイルを作ったら、[save]をクリックしてファイルまず保存する。



プログラムファイルを保存したら、このファイルを実行する。[Run]をクリックすると、次のようなウインドウが出てくるが、何も変更せずに、ＯＫを押す。複雑なプログラムを書いて、とりあえず、何ヶ所かエラーがあっても最後まで実行させたいときは、「Maximum errors before halting」に１以上の数字を入れる。何個エラーがあったらとまるかを指定できる。



プログラムの書き方

簡便法

コマンドをそのままファイルに書き込めば、それを実行する。
たとえば、プログラムファイルに
ls cp95 c gdp95
と入力して実行すれば、EViews は最小二乗法を実行する。

オブジェクトを定義して、実行する

データ、方程式、表、行列といったオブジェクトに名前を付けて、その後数値などを入れるというのが基本的な考え方。
（例１）

```
equation eqcp95                 ←eqcp95 という名前の方程式を作成する。
eqcp95.ls cp95 c gdp95          ←方程式 eqcp95 で最小時二乗法を使う。
```

(例２)

```
series cp95                     ←cp95 という名前のデータを作成する。
cp95=1                          ←cp95 の全期間を１にする。
```

オブジェクトに名前をつけるのと、オブジェクトの操作は動じにできる。これらの系列を使ったほうがプログラムは簡潔になる。

```
series cp95=1               ←cp95 というデータを作り、全期間に１を入れる。
table   out(1,1)=1          ←out という表を作り、１行１列目に１を入れる。
equation   eqcp95.ls cp95 c gdp95 ←eqcp95 という方程式名で最小二乗法を使う。
```

ワークファイルの呼び出し

ワークファイルの呼び出しもプログラムに入れることができる。同じディレクトリーにあれば、ワークファイル名を次のように指定する。

LOAD ワークファイル名

プログラムで実行しようするワークファイルを開いておくと、新しいワークファイルが２重に開くことになる。ワークファイルのロードも含めてプログラムを実行するときは、実行する前にワークファイルを全部閉じておくのが望ましい。

表の作り方

まず、表の名前を宣言（declear）する。
Table (table の名前)

表のセルの位置に数字や文字列を入れていく。

(table の名前)(1,1)=(スカラー)
(table の名前)(1,1)=” 文字列 “

表中のセルの位置

(1,1)	(1,2)	‥
(2,1)	(2,2)	
：		

表に水平な 2 重の区切り線を入れる場合は、setline コマンドを使う。
setlin(テーブル名、行数)
setline(tab,2)というコマンドは、tabという表の2行目に水平線を入れるという意味である。水平線は、文字列が入力されているところまで引かれるので、表の入力が終わった最後の段階のコマンドとする。最初に指定すると、1列目しか水平線が引かれない。

（例）
Table tab1
tab1(1,1)=”coefficient”
tab1(1,2)=”t-value”
show tab1

推計結果を表にする
テーブル名を tab にした場合
table tab

equation eq1.ls cp95 c gdp95

tab(1,1)="coef(1)"
tab(1,2)="t-stats"
tab(1,3)="p-stats"
tab(1,4)="coef(2)"
tab(1,5)="t-stats"

```
tab(1,6)="p-stats"
tab(1,7)="Rbar2"
tab(1,8)="D.W."
setline(tab,2)

tab(3,1)= @coefs(1)
tab(3,2)=@tstat(1)
tab(3,3)=@tdist( eq1.@tstats(1),@regobs-@ncoef)
tab(3,4)= @coefs(2)
tab(3,5)=@tstat(2)
tab(3,6)=@tdist( eq1.@tstats(2),@regobs-@ncoef)
tab(3,7)=eq1.@rbar2
tab(3,8)=eq1.@dw

show tab
```

（２）置換変数

「%x」という記号を使うと、同じ操作をさまざまな変数に変えて使うことができる。次の例は%X に CPI を入れて表を作成するコマンドである。%ＸをＧＤＰに置き換えればＧＤＰについての作業ができる。

```
%x="cpi"
delete tab{%x}1
table tab{%x}1
```

