Peter Revesz
Any questions to: peterrevesz86@gmail.com
2013.03.29.

DOCUMENTATION FOR THE ESTIMATION 3D RANDOM EFFECTS PANEL DATA ESTIMATION PROGRAMS

All algorithms stored in separate do files. Usage is to first run the full ".do" file in Stata, then run the desired estimations using the appropriate commands.

This is the documentation for the estimation of 3D random effects panel data models based on The CEU Working Paper WP/20012/1 by Laszlo Matyas, Cecilia Hornok and Daria Pus.

MODIFIED STANDARD ERRORS PROGRAM

First program calculates the modified standard errors of the OLS estimators, due to the complex structure of the variance-covariance matrix of the model. The corresponding paper by Laszlo Matyas, Cecilia Hornok and Daria Pus canbe reached at: http://www.personal.ceu.hu/staff/repec/pdf/2012_1_revised.pdf (model (7), pp. 24-27).

The program calculates the following matrix and displays its diagonal elements separately for the user:

$$\Omega = (X'X)^{-1} * (X'\mathrm{E}\,X) * (X'X)^{-1}$$

where X marks the independent variables involved in the OLS and E marks the paper-specified variance-covariance matrix.

Program is for 3 dimensional data only.

Do file name: covre5.do [Abbreviation of Covariance Random Effect]

Start command: recov1.

Syntax:

recov1 varlist [if] [in] , Time(time panel variable) Panels(panel variables) [REGOPTions(options for OLS esmitations)]

Syntax means that Statat-regular method could be used to reduce the sample, and the options of Stata "regress" command could be used here as well. Take care, that regress command could estimate more than an OLS regression, in these cases the programs estimations would be erroneous. Time and Panel options are mandatory.

ALGORITHM WITH COMMENTS:

```
*! Header
*************************
capture noisily program drop recov1

program define recov1 , sortpreserve
        version 12
        #delimit ;
        syntax varlist(min=2 numeric) [if] [in] , Time(varlist numeric max=1)
        Panels(varlist numeric min=2 max=2)
        [
        REGOPTions(string asis)
        ]
        ;
        #delimit cr
        marksample touse
        markout `touse' `panels'
        markout `touse' `time'

        tempfile master
        save `master' , replace
```

Part to define the syntax, version and to mark the sample based. Sample is in the estimation if:

- the “if” and “in”parameters are allow it.
- none of the panel variables contain missing value
- the time variable doesn’t contain missing value.

```
* No panelvars in regressvars.

        tokenize `panels'
        local token_i=1
                while  "``token_i''"!="" {
                        local panel`token_i'="``token_i''"
                        local ++token_i
                }

        quietly replace `touse'=0 if `panel1'==`panel2'

        keep if `touse'==1
        gen keept=`touse'


        local allvars="`varlist'"
        tokenize `varlist'
        local dependvar="`1'"
        local vartoken_i=1
        while "``vartoken_i''"!="" {
                foreach i of numlist 1/`token_i' {
                  if "`panel`i''"=="``vartoken_i''" {
                   display as error " Panelvariable found in the regression varlist." _newline "
Exiting with error."
                error 141
                  }
                 if "`time'"=="``vartoken_i''" {
                  display as error "  Time variable found in the regression varlist." _newline
" Exiting with error."
                error 141
                }
                }
         local ++vartoken_i
        }

        macro shift
        local indepvars ="`*'"

        display as text "Variable check passed succesfully" _newline
```

Separate panel variables for later use. Mark out-of-the sample the observations where panels variables have the same value, as this is a “no-self flow” estimation. Keep only the marked sample.

Check that none of the panel variables or the time variable is in the regression.

