ПРОГРАМЕН ПОДХОД В ИКОНОМЕТРИЧНАТА ПРАКТИКА- ОБЕКТНИЯ ЕЗИК НА Eviews ПРИ ИЗСЛЕДВАНЕТО НА ВРЕМЕННИТЕ РЕДОВЕ

от

Светослав Каменов Петков

свободен аспирант към катедра "Финанси", УНСС
гр.София

гр.София
09.2000г.

СЪДЪРЖАНИЕ

УВОД

"…на практика всички иконометрични спецификации са по правило "грешни" модели…Приложния иконометрик, подобно на теоретика, скоро открива с личен опит, че действащ модел, е не "верният" или "реалистичен", а онзи, който е пестелив, правдоподобен и поучителен…"
Фелдстеин М.[1]

Иконометрията е "съчетание на икономическа теория, статистика и математика, с оглед осъществяване на измерителния процес на икономическите обекти"[2]. В такъв смисъл коректния резултат от иконометричните изследвания зависи от детайлното познаване и прилагане на фундаменталните принципи на тези науки.

Имайки в предвид този "инструментариум" от възможности и ограничения, ако оставим настрана обективните критики към иконометрията[3] и допуснем, че изходната база данни е коректна и допуснем[4], можем да определим евентуалните проблеми пред които е поставено едно иконометрично изследване, по следния начин:

- не съблюдаване на техническата последователност от математически процедури използвани от конкретен иконометричен метод

- проблеми на спецификацията на изследвания икономически модел

Това са всъщност субективните грешки на едно изследване, към тях се отправят и субективните критики.

В първия случай става дума за детайлно познаване на математическата логика на даден метод, както и неговите допускания. Например класическата иконометрия е концентрирана около Гаусовско-Марковскта теорема за резултатите получени чрез метода на най-малките квадрати при задължителната хипотеза за оценките BLUE(best linear unbiased estimator). При нарушаване на тази хипотеза съществуват коригиращи техники, които правят оценките на моделите такива. Всичко това е вярно при допускането, че стохастичните процеси са стационарни. Ето защо е задължително всяко иконометрично изследване да започва с проверка на това допускане[5]. По подобен начин изглежда и допускането, че две и повече интегрирани от една и съща степен серии I(k) са коинтегрирани, без това да е доказано изрично[6].

---

[1] (1982) Inflation, Tax Rules and Investment: Some Econometric Evidence. Econometrica, 50,825-62.

[2] Проф.д-р.ик.н. Димитров.А. "Въведение в Иконометрията" с.9;Изд."Абагар"В.Търново 1998г.

[3] критиката на Лукас/1976,1980/ и неговите последователи към ефективността на иконометрията

[4] Подобно допускане е несериозно за изследвания свързани с българската икономика.

[5] ADF test;Phillips-Perron test

[6] "If there are N cointegrating equations, it means that none of the series is actually integrated, and the VAR can be reformulated in terms of the levels of all of the series. If augmented Dickey-Fuller tests show that some of the series are

Тези и подобни на тях грешки в техническото изпълнение на даден иконометричен метод, често водят до неверни заключения и прогнози[7].

В болшинството си тези техники са рутинни, но трудоемки дори и при използване на иконометричен софтуер в графичен интерфеис.

Втория субективен проблем е свързан с избора на методология за моделиране на спецификацията на конкретен модел. Теорията най-общо определя три подхода[8]:

1. Average Economic Regression(AER)
2. Test,Test,Test(TTT)
3. Fragility Analysis

За предимствата и недостатъците на тези модели може да се спори много без да се изясни, кои от тях е принципно най-добър. Възможно е и категоричен отговор изобщо да няма, защото целите на иконометричния анализ мотаг да са различни.

На практика е възможен по-радикален подход към тази проблематика. Такъв е обхващането на максималния набор от n на брои фактори/между които съществува икономически обоснована връзка/, които да образуват M на брой модела от всички техни комбинации от n-ти до 2-ри клас, където:

$$M=\sum_{k=n}^{2}\frac{n!}{k!*(n-k)!}$$

минималната стойност на k е 2 тъй като комбинации от 1-ви клас са неприложими за целите на иконометричния анализ.

По този начин от самото начало на изследването ще имаме всички възможни **базови модели**, които могат да бъдат манипулирани чрез избрана от нас техника. Така формулиран този подход, като чели най-много се доближава до TTT метода, но за разлика от него тук няма ограничения в последващата еволюция на даден модел. В такъв смисъл това предложение не трябва да се разглежда като алтернатива на изброените по-горе методи, а по-скоро като етап предхождащ избора на конкретна методология, чрез която в последствие ще се търси оптималния модел.

По такъв начин в самото начало на иконометричния анализ ще съставим **M** на брой[9] **базови модели**. Чрез някои от тях ще се проверяват съществуващи теоретични модели, а други ще описват нови връзки. При все това не трябва да се пренебрегва и факта, че само по себе си това предложение частично решава проблема със спецификацията на модела, като в самото начало конструира всички възможни комбинации на изследваните икономически редове.