（３）for next 文

```
for !j=1 to 18
：
gdp!j
:
Next
```

for と next にはさまれた部分では、!j に、１から１８までの数字を順番に入れる。

```
%x=”GDP”
```

```
equation eq!j.ls    {%x}!j-0.16-0.72*{%x}act(-1)    dum    dum*{%x}act(-1)
scalar coef1!j=eq!j.@coefs(1)
scalar coef2!j=eq!j.@coefs(2)
scalar   rbar2!j=eq!j.@rbar2
scalar dw!j=eq!j.@dw
scalar coef1p!j=@tdist( eq!j.@tstats(1),@regobs-@ncoef)
scalar coef2p!j=@tdist( eq!j.@tstats(2),@regobs-@ncoef)
scalar se!j=eq!j.@se

tab{%x}1(!j,1)=coef1!j
tab{%x}1(!j,2)=coef2!j

tab{%x}1(!j,3)=rbar2!j
tab{%x}1(!j,4)=dw!j
tab{%x}1(!j,5)=se!j

tab{%x}1(!j,6)=coef1p!j
tab{%x}1(!j,7)=coef2p!j

eq!j.wald   c(1)=c(2)=0

next
show tab{%x}1
```

（４）サブルーチン

```
%x="gdpr"
delete tab
table tab
!counter=1

subroutine maketable
scalar stder=@stdev(e{%x}18)
scalar stdact=@stdev({%x}act)
scalar stdavg=@stdev({%x}18)
tab(!counter,1)=stder
```

```
tab(!counter,2)=stdact
tab(!counter,3)=stdavg
tab(!counter,4)=!counter
!counter=!counter+1
endsub

call maketable
%x="cp"
call    maketable
%x="ihp"
call maketable
%x="iop"
call maketable
%x="ipub"
call maketable
%x="ext"
call maketable
%x="mxt"
call maketable
%x="gdp"
call maketable
%x="cpi"
call maketable
%x="wpi"
  call maketable
%x="iip"
  call maketable
%x="frexda"
  call maketable
%x="bopcrnt"
  call maketable

show tab
```

（５）その他プログラム例

アーモンラグの制約条件を変えたときの出力法

```
load exm
equation none.ls  exviusa c iipus  pdl(expiusa,6,2,0)
equation  near.ls  exviusa c iipus  pdl(expiusa,6,2,1)
equation far.ls  exviusa c iipus  pdl(expiusa,6,2,2)
equation both.ls  exviusa c iipus  pdl(expiusa,6,2,3)
show none
show near
show far
show both
```

単位根検定で、誤差項のラグの次数を変えた時のプログラム

```
load unitgdp
for !j=0 to 8
gdp95.uroot(c,!j)
freeze gdp95.uroot(c,!j)
next
for !j=0 to 8
gdp95.uroot(t,!j)
freeze gdp95.uroot(t,!j)
next
for !j=0 to 8
gdp95.uroot(n,!j)
freeze gdp95.uroot(n,!j)
next
```

パネルデータの推計

```
pool01.ls(cx=f)  growth1? ligdp1? enrollg? aid? fdi? export? work? credit? credit?*infl?
tab(1,1)="coef(1)"
tab(1,2)="t-stats"
tab(1,3)="p-stats"
tab(1,4)="coef(2)"
tab(1,5)="t-stats"
```

```
tab(1,6)="p-stats"
tab(1,7)="Rbar2"
tab(1,8)="D.W."
setline(tab,2)

tab(3,1)= @coefs(1)
tab(3,2)=@tstat(1)
tab(3,3)=@tdist( eq1.@tstats(1),@regobs-@ncoef)
tab(3,4)= @coefs(2)
tab(3,5)=@tstat(2)
tab(3,6)=@tdist( eq1.@tstats(2),@regobs-@ncoef)
tab(3,7)=eq1.@rbar2
tab(3,8)=eq1.@dw

show tab
```

見せかけの相関

```
smpl @first @first
series y=10
smpl @first+1 @last
y=0.5+y(-1)+nrnd
smpl @first @first
series x=100
smpl @first+1 @last
x=0.1+x(-1)+nrnd
smpl @all
equation eq1.ls y c x
equation eq2.ls d(y) c d(x)
show eq1
show eq2
```