Separate the dependent variable and the independent variables for later use.

```
** Acquireing max's

        *tempvar tvar
        quietly egen tvar=group(`time') if `touse'
        quietly sum tvar if `touse'
        local tmax=r(max)
        drop tvar

        *tempvar ivar
        quietly egen ivar=group(`panel1') if `touse'
        quietly sum ivar if `touse'
        local imax=r(max)
        drop ivar

        *tempvar jvar
        quietly egen jvar=group(`jpanel') if `touse'
        quietly sum jvar if `touse'
        local jmax=r(max)
        drop jvar

if `imax'>`jmax' {
        local nmax=`imax'
        }
        else {
        local nmax=`jmax'
        }
```

Count the number of different values of panel variables and time variable. Define the max number of panel values as the larger value of the each panel variable.

```
** Hasnocons

        tokenize `regoptions'
        local opttoken=1
        local nocons=0
        while "``opttoken''"!="" {
        if "``opttoken''"=="noc" local nocons=1
        if "``opttoken''"=="noco" local nocons=1
        if "``opttoken''"=="nocon" local nocons=1
        if "``opttoken''"=="nocons" local nocons=1
        if "``opttoken''"=="noconst" local nocons=1
        if "``opttoken''"=="noconsta" local nocons=1
        if "``opttoken''"=="noconstan" local nocons=1
        if "``opttoken''"=="noconstant" local nocons=1
```

```
if "``opttoken'"=="h" local nocons=1
if "``opttoken'"=="ha" local nocons=1
if "``opttoken'"=="has" local nocons=1
if "``opttoken'"=="hasc" local nocons=1
if "``opttoken'"=="hasco" local nocons=1
if "``opttoken'"=="hascon" local nocons=1
if "``opttoken'"=="hascons" local nocons=1
local ++opttoken
}

if `nocons'==1 display " Regression estimated without constant"
if `nocons'==0 {
display " Regression estimated with constant"
gen cons=1
}
```

Check if the user defined regression options have note about the constant in the OLS regression. If so, create or leave out the constant accordingly.

```
* 3. Generate residuals
*************************

quietly regress `dependvar' `indepvars' if `touse', `regoptions'
quietly predict resid , resid
estimates store myregress
quietly count if `touse'
display "Number of observations `r(N)'"
```

Do the OLS estimation, store the estimations and display the Number of Observations. Generate the residuals.

```
***************

mata: method()

****************
```

Invoke MATA and run algorithm. Detailed later.

```
        quietly estimates restore myregress

        if `nocons'==1 {
        matrix rownames omega= `indepvars'
        matrix colnames omega= `indepvars'
        matrix rownames stderr= `indepvars'
        }
        else {
        matrix rownames omega= `indepvars' cons
        matrix colnames omega= `indepvars' cons
        matrix rownames stderr= `indepvars' cons
        drop cons
        }

        display _newline  "Regression result"
        regress
        display _newline "Newly estimated Std. Error Matrix"
        matrix list stderr

        display _newline "Original Variance-Covariance Matrix"
        matrix list e(V)

        display _newline "Newly estimated Variance Covariance matrix"
        matrix list omega

        merge 1:1  `panels' `time' using "`master'"   , noreport nogen
end
```

MATA environment “sent back” the result in two matrices. Name the columns and rows of the result matrices and display the results to the user.

Merge back sample to the original full database.

```
mata:
                mata clear

void function method()
{
if(strtoreal(st_local("nocons"))==0) {
st_view(X, . , st_macroexpand(st_local("indepvars")+" cons"), )
}
if(strtoreal(st_local("nocons"))==1) {
st_view(X, . , st_macroexpand(st_local("indepvars")), )
}
st_view(Y, . , st_macroexpand(st_local("panels")+" "+st_local("time")+" resid” , )

bigt=rows(Y)
```

Start MATA and clear any leftovers from previous running.

Generate views (matrices with low memory need) from the database. Generate

- X that is the independent variables, and a constant 1 if the OLS regression had one.
- Y that is the matrix of the panel variables, time variable and the OLS residual.
- Store the number of observations in scalar “bigt”

```
/***** VAR MU *****/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

p1_info=panelsetup(Y,1)
p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,2)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)

seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varmu=outside*inside

if(varmu<0) {
varmu=0
}
printf(" Varmu \n")
varmu
```

Variance component estimation. Variance mu.