Следвайки тази логическа последователност от рутинни операции, всяко иконометрично изследване би следвало да започва с изследване на стационарността на интересуващите ни променливи, привеждането им в изисквания от конкретния иконометричен метод вид/за класическата иконометрия

---

integrated, but the Johansen tests show that the cointegrating rank is N, there is a contradiction. Some specification error might be responsible for this contradiction." Eviews2.0 HELP;QMS 1996.

[7] Изследване от Икономическия институт на БАН "Икономиката на България днес и утре/състояние и краткосрочни перспективи /" С.1994г., в което няма изследване на стационарността. Приета е интегрираност на сериите от първи ред.

[8] Kennedy P.;Guide to Econometrics;1995;p74;Blackwell Publishers LTD

[9] За целите на конкретно изследване комбинациите могат да са повече. Например при необходимост от използването на тренд комбинациите ще са 2*M.

този вид трябва да осигурява стационарноста на сериите, а за VEC модела за да са коинтегрирани променливите необходимо, но не достатъчно условие е да са интегрирани от един и същ ред/ и комбинирането на им в М на брой базови модели.

Това е почти непосилна задача ако трябва да се прави без РС и съответен иконометричен софтуеар. Напоследък решаването на този технически проблем е улеснено от излезлите на пазара богата палитра[10] от програмни иконометрични продукти. Дисциплини с подобен предмет се появиха в доста престижни университети. Въпреки това простото използване на тези програми в графичен интерфейс е също бавно и трудоемко, когато става дума за по-големи иконометрични изследвания. Имайки в предвид рутинноста на по-горе изброените операции, както и други иконометрични техники, които имат блок схемен характер, създаването на програми които могат да ги изпълняват многократно, бързо и надеждно спестява доста повече от колкото може да си представи всеки, които не ги е използвал. За щастие повечето от съвременния иконометричен софтуер поддържа обектно базиран програмен език опериращ с вградените функции и процедури на графичния интерфейс, като по този начин позволява гъвкавост при организиране на иконометричните изследвания съобразно използваната методология и предпочитания на потребителя.

Настоящата студия предлага програмни решения на често срещани рутинни в иконометричния анализ, с цел улесняване на работата на колегите си. Това може и да се разглежда като призив за научен обмен на опит в областта на програмирането в тази насока, които за съжаление е поне незабележим в България на този етап.

Разгледани са методите за изпълнение на основните иконометрични техники, с помощта на софтуера на Eviews разработен от Quantitative Micro Software.

Практични решения, чрез Eviews 2.1

Обектната конструкция и програмната реализация на на Eviews, както и информация за алтернативен иконометричен софтуер са поместени в Приложението.

Програмите разработени от автора, са писани на Eviews 2.1, които след коригиране на координатите на използваните табличните форми /Freeze object , са съвместими с Eviews3.0.

Примерите са разработени само за месечни честоти на изследванията. За конвенционалния вариант на всички примери се свържете с автора[11] или конструирайте сами.

1. Автоматично зареждане на N на брой серии и техните логаритмични форми в работен фаил.

Предхождащо всяко иконометрично изследване е коректното зареждане на сериите в работен файл на Eviews. Честото използване на логаритмичните изражения на изследваните променливи, прави тази операция желателна при самото зареждане на променливите. Следната програма изпълнява тези операции, като проверява за наличие на отрицателни елементи във всяка серия с цел

[10] Има се в предвид международния пазар на подобен софтуер.
[11] E-mail: svetlikp@yahoo.com

коректното изчисляване на логаритмичните форми[12]. Новосъздадените серии имат съответното име пред което е добавено **log**.

```
'=========================================================
WORKFILE s3 m  1985:1 1999:12
!broi={%0}  '!broi Променлива за брой изследвани променливи
      FOR !i=1 to !broi 'Цикъл за "зареждането" на сериите
      fetch %{!i} 'Зареждане на оригиналните серии
%toti="%{!i}"
IF {%toti}>0 then       'Проверка за наличие на отрицателни
елементи
genr log{%toti}=log(%{!i})    'Създаване на логаритмичната серия
Endif
     NEXT
'=========================================================
```

След стартиране на програмата трябва да се попълни прозорецът Program Argument с: първо броя на сериите/като число/, последвано от точните имена на сериите/букви и цифри/ в базата данни, с интервал между тях. Броят на последните трябва да отговаря на числото въведено в началото. Пример:

5 hsf gmpyq fybaac fygt10 fspcom

Обръщане към така заредените серии става по следния начин:

show %1

equation EQ1.ls log{%1} c log{%2} и т.н.

Следващите точки на тази глава представят практичното решение на основни иконометрични процедури и са в в същност SUBROUTINE-и изграждащи една реална програма. Пасажите от текста оградени от следния показалец `'==========` , могат директно да бъдат копирани в програмния интерфейс на Eviews2.1., като по този начин ще изградят общата програма[13]. Отделните SUBROUTINE's могат да се извикват от други потребителски програми, ако това става по коректен начин[14].

2. Тестове за степен на интегрираност ADF на автоматично заредени серии и техните логаритмични форми.