単位根検定

```
gdp95.uroot(none,adf,save=mnone)
gdp95.uroot(const,adf,save=mconst)
gdp95.uroot(trend,adf,save=mtrend)
matrix(8,3) out
```

```
colplace(out,mnone,1)
colplace(out,mconst,2)
colplace(out,mtrend,3)
```

エラーコレクションモデル

```
equation eq1.ls cp95 c gdp95
eq1.makeresids res01
equation eq2.ls d(cp95)  d(gdp95) res01(-1)
show eq1 eq2
```

グレンジャーの因果関係

```
equation eqgdp951.ls gdp95 c  gdp95(-1) gdp95(-2) m2(-1) m2(-2) rblav(-1) rblav(-2)
scalar ssr1=eqgdp951.@ssr
scalar  regobs=@regobs
scalar ncoef=@ncoef
equation eqgdp952.ls  gdp95 c gdp95(-1) gdp95(-2) rblav(-1) rblav(-2)
scalar ssr2=eqgdp952.@ssr
scalar F=((ssr2-ssr1)/2)/(ssr1/(regobs-ncoef))
scalar  W=f*2
scalar  p=@chisq(w,2)
table  test
test(1,1)=w
test(1,2)=p
show test
```

プログラム

期間を変えて推計し、そのときの係数を記録するというプログラムは以下の方法で作成できる。何度も繰り返して推計するばあいにはプログラムがあると重要である。さまざまなバリエーションが考えられるが、基本的には次のようなプログラムを拡大させていけばよい。

table tab	tab というテーブル（表）を作成。
smpl 1980 2001	期間を 1980 から 2001 とする。
equation eq1.ls cp95 c gdp95	最小二乗法（ls）行い方程式名を”eq1”とする。
tab(1,1)=@coefs(2)	表 tab の 1 行 1 列目に推計した係数の 2 番目（gdp95）にかかる係数を入れる。

```
smpl 1980 2002
equation eq2.ls cp95   c gdp95
tab(2,1)=@coefs(2)
```

ステップ・ワイズ・チャウテストの例
eq1 変化前の推計
eq2 変化後の推計
eq3 全期間の推計
　構造変化のあった時点を１期ずつずらして推計する。変化前の推計期間のうち最も短い場合は、サンプルの初期 t=1 から　t=1+1 期での推計である。構造変化のある期は、t=1+２となる。
　推計終期を t+2,t+3 と順に伸ばしていく。変化前の推計期間のういち最も長いのは、サンプルの初期から、サンプルの終期の 2 期前までである。サンプル数を n とすると、t+(n-1)がサンプルの終期なので、２期前までだと、t+n-3 期となる。

```
scalar obs=@obs(gdp95)          ←サンプル数を計算

for !J=1 to obs-3               ←変化前の推計終期を t+k と表したときの k に当たる。
smpl @first   @first+{!j}  ←変化前のサンプル期間
equation eq1.ls gdp95 c @trend   ←変化前の推計
scalar ssra=@ssr                    ←eq1 の残差二乗和

smpl @first+{!j}＋１   @last   ←変化後のサンプル期間
equation eq2.ls gdp95 c @trend   ←変化後の推計
scalar ssrb=@ssr                    ←eq2 の残差二乗和

smpl @all                           ←すべてのサンプル期間
equation eq3.ls gdp95 c @trend      ←全期間での推計
scalar   ssr=@ssr                   ←eq3 の残差二乗和

scalar   f=((ssr-(ssra+ssrb))/@ncoef)/((ssra+ssrb)/(@regobs-2*@ncoef)) ← f 値の計算。
```
制約の数は、説明変数の個数と一致する。自由度は、サンプル数から、２期間の説明変数の個数の２倍である。

```
table out                           ←テーブルの名前を out にする。
```

```
out({!j},1)=@otod({!j}+2)                ←構造変化のある期を表示。
out({!j},2)=f
out ({!j},3)=@fdist(f,2,@regobs)
out({!j},4)=@regobs
out({!j},5)=@ncoef
next

show out
```