Sort database (therefore the views X and Y as well) according to the now important variables: panel1, panel2 and latest the time variable.

Step 1.

a. Segment the database (actually the view Y) according to the different values of panel1. (segments called SY1)
b. generate an empty matrix with 2 columns and as many rows as many segments created in 1a. step. (matrix called middlemean)

Step 2. For each if the above created segments.

a. Segment the submatrices according to the different values of panel2. New sub of submatrices called SY2.

b. Generate an empty matrix with 2 columns and as many rows as many segment now created. (matrix called innermean)

Step 3. For each of the SY2 sub-submatrices of the current SY1 submatrix.

a. Store the mean of residuals in the first element of the corresponding row of the "innermean" matrix.

$$\frac{1}{T_{ij}}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*}$$

b. Store the reciprocal of the number of elements in the second element of the corresponding row of the "innermean" matrix.

$$\frac{1}{T_{ij}}$$

Step 4. After the loop is over, therefore finished the last SY2 sub-sub matrix of the current SY1 submatrix and the current "innermean" matrix is full:

a. Store the mean if the squares of the values of first column if "innermean" in the first column of the corresponding row in matrix "middlemean". Correspondence is based on the order if SY1.

$$\frac{1}{N_i}\sum_{j=1}^{N}\left(\frac{1}{T_{ij}}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*}\right)^{2}$$

b. Sum the values of the second column of "innermean" and multiply it with the reciprocal of the number of row of rows in "innermean"(e.g. is number if sub-sub semgment created.) Store in the second column of the corresponding row in matrix "middlemean"

$$\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}$$

Step 5. After all SY1 sub matrices is finished, therefore matrix "middlemean" is full.

a. Generate scalar "firstpart", that is the mean of the first column of matrix "middlemean".

$$\frac{1}{N^{**}}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\left(\frac{1}{T_{ij}}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*}\right)^{2}$$

b. Generate scalar "seconddenominator1" that is the sum of the second column if matrix "middlemean".

$$\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}$$

c. Generate scalar "seconddenominator2", that is the reciprocal of the number of sub segments SY1 multiplied by the number of all observations.

$$\frac{1}{N^{**}T}$$

d. Generate scalar "seconddenominator" that is "seconddenominator1" multiplied by "seconddenominator2".

$$\frac{1}{N^{**}T}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}$$

e. Generate scalar “secondtotal” that is the sum all the square of the residuals for all the observations.

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*^2}$$

f. Generate scalar “second” that is “seconddenominator” multiplied by “secondtotal”

$$\frac{1}{N^{**}T}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*^2}$$

g. Generate scalar “inside” that is “firstpart” – “second”

$$\frac{1}{N^{**}}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\left(\frac{1}{T_{ij}}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*}\right)^2-\frac{1}{N^{**}T}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*^2}$$

h. Generate outside.

$$\frac{1}{1-\frac{1}{N^{**}}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}}$$

i. Generate Variance component mu, that is “outside” multiplied by “inside”.

$$\frac{1}{1-\frac{1}{N^{**}}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}}\left(\frac{1}{N^{**}}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\left(\frac{1}{T_{ij}}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*}\right)^2-\frac{1}{N^{**}T}\sum_{i=1}^{N}\frac{1}{N_i}\sum_{j=1}^{N}\frac{1}{T_{ij}}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=1}^{T_{ij}}\hat{u}_{ijt}^{*^2}\right)$$

j. If the variance component is smaller than 0, replace it with 0. Display the result to the user.

As the Variance Component U and V are symmetric, the difference is:

- The sorting order changes in the beginning
- The segmentation is based on the corresponding variables, e.g. panel1-time and panel2-time.