След като сериите са заредени по описания по-горе начин, самите им имена не могат да се записват в таблици, което предизвиква известно неудобство. Ето защо при зареждането на сериите/и на логаритмичните им форми/ в този пример ще бъде използван един допълнителен метод за записване на имената на сериите в специална таблица NAME. Това става чрез записването им от стандартната таблица на Freeze на ser(n), където на 2-ри ред и 2-ра колона, от 30-ти символ винаги е изписано името на серията. За да е вярно това, независимо каква е заредената серия- преди това създадена от потребител или от външно създадена база данни/която също трябва да е заредена в базата на Eviews/, програмата създава временно нова серия, която е равна на конкретна серия, но с име завършващо със символа: $

`GENR %{!i}$=%{!i}` което става в **`SUBROUTINE NAME`**

Но първо нека представим "основната" програма, чрез която се "извикват" основните SUBROUTINE's. В отделните точки са представени SUBROUTINE's, които се "извикват" последователно.

Основна програма

```
'=========================================================
WORKFILE s3 m  1975:1 1991:12
!lagl={%0}  '!lagl Променлива за ЛАГОВЕТЕ на UNIT ROOT test
```

[12] В случай, че има отрицатилни числа в една серия, програмата попълва тези елементи с "NA", при изчисляването на логаритъм от нея. В последствие това може да предизвика неточности.

[13] Трябва да бъде спазено изискването Основната програма да се разположи под SUBROUTINE's.

[14] например: INCLUDE c:\FOLDER NAME\PROGRAM NAME.PRG

```
!broi={%1}  '!broi Променлива за брой изследвани променливи
!skp=0            'Брояч за новосъздадените LOG серии
TABLE((4*!BROI),9) NAME
CALL TABLICI 'Създава работните таблици
call Zarejdane
'============================================================
```

Програмата извиква първо **TABLICI**, която създава дизайна на таблицата TABUROT с окончателните резултати от ADF теста.

```
'============================================================
SUBROUTINE TABLICI
TABLE((8*!BROI),9) TABUROT
setcell(TABUROT,1,4,"Augmented Dickey Fuller Unit Root
Test",25,"c")
setline(TABUROT,2)
SETCOLWIDTH(TABUROT,1,15)
setcell(TABUROT,3,1,"VARIABLE",25,"c")
setcell(TABUROT,3,2,"LEVEL",25,"c")
setcell(TABUROT,3,3,"1-st dif.",25,"c")
setcell(TABUROT,3,4,"2-nd dif.",25,"c")
setcell(TABUROT,3,5,"1% McKin.",25,"c")
setcell(TABUROT,3,6,"5% McKin.",25,"c")
setcell(TABUROT,3,7,"10% McKin.",25,"c")
setcell(TABUROT,3,8,"INTEG.",25,"c")
SETCOLWIDTH(TABUROT,9,4)
setcell(TABUROT,3,9,"LAG",25,"c")
setline(TABUROT,4)
EndSub
'============================================================
```

След това се извиква SUBROUTINE **Zarejdane**, която зарежда посочените от нас серии и евентуално създава LOG сериите, по подобен начин на описания по-горе, но както вече стана ясно, със записване на имената на сериите в таблица NAME, което става в SUBROUTINE NAME.

```
'============================================================
SUBROUTINE Zarejdane
      FOR !i=2 to !broi+1     'Цикъл за "зареждането" на сериите
      fetch %{!i}
      call name("",1) 'Викане на процедурата за имената на
променливите
%toti="%{!i}"
if {%toti}>0 then       'Проверка дали всички елементи са
полужителни
genr log{%toti}=log(%{!i})     'Създаване на LOG серията
!skp=!skp+1                    'Брояч за новосъздадените LOG серии
call name("log",2) 'Викане на процедурата за имената на
endif
      NEXT
!br=5       'Брояч за Определяне на реда на запис в таблицата
taburot
      FOR !i=2 to !broi+1 'Цикъл за ADF тестове на оригиналните
серии
      %v="%{!i}"
      !Q=!i-1  'Променлива показваща последователния номер на
серията
            FOR %m n c t 'Цикъл за опциите на ADF теста
            call lagopt(%v,%m)      'Викане на процедурата lagopt
            call UROT(%v,%m,1)      'Викане на ПРОЦЕДУРАТА-UROOT
              %v="%{!i}"    'Премахване на ефекта от послед.
разлики
            NEXT
      !br=!br+1         'Брояч за ред на запис в taburot
%toti="%{!i}"
IF {%toti}>0 then      'Проверка дали е създадена съответна LOG
серия
%v="log{%toti}"
```

```
FOR %m n c t
            call lagopt(%v,%m)      'Викане на процедурата lagopt
            call UROT(%v,%m,2)      'Викане на ПРОЦЕДУРАТА-UROOT
             %v="log{%toti}"
            NEXT
      !br=!br+1
ENDIF
      NEXT
      !br=!br+1
setline(TABUROT,!br)
!br=!br+1
setcell(TABUROT,!br,1,"* McKinnon critical values for rejection
of hypothesis of a unit root",25,"c")
EndSub
'============================================================
SUBROUTINE NAME(string %dumi,scalar !hh)
GENR %{!i}$=%{!i} 'Създаване на временна серия
            freeze(f{!i}) %{!i}$ 'Таблица на изглед на серията
            !c=30             'Начало на записа на името
                  while @mid(f{!i}(2,2),!c,1)<>"$"
                  !c=!c+1
                  WEND
            !p=!c-30
            %r=%dumi+@mid(f{!i}(2,2),30,!p)
            setcell(NAME,!i-1,!hh,%r) 'записване на името
delete f{!i}      'Изтриване на Freeze
delete %{!i}$     'Изтриване на временната серия
EndSub
'============================================================
```