最尤法

プログラムを書く場合は次のようにする。最小二乗法で計算した結果を初期値に採用し、それを使って最尤法を使う場合は次のようなプログラムになる。

「logl」で最尤法を適用する対数尤度を定義する。ウインド方式との違いは、appendというコマンドを使うところである。appendは追加すると言う意味である。[ml]というコマンドで、最尤法による推計を実行する。

```
Equation eq1.ls cp95 c gdp95
C(1)=eq1.@coef(1)
C(2)=eq1.@coef(2)
C(3)=eq1.@se^2

equation ls.eq1    cp95 c gdp95
c(1)=@coefs(1)
c(2)=@coefs(2)
c(3)=@se

logl    mlcp95
mlcp95.append @logl logl1
mlcp95.append res=cp95-c(1)-c(2)*gdp95
mlcp95.append se=c(3)
mlcp95.append logl1=log(@dnorm(res/se))-log(se^2)/2

mlcp95.ml
```

ＡＲＩＭＡモデルの次数探索プログラム

ARMA モデルをさまざまに変えてＡＩＣを表示させ、最も低いＡＩＣを探す。

	noMA	MA(1)	MA(2)	MA(3)
AR(1)	7.215263	6.761484	6.466351	6.484631
AR(2)	6.380968	6.395975	6.412498	6.407816
AR(3)	6.405320	6.373856	6.375635	6.383247
AR(4)	6.393103	6.385140	6.326474	6.349493

```
table tab
%x="x"
tab(1,2)="noMA"
tab(1,3)="MA(1)"
tab(1,4)="MA(2)"
tab(1,5)="MA(3)"
tab(3,1)="AR(1)"
tab(4,1)="AR(2)"
tab(5,1)="AR(3)"
tab(6,1)="AR(4)"

equation eq1.ls   {%x}    c   {%x}(-1)
tab(3,2)=eq1.@aic
equation eq11.ls   {%x}    c   {%x}(-1)   ma(1)
tab(3,3)=eq11.@aic
equation eq12.ls   {%x}    c   {%x}(-1) ma(1) ma(2)
tab(3,4)=eq12.@aic
equation eq13.ls   {%x}    c   {%x}(-1) ma(1) ma(2) ma(3)
tab(3,5)=eq13.@aic

equation eq2.ls   {%x}    c   {%x}(-1)   {%x}(-2)
tab(4,2)=eq2.@aic
equation eq21.ls   {%x}    c   {%x}(-1)   {%x}(-2)   ma(1)
tab(4,3)=eq21.@aic
equation eq22.ls   {%x}    c   {%x}(-1) {%x}(-2) ma(1) ma(2)
tab(4,4)=eq22.@aic
equation eq23.ls   {%x}    c   {%x}(-1) {%x}(-2) ma(1) ma(2) ma(3)
tab(4,5)=eq23.@aic
```

```
equation eq3.ls  {%x}   c  {%x}(-1)  {%x}(-2) {%x}(-3)
tab(5,2)=eq3.@aic
equation eq31.ls  {%x}   c  {%x}(-1)  {%x}(-2) {%x}(-3) ma(1)
tab(5,3)=eq31.@aic
equation eq32.ls  {%x}   c  {%x}(-1) {%x}(-2) {%x}(-3) ma(1) ma(2)
tab(5,4)=eq32.@aic
equation eq33.ls  {%x}   c  {%x}(-1) {%x}(-2) {%x}(-3) ma(1) ma(2) ma(3)
tab(5,5)=eq33.@aic

equation eq4.ls  {%x}   c  {%x}(-1) {%x}(-2) {%x}(-3) {%x}(-4)
tab(6,2)=eq4.@aic
equation eq41.ls  {%x}   c  {%x}(-1) {%x}(-2) {%x}(-3) {%x}(-4) ma(1)
tab(6,3)=eq41.@aic
equation eq42.ls  {%x}   c  {%x}(-1) {%x}(-2) {%x}(-3) {%x}(-4) ma(1) ma(2)
tab(6,4)=eq42.@aic
equation eq43.ls  {%x}   c  {%x}(-1) {%x}(-2) {%x}(-3) {%x}(-4) ma(1) ma(2) ma(3)
tab(6,5)=eq43.@aic

setline(tab,2)

show tab
```