All the work in progress estimations are based on relative numbers (apart from the number of all observations) therefore the symmetry assures that the estimations are good.

```
/***** VAR VU *****/

stata("sort "+st_local("panel1")+" "+st_local("time")+" "+st_local("panel2") )
p1_info=panelsetup(Y,1)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)

seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varvu=outside*inside

if(varvu<0) {
varvu=0
}
printf(" Var Vu seconddenomnators \n")
(1/panelstats(p1_info)[1])
seconddenominator1
seconddenominator2
seconddenominator
inside
outside
printf(" Var Vu \n")
varvu
```

```
/***** VAR UU *****/

stata("sort "+st_local("panel2")+" "+st_local("time")+" "+st_local("panel1") )
p1_info=panelsetup(Y,2)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)


seconddenominator=seconddenominator1*seconddenominator2

secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varuu=outside*inside
if(varuu<0) {
varuu=0
}
printf(" Var Uu \n")
varuu
```

```
/***** STRUCTURE MATRIX **********/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

Xmod=J(bigt,cols(X),.)

        printf("Structure Matrix Started  "+st_global("c(current_time)")+ " \n")
        readypercent=5
        for(i=1;i<=bigt;i++) {

                rprow=J(1,bigt,0)

                /* Adding Varmu part -- Structure Matrix B*/
                /* Equality at the panel variables */
                keep1=Y[.,1]:==Y[i,1]
                keep2=Y[.,2]:==Y[i,2]
                keepmod=(keep1+keep2)[.,1]:==2
                rprow=rprow+varmu*keepmod'

                /* Adding Varuu part -- Structure Matrix C */
                /* Equality at the 2nd panel variable and the Time*/
                keep3=Y[.,3]:==Y[i,3]
                keepmod=(keep2+keep3)[.,1]:==2
                rprow=rprow+varuu*keepmod'

                /* Adding Varvu part -- Structure Matrix D */
                /* Equality at the 1st panel variable and the Time*/
                keepmod=(keep1+keep3)[.,1]:==2
                rprow=rprow+varvu*keepmod'

                /* Adding Varee part -- Structure Matrix It*/
                rprow[i]=rprow[i]+varee

                /*Replaceing row in Xmod */

                Xmod[i,.]=rprow*X

                if(mod(i,round(0.05*bigt))==0) {
                printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                readypercent=readypercent+5
                }
        }
        printf("100 percent  "+st_global("c(current_time)")+ " \n")
```

The Structure Matrix is used only in the (X' E X) multiplication. To reduce memory use we don't estimate the E matrix by itself, just the EX matrix as a first step. (This is called Xmod in the program) The EX matrix is built row-by-row, generating one row of the E matrix each time, then promptly multiplying that row with the matrix X (the matrix of dependent variables), and we store only the ever building EX matrix.

It is proven that Stata fills in faster an empty matrix than adds rows to an existing one; therefore the first step is to create the empty EX (Xmod in the algorithm) matrix.

In the corresponding paper the structure matrix is built up by the sum of matrices B, C ,D and I_t, therefore the rows of the structure matrix E are also built up from these components. According to the paper:

- Matrix B in the n-th row has 1 at only those j positions, where the $m_1..m_j$-th element has the same value with the n-th element in panel1 and the same value in panel2.
- Similarly, Matrix C in the n-th row has 1 at only those k positions, where the $m_1..m_k$-th element has the same value with the n-th element in panel2 and the same value in time.
- Also, Matrix D in the n-th row has 1 at only those l positions, where the $m_1..m_l$-th element has the same value with the n-th element in panel1 and the same value in time.
- Matrix E has 1 only in the diagonal.

To create the rows, a loop goes through all the elements. An empty row vector is created at the beginning every time. A signaling vector is created according to above rules and this signaling vector is multiplied by the corresponding variable component, then added to the row vector.