След зареждане на конкретна поредна серия, се прави проверка дали всички нейни елементи са положителни **/if {%toti}>0 then/** и ако това е така се създава нейната логаритмична нова серия /**genr log{%toti}=log(%{!i})**/. След като този цикъл е повторен до изчерпване на посочените от потребителя серии/в Program Argument на RUN менюто/ и като за всяка оригинална е евентуално създадена съответна LOG серия, започва изпълнението на цикъла за тестването на степента на интегриране. Този цикъл също се повтаря толкова пъти колкото променливи първоначално сме въвели. За всеки цикъл програмата проверява дали съществува съответна LOG серия и ако има прави отделни тестове и на нея. За всяка променлива се стартира втори FOR цикъл за различните опции на ADF теста: **FOR %m n c t**.

Всеки ADF тест се предхожда от процедура за определяне на оптималния лаг. Критерии за което е минималната стойност на AIC criteria в стандартното уравнение за стационалност. Това става с извикването на SUBROUTINE LAGOPT, чрез **call lagopt(%v,%m)**, която изглежда по следния начин:

```
'============================================================
SUBROUTINE LAGOPT(string %lag,string %arg)
      FOR !x=0 to !lagl 'Цикъл за проверката на лаговете
      freeze(fr2) {%lag}.uroot({%arg},!x) 'Таблица с ADF test
            IF %arg="n" then !j=0   'Контроли за позиция на AIC
            ENDIF
            IF %arg="c" then !j=1   'Контроли за позиция на AIC
            ENDIF
            IF %arg="t" then !j=2   'Контроли за позиция на AIC
            ENDIF
      IF !x=0 then
      !mm=20+!j
      %min=fr2(!mm,5)   'Начален минимум
      scalar min1={%min}
      !broilag=!x
      ENDIF
      !mn=20+!x+!j
      %kon=fr2(!mn,5)
```

```
      scalar kon1={%kon}
      IF min1>kon1 then
      min1=kon1    'Окончателен минимум
      !broilag=!x 'Запис на съответния лаг
      ENDIF
      delete fr2   'Изтриване на работната таблица
      NEXT
      EndSub
'============================================================
```

Тази процедура се изпълнява в рамките на всеки цикъл от **FOR %m n c t**, като резултата е определяне на оптималния лаг записан в **!broilag**, които е и използван в ADF теста, които се извиква след всяко изпълнение на **SUBROUTINE LAGOPT**, чрез **call UROT**, която изглежда по следния начин:

```
'============================================================
SUBROUTINE UROT(string %f,string %ar,scalar !col)
!b=0               'Брояч за WHILE loop
!ur=0              'Брояч за UNITROOT
!sic=0             'Контролна променлива за риск за грешка
!br=!br+1          'Брояч за запис на ред
setcell(taburot,!br,1,NAME(!q,!col)+"_"+%ar,"l")'Запис на име на
сер.

while !b=0
  freeze(fr1) {%f}.uroot({%ar},!broilag)
setcell(taburot,!br,5,fr1(1,5),"c",4) 'Запис на критичните с-ти
setcell(taburot,!br,6,fr1(2,5),"c",4) 'Запис на критичните с-ти
setcell(taburot,!br,7,fr1(3,5),"c",4) 'Запис на критичните с-ти
IF !UR=0 and abs(fr1(1,2))>abs(fr1(3,5)) and
abs(fr1(1,2))<abs(fr1(1,5)) Then 'Проверка за стационарност в
нивата
            !SIC=0            'Указва, че нивата не са
стационарни
ELSE
      if abs(fr1(1,2))>abs(fr1(3,5)) then !sic=10 'Риск за грешка
      endif
      if abs(fr1(1,2))>abs(fr1(2,5)) then !sic=5  'Риск за грешка
      endif
      if abs(fr1(1,2))>abs(fr1(1,5)) then !sic=1  'Риск за грешка
      endif
ENDIF
'Запис на стационализираните резултати
IF !sic=10 or !sic=5 or !sic=1 then
        setcell(taburot,!br,8,!ur,"C",3.0)/ 'Запис на
интегрираност
        setcell(taburot,!br,9,!broilag,"C",3.0) 'Запис на лаг
      if !ur=0 then
      setcell(taburot,!br,2,fr1(1,2),"c",-4)'Запис на стоиността
на теста
      endif
      if !ur=1 then
      setcell(taburot,!br,3,fr1(1,2),"c",-4)
      endif
      if !ur=2 then
      setcell(taburot,!br,4,fr1(1,2),"c",-4)
      endif
if  !sic=1 then
setcell(taburot,!br,5,@str(fr1(1,5))+"*","c",6.2) 'Запис на
критичните стойности и отбелязване на стационализирането
endif
if  !sic=5 then
setcell(taburot,!br,6,@str(fr1(2,5))+"*","c",6.2)
endif
if  !sic=10 then
setcell(taburot,!br,7,@str(fr1(3,5))+"*","c",6.2)
```

```
endif
        !b=1                                    'Знак за край на Wend
else
'Записване на нестационарните резултати
      if !ur=0 then
      setcell(taburot,!br,2,fr1(1,2),"c",-4)
      endif
      if !ur=1 then
      setcell(taburot,!br,3,fr1(1,2),"c",-4)
      endif
      if !ur=2 then
      setcell(taburot,!br,4,fr1(1,2),"c",-4)
      endif
     genr d{%f}=d({%f})
     %f="d"+%f
     !ur=!ur+1
       endif
delete fr1
wend
EndSub
'============================================================
```