みせかけの相関

　みせかけの相関に関するコマンド、プログラム例は次のようになる。期種や期間を適当に決めて、ワークファイルを開いてからの操作である。

```
smpl @first @first          ←サンプルの最初の期を指定。
series y=10                 ←y の初期値を 10 とする
smpl @first+1 @last         ←2 期目以降を指定。
y=0.5+y(-1)+nrnd            ←ドリフト付きランダムウオークの作成。「nrnd」とは、正規分
                                布を発生させる変数。

smpl @first @first
series x=100
smpl @first+1 @last
x=0.1+x(-1)+nrnd
```

```
smpl @all                          ←サンプルを全期間に変更する
equation eq1.ls y c x              ←y を x に回帰する。
equation eq2.ls d(y) c d(x)        ←y と x のそれぞれ階差を回帰する。
show eq1                           ←方程式を表示する。
show eq2
```

コマンド例

系列 gdp95 について、誤差項のラグ数を０（通常のディッキー・フラーテスト）として、ドリフト項を付ける場合。

```
uroot(const,lag=0)gdp95
```

系列 gdp95 について、フィリップス・ペロンテストを行う場合。

```
uroot(pp,const)gdp95
```

プログラム例

３種類のタイプ単位根検定を行い、その結果を出力する場合。まず単位根検定をして、save オプションで、その結果をそれぞれの行列（mnone,mconst,mtrend）に出力するものだ。

次に、最終結果出力用の行列（out）に各行列のデータを書き込む。colplace を使うと、行列 out の１列目に、mnone, ２列目に mconst, ３列目に mtrend を並べてデータをコピーできる。

```
gdp95.uroot(none,adf,save=mnone)   ←単位根検定を行い、結果をmnoneに保存する。
gdp95.uroot(const,adf,save=mconst)
gdp95.uroot(trend,adf,save=mtrend)

matrix(8,3) out        ←8行3列のoutという行列を作る。

colplace(out,mnone,1)   ←行列outの1列目に、mnoneの列の数値を入れる。
colplace(out,mconst,2)
colplace(out,mtrend,3)
```

行列outには、タイトルは付いてないが、次のような数値が入っている。途中の行は今回

のオプションでは空欄になっている。誤差のラグ数は、標準設定では自動的にＳＢＩＣ基準によって選ばれる。選択されたラグ数が２行目に、自動的に選ぶときの最大ラグ数11（標準設定）が最後の行に表示される。

どの行に何が入っているのかはマニュアルなどに明示されていないので、単位根検定の出力結果と見比べながら確認する。

	C1	C2	C3
サンプル数	78	83	83
ラグ数	5	0	0
t値	1.18	-1.40	-0.51
P値	0.94	0.58	0.98
未使用			
〃			
〃			
最大ラグ数	11	11	11

プログラム例

グレンジャーの因果関係はＶＡＲモデルの画面上で、計算はされるが、そのメカニズムを、次のようなプログラムで確認できる。採集的に出力されるのは、マネーサプライがＧＤＰに対してグレンジャーの意味での因果関係があるかどうかのワルド検定量である。

```
equation eqgdp951.ls gdp95 c  gdp95(-1) gdp95(-2) m2(-1) m2(-2) rblav(-1) rblav(-2)
scalar ssr1=eqgdp951.@ssr
scalar  regobs=@regobs
scalar ncoef=@ncoef
equation eqgdp952.ls  gdp95 c gdp95(-1) gdp95(-2) rblav(-1) rblav(-2)
scalar ssr2=eqgdp952.@ssr
scalar F=((ssr2-ssr1)/2)/(ssr1/(regobs-ncoef))
scalar  W=f*2
scalar  p=@chisq(w,2)
table  test
test(1,1)=w
test(1,2)=p
show test
```