This signaling technique made the building the quickest possible method, yet the long loop with constant multiplication makes it the most time consuming part of the program. Therefore regular feedback is given to the user after every 5% of the rows finished.

```
/** Final Estimations */

XX=invsym(cross(X,X))

 middle=cross(X,Xmod)
 omega=XX*middle*XX

cov=diagonal(omega)
stderr=(cov:^0.5)

 st_matrix("omega",omega)
 st_matrix("stderr",stderr)

}

end
```

The new covariance matrix is estimated with parsimonious memory usage. The full covariance matrix and the standard errors alone are “sent back” to the STATA environment. The covariance matrix is estimated as:

$$\Omega = (X'X)^{-1} * (X'\mathrm{E}\,X) * (X'X)^{-1}$$

FEASIBLE GENERAL LEAST SQUARES

The program estimates an FGLS model based on the paper of Laszlo Matyas, Cecilia Hornok and Daria Pus. (http://www.personal.ceu.hu/staff/repec/pdf/2012_1_revised.pdf)

The program display the user the estimation Coefficients, their Standard Error, the t and the P(t) values of the coefficients, and the R^2 value separately.

The estimations are based on that:

$$\beta = (X'\mathrm{E}^{-1} X)^{-1} X \mathrm{E}^{-1} y$$

$$\Omega = (X'\mathrm{E}^{-1} X)^{-1}$$

where X marks the independent variables , y the dependent variable, and E marks the paper-specified variance matrix.

Program is for 3 dimensional data only.

Do file name: fgls5.do

Start command: rpfgls1

Syntax: rpfgls1 *varlist* [if] [in] , Time(time panel variable) Panels(panel variables) [REGOPTions(options for OLS esmitations)]

Syntax means that Statat-regular method could be used to reduce the sample, and the options of Stata “regress” command could be used here as well. Take care, that regress command could estimate more than an OLS regression, in these cases the programs estimations would be erroneous. Time and Panel options are mandatory.

The program is highly depending on the modified OLS program above. The shared parts will be shown here, although without repeating the comments.