Резултат от изпълнението на програмата до тук е създаването и попълването на таблицата TABUROT, в която са попълнени всички тестови стойности, за всяка възможна опция на теста, критичните стойности на McKinnon, степените на интегрираност и оптималния лаг.

Следва извикването на SUBROUTINE Spisak, с която в таблицата Spisak се попълват имената на онези серии които са стационарни или стационализирани чрез определен брой последователни разлики. Извикването на тази процедура става чрез **call Spisak** , в основната програма.

```
'============================================================
SUBROUTINE Spisak
TABLE(4,9) Spisak 'Таблица за списък на стационарните серии
!fif=0      'Брояч за зъписаните колони в Spisak
FOR !aa=1 to %1+!skp
!lll=(!aa-1)*4+6
IF  taburot(!lll,8)=0 or (taburot(!lll+1,8)=0) or
(taburot(!lll+2,8)=0) Then     'Проверка за стационалност
      !fif=!fif+1
      setcell(spisak,1,!fif,taburot(!lll,1),25,"c")
else
      !fif=!fif+1
      setcell(spisak,1,!fif,"d"+taburot(!lll,1),25,"c")
ENDIF
NEXT
EndSub
'============================================================
```

Тук трябва да се отбележи, че за целите на VEC анализа ще са ни нужни само интегрираните от еднакъв/първи/ ред променливи, т.е. единствения If statement в SUBROUTINE Spisak, трябва да се промени както следва:

```
if  taburot(!lll,8)=1 or (taburot(!lll+1,8)=1) or (taburot(!lll+2,8)=1) Then
        !fif=!fif+1
        setcell(spisak,1,!fif,taburot(!lll,1),25,"c")
endif
```

3. Генериране на възможните комбинации от снижаем клас и определяне на онези от тях, които могат да бъдат на базовите модели.

Следваща стъпка е да се генерират всички възможни комбинации от записаните в таблицата Spisak променливи, т.е. комбинациите от !fif елемента от

!fif, !fif-1, !fif-2…2 клас. Тъй като някои комбинации могат да включват серия, нейната логаритмична форма или нейните n-ти разлики /ако ни интересуват стационарните серии/ едновременно, а това за целите на иконометричното моделиране е безсмислено, програмата сортира всички комбинации от спадащ клас и не записва подобни комбинации. Генерирането на комбинациите става в Subroutine Forloop, след което записа на адекватните от тях става в **Subroutine Forloop**. Това изглежда по следния начин:

```
'============================================================
SUBROUTINE Kombin
!red=1       'Брояч за редовете на комбинациите
table(300,20) tab
FOR !MAM=!fif TO 2 STEP -1 'Цикъл за клас на комбинациите
FOR !x1=1 to !fif-!MAM+1
If !MAM>=2 then
      For !x2=!x1+1 to !fif-!MAM+2
      If !MAM>=3 then
            For !x3=!x2+1 to !fif-!MAM+3
……
            call Forloop 'Извикване на Forloop
……
            next
      else
      call Forloop
      endif
      next
else
call Forloop
endif
next
NEXT
EndSub
'============================================================
```

В зависимост от броя на променливите които ще се комбинират, т.е. !fif трябва да се допише **SUBROUTINE Kombin**, като включването на прекалено много променливи е безсмислено, а оттам и разписването на циклите до големи нива е ненужно.

```
'============================================================
SUBROUTINE Forloop
For !cell=1 to !MAM 'Цикъл за запис на елем. От една комбинация
!flag=4            'Променлива за While
!buk=1             'Брояч за символите в едно име
WHILE !flag=4      'Процедура за премахване на "_n" символа от
името
      IF @mid(SPISAK(1,!x{!cell}),!buk,2)="_n" then
      %sim=@left(SPISAK(1,!x{!cell}),!buk-1)
      !flag=5
            else
            !buk=!buk+1
            endif
Wend
setcell(tab,!red,!x{!cell},%sim,20,"l")'Записване на име в
комбин.
next
!far=0
FOR !waw=1 to !fif-1     'Цикъл за премахване на неадекватните
комбин.
IF tab(!red,!waw)<>"" and tab(!red,!waw+1)<>"" then
      If @mid(tab(!red,!waw+1),4,10)=tab(!red,!waw) or
(@mid(tab(!red,!waw+1),5,11)=tab(!red,!waw)) or
(@mid(tab(!red,!waw+1),4,10)=@mid(tab(!red,!waw),2,15)) or
(@mid(tab(!red,!waw+1),5,11)=@mid(tab(!red,!waw),2,11)) then
      !far=1             'Знак за премахване на неадекв. Комбин.
      exitloop
```

```
      endif
endif
NEXT
      IF !far=0 then
      !red=!red+1
      else
      FOR !q1=1 to !fif
      setcell(tab,!red,!q1,"",20,"l")
      NEXT
      ENDIF
EndSub
`=========================================================
```

След приключване на действието си програмата на този етап е сортирала всички комбинации от сериите/в интересуващата ни форма на нива или разлики/, като е записала в таблицата tab, онези от тях които са съставени от елементи образуващи непротиворечащи на спецификационните правила модели. По този начин в tab не са попаднали комбинации от типа: …ser1 log(ser1) … или …ser1 d(ser1)…, които се генерират от **SUBROUTINE Kombin**, но се отхвърлят от **SUBROUTINE Forloop**.

4. Генериране на базовите модели и оперирane с тях /Случая с Традиционната Иконаметрия./

В зависимост от избрания от потребителя метод на иконометричен анализ, от тук нататък програмата може да се развие в няколко направления:

- за нуждите на Традиционната иконометрия- в такъв случай, както стана ясно в увода сериите използвани за конструирането на базовите модели трябва да са стационарни, т.е. онези които програмата записа в таблицата **Spisak** и бяха използвани за генерирането на иконометрично обосновани модели в таблицата **tab**.
- за нуждите на VEC анализа- тогава използваните за моделирането серии трябва да са нестационарни, обикновено от първи ред. За тази цел в **SUBROUTINE Spisak**, трябва да бъде направена поправката в IF station на същата, с цел записването на нестационарните серии/за разлика от така записаното в програмата, където се записват сериите в стационарна форма/
- друг специфичен метод

Автора избра демонстрация на първия случай.

Използвайки вече записаните “адекватни” от техническа гледна точка комбинации в таблицата **tab**, създаваме **!red-1**/броя на записаните редове в табл. **tab**/ на брой уравнения от вида:

ser1=c+c(1)*ser2/или стационарната форма/+…

За целта добавяме в основната програма **call EQ**, с което се извиква **SUBROUTINE Eq**, в която се генерират моделите одухотворени в обект Equation с последователни имена от 1 до !red-1.

```
`=========================================================
SUBROUTINE Eq
FOR !s=1 to !red-1      `Цикъл за генериране на уравнения
!flag=0
%sos=""                 `Стринг за уравнението
FOR !w=1 to !fif        `Цикъл за зареждането на елементите
      IF tab(!s,!w)<>"" then `Проверка за първа серия
      !flag=!flag+1
            IF !flag=2 then
            %sos=%sos+" c"    `Добавяне на конс. След първия
елем.
            ENDIF
      ENDIF
```

```
      %sos=%sos+" "+tab(!s,!w)'Добавяне на останалите елеминти
      next
      Equation EQ{!s}.ls %sos 'Деклариране на уравнението
NEXT
EndSub
'=========================================================
```

Така създадените базови модели могат да бъдат манипулирани по избран от потребителя начин, прилагайки богата гама от тестове и коригиращи техники, с цел оптимизирането им.

Ето например как би изграждала SUBROUTINE която използва CHOW forecast теста за стабилност. Ще бъдат тествани всички наблюдения в период обхващащ период от 15% от всички наблюдения, считано от последното наблюдение. За целта в главната програма се добавя **call ChowF**, с което се извиква **SUBROUTINE ChowF**, която прави процедурите на въпросния тест.

```
'=========================================================
SUBROUTINE ChowF
genr s1=1   'Създава нова серия с попълнени елементи 1
!rer=@obs(s1)     'Преброява наблюденията в нея
delete s1         'Изтриване на S1
table(100,!red-1) tabChow     'Деклариране на табл. за
резултатите
setcell(tabChow,1,1,"Резултати от Cshow Forecast Test при 5% риск
за грешка")
setcell(tabChow,3,1,"Начало на теста"+@otod(@ceiling(0.5*!rer)))
setline(tabChow,6)
setline(tabChow,4)
For !x=1 to !red-1
setcell(tabChow,5,!x,"eq"+@str(!x))
next
!max=7      'Първоначален максимум
FOR !e=1 to !red-1
!redn=7     'Първоначална стойност на брояча за реда
For !rr=@ceiling(0.5*!rer) to !rer-1  'Цикъл за наблюденията за
теста
%rar=@otod(!rr)
freeze(frf) eq{!e}.chow(f) %rar       'Таблица на chow(f)теста
      IF @val(frf(3,5))<0.05 then 'Проверка за верността на теста
5%
      setcell(tabChow,!redn,!e,@otod(!rr))
      !redn=!redn+1     'Брояч за редовете на таблицата
      ENDIF
delete frf              'Изтриване на таблицата на теста
call Max(0)
NEXT
IF !redn=7 then
setcell(tabChow,!redn,!e,"Стабилна")
setcell(tabChow,!redn+1,!e,"за")
setcell(tabChow,!redn+2,!e,"периода")
call Max(3)
ENDIF
NEXT
setline(tabChow,!max)
EndSub
'=========================================================
```