ALGORITHM WITH COMMENTS:

```
* Header
**************************
capture noisily program drop rpfgls1

program define rpfgls1 , sortpreserve
        version 12
        #delimit ;
        syntax varlist(min=2 numeric) [if] [in] , Time(varlist numeric max=1)
                        Panels(varlist numeric min=2 max=2)
                                                [
                                                REGOPTions(string asis)
                                                                ]
                                                                ;
        #delimit cr
        marksample touse
        markout `touse' `panels'
        markout `touse' `time'
```

```
        tempfile master
        save `master' , replace


* Check and order data
*****************************

 * No panelvars in regressvars.

        tokenize `panels'
        local token_i=1
                while  "``token_i''"!="" {
                        local panel`token_i'="``token_i''"
                        local ++token_i
                }

        quietly replace `touse'=0 if `panel1'==`panel2'

        keep if `touse'==1
        gen keept=`touse'


        local allvars="`varlist'"
        tokenize `varlist'
        local dependvar="`1'"
        local vartoken_i=1
        while "``vartoken_i''"!="" {
                foreach i of numlist 1/`token_i' {
                  if "`panel`i''"=="``vartoken_i''" {
                  display as error " Panelvariable found in the regression varlist." _newline "
Exiting with error."
                error 141
                  }
                }
         local ++vartoken_i
        }

        macro shift
        local indepvars ="`*'"

        display as text "Variable check passed succesfully" _newline

        display "Panel variables              : `panels'"
        display "Time variable                : `time'"
        display "Dependent variable           : `dependvar' "
        display "Independent variables        : `indepvars'"
        display "Regression options           : `regoptions'" _newline
```

```
** Acquireing maxs

        *tempvar tvar
        quietly egen tvar=group(`time') if `touse'
        quietly sum tvar if `touse'
        local tmax=r(max)
        drop tvar

        *tempvar ivar
        quietly egen ivar=group(`panel1') if `touse'
        quietly sum ivar if `touse'
        local imax=r(max)
        drop ivar

        *tempvar jvar
        quietly egen jvar=group(`jpanel') if `touse'
        quietly sum jvar if `touse'
        local jmax=r(max)
        drop jvar

if `imax'>`jmax' {
        local nmax=`imax'
        }
        else {
        local nmax=`jmax'
        }

** Hasnocons

        tokenize `regoptions'
        local opttoken=1
        local nocons=0
        while "``opttoken'"!="" {
        if "``opttoken'"=="noc" local nocons=1
        if "``opttoken'"=="noco" local nocons=1
        if "``opttoken'"=="nocon" local nocons=1
        if "``opttoken'"=="nocons" local nocons=1
        if "``opttoken'"=="noconst" local nocons=1
        if "``opttoken'"=="noconsta" local nocons=1
        if "``opttoken'"=="noconstan" local nocons=1
        if "``opttoken'"=="noconstant" local nocons=1

        if "``opttoken'"=="h" local nocons=1
        if "``opttoken'"=="ha" local nocons=1
        if "``opttoken'"=="has" local nocons=1
        if "``opttoken'"=="hasc" local nocons=1
        if "``opttoken'"=="hasco" local nocons=1
        if "``opttoken'"=="hascon" local nocons=1
        if "``opttoken'"=="hascons" local nocons=1
        local ++opttoken
        }

        if `nocons'==1 display " Regression estimated without constant"
        if `nocons'==0 {
        display " Regression estimated with constant"
        gen cons=1
        }
```

```
* 3. Generate residuals
************************

        quietly regress `dependvar' `indepvars' if `touse', `regoptions'
        quietly predict resid , resid
        estimates store myregress
        quietly count if `touse'
        display "Number of observations `r(N)'"

****************

        mata: method()

****************
if `nocons'==1 {
        matrix rownames beta= `indepvars'
        *matrix colnames cov= covariance
        *matrix rownames cov= `indepvars'
        matrix rownames results= `indepvars'
        }
        else {
        matrix rownames beta= `indepvars' cons
        *matrix colnames cov= covariance
        *matrix rownames cov= `indepvars' cons
        matrix rownames results= `indepvars' cons
        matrix colnames results= "Beta" "Std_Err" "t" "P(t)"

        drop cons
        }

        quietly estimates restore myregress

        display _newline  "Regression result"
        regress

        display _newline "FGLS results"
        matrix list results

        display _newline "R2 of FGLS"
        matrix list R2

        merge 1:1  `panels' `time' using "`master'"   , noreport nogen
end
```

As the results are different from the modified OLS program, in this last part the displaying has changed, according to the necessary differences.

```
mata:
                mata clear

void function method()
{
if(strtoreal(st_local("nocons"))==0) {
st_view(X, . , st_macroexpand(st_local("indepvars")+" cons"),/* st_local("touse")*/ )
}
if(strtoreal(st_local("nocons"))==1) {
st_view(X, . , st_macroexpand(st_local("indepvars")), /* st_local("touse") */)
}
st_view(Y, . , st_macroexpand(st_local("panels")+" "+st_local("time")+" resid ")/* varmu
varuu varvu varee  ")*/, /*st_local("touse") */)
st_view(Depvar, . , st_macroexpand(st_local("dependvar")), /* st_local("touse") */)
bigt=rows(Y)
```