За коректен запис на последната линия в таблицата с резултатите от теста се създава брояч на най-много записани редове, който е поместен **SUBROUTINE Max**, която се извиква от **SUBROUTINE ChowF**, след края на всяка проверка на стабилността за всяко уравнение.

```
'=========================================================
SUBROUTINE max(scalar !k)
IF !redn+!k>!max then
!max=!redn+!k     'Записване на максималната стойност на редовете
ENDIF
```

```
EndSub
`============================================================
```

Стартирането на програмата става, чрез въвеждане на:

Максимален брой на лаговете за които ще се търси оптималното уравнение при ADF test	***Брой на изследваните променливи***	***Точно име на първата променлива/не трябва да има интервали в името/***	…	***Точно име на P-та променлива/не трябва да има интервали в името/***
ЧИСЛО: **N**	ЧИСЛО **P**	СТРИНГ		СТРИНГ

… в Program Argument полето на RUN менюто. Горните аргументи трябва да са отделени с интервал помежду си.

ЗАКЛЮЧЕНИЕ

Използването на големите възможности на програмирането с компютърните езици от пакетите на съвременните иконометрични софтуери, става необходимост в приложната иконометрия. За съжаление Българската иконометрия изостава от тази тенденция. Обективните факти налагат в обучението на студентите във Висшите Учебни Заведения по икономика, да бъдат широко застъпени специализираните модули по работа, а защо не и програмиране със **съвременни** иконометрични програмни продукти.

Създаването на потребителски програми за целите на иконометричните анализи, би улеснило не само авторът им, но и потребителя, ако те също бъдат публикувани. Често ползвателя на такива изследвания гледа на резултатите от тях с недоумение и голямя доза подозрение, тъй като "тънките" моменти на техническите процедури остават обикновенно забулени в "неизвестност". В случай, че използваната програма е достъпна за всеки, спорните моменти /често това са допусканията на анализа/, които биха заинтересували читателя ще бъдат известни от самото начало. Това предложение в действителност е далеко от Българската реалост, като се има в предвид, че дори сериите използвани в и без това малкото иконометрични изследвания остават неизвестни за читателя.

Всчко което ще представлява интерес за читателя от изложеното в тази студия, може да се ползва без всякъкви ограничения. В такъв смисъл автора очаква критики и предложения свързани с темата.

ПРИЛОЖЕНИЕ

Платформа на EViews

Обективната нужда от компютърна обработка на данните и изпълнението на сложните иконометрични техники е предпоставка за разработването на богата гама от компютърен софтуер в тази насока. Ето малка част от тях[15]:

Autobox: http://www.autobox.com/
BMDP: http://www.meridian-marketing.com/BMDP/index.html
EasyReg: http://econ.la.psu.edu/~hbierens/EASYREG.HTM
Gauss: http://www.aptech.com/
MiniTab: http://www.minitab.com/
PcGive: http://hicks.nuff.ox.ac.uk/Users/Doornik/
Soritec: http://www.fisisoft.com/
SPSS: http://www.spss.com/
Mx: http://views.vcu.edu/mx/
…

Анализа на предимствата и недостатъците на тези продукти би отнело достатъчно време, за да му бъде посветена специална дискусия. В България разпространение получиха EasyReg, PcGive, Soritec, SPSS и др., но със специално внимание трябва да бъде удостоен Eviews: http://www.eviews.com/. Последния комплексно обхваща почти всички съвременни иконометрични техники, представени в лесен за работа графичен интерфейс[16]. Неоспорим плюс е и интегрирането на обектно базиран програмен език в неговата платформа.

Въпреки това основна причина за обвързването на настоящата студия с програмирането на Eviews, е използването на тази програма от основните икономически институции на България, като БНБ, Икономическия институт на БАН, АИАП и др., за целите на иконометричния анализ.

Бърз преглед на обектната конструкция и реализация на Eviews 2.1/почти без промени за Eviews 3.0/ би осветлила възможностите на програмния пакет, като в същото време ще изясни параметрите на програмния модул на програмистите заинтересовани от този софтуер.

1.Обекти

Eviews поддържа два вида обекти:

- **data objects** – съдържа данни с декларирано име. Всеки data object има процедури за различни **views** и **procedures**, сварзани с определен тип data object. Всички типове data object са както следва:

[15] Допълнителна информация на Интернет адреса:Software Products: http://www.oswego.edu/~economic/econsoftware.htm

[16] In March of 1994, QMS launched a revolution in econometric software with the release of EViews 1.0. Featuring a modern, graphical object-oriented user interface, EViews provided an alternative to antiquated, software design. The combination of power and ease-of-use made EViews an immediate best seller. http://www.eviews.com/.

COEFFICIENT
Коефициент вектор. Вектор съдържащ коефициентите изчислени от EQUATION и SYSTEM.

EQUATION

LS
TSLS
GMM
ARCHLOGIT
PROBIT

GROUP
Група от серии

MATRIX
Матрица с размери от редове и колони.

POOL **(time-series, cross-section)**
Понякога базата от данни съдържа cross-sectional и time series развитие. Например може да имаме данни организирани по държави и години. EViews ще оперира автоматично с двата вида данни.

ROWVECTOR
Вектор ред.

SAMPLE
Обект деклариращ обхвата на изследването.

SCALAR
Число (single number).

SERIES
Time series или cross section data.

SYM **(symmetric matrix)**
Симетлична матрица.

SYSTEM
Система от уравнения, за решаване.

VAR **(vector autoregression)**
Vector autoregression.
VAR-unrestrikted
VEC

VECTOR
Вектор

- **view objects**-не съдържат данни, а графики и текст свързани с data objects. Няма процедури. Eviews поддържа три типа **view objects**:

GRAPH
Графиките са view objects създадени от freezing на графичен изглед на друг обект или командите PLOT, BAR, SCAT, или PIE.

TABLE
Обект Таблица.

TEXT
Текст.

2.Структура и реализация на програмния модул.

Програмния език е обектно базиран, и опит в някой програмен компютърен език като BASIC, ще помогне да се разбере, че повечето от методите за контрол и опериране са доста сходни.

Основната постановка на програмния език на EViews е възможността да се създават имена на обекти, чрез комбинирането на променливи съдържащи части от имена.

Програмата е файл от EViews команди. Общия формат на всяка команда е:

```
Action(act_opt) object.view_proc(view_opt) arg_list
```

Action- една от следните команди(do,freeze,print,show)
act_opt- опция, която модифицира текущото състояние на Action

object- името на обекта, с които ще се оперира
view_proc- обектната процедура или езглед **/view/,** с които ще се работи
arg_list- аргументи на **view_proc,** разделене със запетаи

Програмата не е обект от работен файл /workfile/, но вместо това той може да контролира EViews обектите и да манипулира самите работни файлове.

Основните параметри на програмния език Eviews са:

Програмни променливи /Program Variables/

1. Control Variables

Control variables са променливи, които могат да приемат само цифрови стойности. Веднъж приели числово съдържание, чрез знака = могат да се използват навсякъде в програмата, където е уместно използването на число. Името им започва със знака **!**, последвано от име с 1 до 15 символа. Пример:

!X=13.345

2. String Variables

String variable е променлива, чиято стойност е стринг на текст. Името на тези променливи започва със символа **%**, а придобиват стойност чрез знака = и стринг от дясно. Например:

%value="GDP"

3. Replacement Variables

EViews позволява конструирането на командни редове използвайки съдържанието на string and control variables. По този начин към една стрингова променлива можем да се обърнем по съдържанието и. Пример: Ако към декларираната по-горе string variable се обърнем в записа на Equation object, ще използваме вече съществуващата серия GDP.

Equation EQ1.ls %value c INV

Replacement variables могат да се комбинират с букви и числа или други Replacement variables за да формират дълги думи. В този случай те трабва да бъдат поставени в къдрави скоби **{ }**.

Контролните променливи също могат да се използват като Replacement Variables.

Replacement Variables имат важно значение за конструирането на обектните имена.

Програмни аргументи/Program Arguments/

Програмните аргументи са специални стрингови променливи, които се въвеждат в програмата при стартирането й. Аргументите всеки път могат да са с различни стойности. Те могат да са и числа, но в стрингов формат и използването на тази числова стойност в програмата за целите на някакво изчисление може да стане като се обърнем към този аргумент като Replacement Variable. Аргументите автоматично приемат имената %0, %1, %2, и.н.

Контролиране на програмата/Controlling a Program/

1. IF Statements

Стандартна форма използвана от програмните езици:

IF !a/%b/<>=условие then
…
Else
…

Endif

2. FOR Loops with Control Variables or Scalars

Стандартна форма използвана от програмните езици:

For !x=n to p STEP c
…
NEXT

При изпалзване на стрингови променливи:

FOR %y GDP GNP NDP NNP
EQUATION {%Y}trend.ls %y C {%Y}(-1) TIME
NEXT

%y приема последователно изброените след него стритгове. Възможно е използване на повече променливи в един цикъл.

ExitLoop-прекъсва действието на FOR цикъла.

3. The While Loop

Стандартна форма използвана от програмните езици:

!p=0
While !x/%b/<>=условие
…
!p=!p+1
Wend

Subroutines

Subroutine е група от команди, които позволяват многократното изпълнение на определена процедура, в случай на необходимост. Те могат да се използват в определена програма дори и да са позиционирани в друга. Subroutine започва с думата SUBROUTINE последвана от името й и декларирането на използваните от нея променливи. Subroutine завършва с думата EndSub. Между тях се разполагат командите.

SUBROUTINE PWR(SERIES V, SERIES Y, SCALAR PW)
V=Y^PW
ENDSUB

Извикването й става с сомандата: **call(gdp,hsf,2)** последвана от списъка на съответните променливи от вид съответен на декларацията в SUBROUTINE. Има два вида SUBROUTINE:

1. Global

Използват и създават глобални променливи и обекти, които са съществували и остават да съществуват преди и съответно след извикването й.

2. Lockal

При деклариране им след SUBROUTINE се добавя LOCAL последвани от декларациите. Обектите създадени от такава SUBROUTINE са локални и след завършване на действието на локалната SUBROUTINE, няма да съществуват в работния фаил.

Подробно описание на основните правила в програмирането на Eviews, са поместени в “Command and Program Reference” Eviews3.0.