The program has the same matrices as the modified OLS program, with an additional view (column vector) of the dependent variable.

```
/**** VAR MU *****/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

p1_info=panelsetup(Y,1)
p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,2)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)
```

```
seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varmu=outside*inside

if(varmu<0) {
varmu=0
}
printf(" Varmu \n")
varmu
```

```
/***** VAR VU *****/

stata("sort "+st_local("panel1")+" "+st_local("time")+" "+st_local("panel2") )
p1_info=panelsetup(Y,1)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)
seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varvu=outside*inside

if(varvu<0) {
varvu=0
}
printf(" Var Vu \n")
varvu
```

```
/**** VAR UU *****/

stata("sort "+st_local("panel2")+" "+st_local("time")+" "+st_local("panel1") )
p1_info=panelsetup(Y,2)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=(sum(p2_innermean[.,2]))*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)


seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varuu=outside*inside
if(varuu<0) {
varuu=0
}
printf(" Var Uu \n")
varuu
```

```
/***** Var Ee *****/

secondtotal=sum(Y[.,4]:^2)
second=secondtotal/bigt
varee=second-varmu-varvu-varuu
if(varee<0) {
varee=0
}
printf(" Var Ee \n")
varee
```

The variance components and the structure matrix is exactly the same as in the modified OLS program.

```
/****** STRUCTURE MATRIX **********/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

        printf("Structure Matrix Started  "+st_global("c(current_time)")+ " \n")
        readypercent=5

Sigma=J(bigt,bigt,.)
for(i=1;i<=bigt;i++) {

                rprow=J(1,bigt,0)

                /* Adding Varmu part -- Structure Matrix B*/
                /* Equality at the panel variables */
                keep1=Y[.,1]:==Y[i,1]
                keep2=Y[.,2]:==Y[i,2]
                keepmod=(keep1+keep2)[.,1]:==2
                rprow=rprow+varmu*keepmod'

                /* Adding Varuu part -- Structure Matrix C */
                /* Equality at the 2nd panel variable and the Inner order*/
                keep3=Y[.,3]:==Y[i,3]
                keepmod=(keep2+keep3)[.,1]:==2
                rprow=rprow+varuu*keepmod'

                /* Adding Varvu part -- Structure Matrix D */
                /* Equality at the 1st panel variable and the Inner order*/
                keepmod=(keep1+keep3)[.,1]:==2
                rprow=rprow+varvu*keepmod'

                /* Adding Varee part -- Structure Matrix It*/
                rprow[i]=rprow[i]+varee

                Sigma[i,.]=rprow

                if(mod(i,round(0.05*bigt))==0) {
                printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                readypercent=readypercent+5
                }
}
```

In the FGLS model the E matrix in inverted. Although there might be methods to invert the matrix in separate blocks, this program uses the built-in MATA command, and that needs the matrix to exist in full. Therefore no memory saving is possible with a row-by-row process.

The program therefore fills in the Sigma matrix row-by-row, based on the same method as used in the modified OLS program.

```
invSigma=invsym(Sigma)
printf(" Sigma inverted "+st_global("c(current_time)")+ " \n")
Sigma=.
printf(" Sigma dropped "+st_global("c(current_time)")+ " \n")
XS=cross(X,invSigma)
printf(" XS ready "+st_global("c(current_time)")+ " \n")
XSX=XS*X
printf(" XSX "+st_global("c(current_time)")+ " \n")
XS=.
printf(" XS dropped "+st_global("c(current_time)")+ " \n")
invXSX=invsym(XSX)
printf(" XSX inverted "+st_global("c(current_time)")+ " \n")
XSX=.
printf(" XSX dropped "+st_global("c(current_time)")+ " \n")
cov=diagonal(invXSX)
printf(" cov ready "+st_global("c(current_time)")+ " \n")
stderr=(cov:^0.5)
printf(" std. errors ready "+st_global("c(current_time)")+ " \n")
beta=invXSX*cross(X,invSigma)*Depvar
printf(" beta ready "+st_global("c(current_time)")+ " \n")
tees=beta:/stderr
pees=tden(bigt-1,tees)
printf(" T and P(T) ready "+st_global("c(current_time)")+ " \n")
yhat=X*beta
errterm1=Depvar:-yhat
errterm=errterm1:^2
sse=sum(errterm)
ymean=mean(Depvar)
totterm1=Depvar:-ymean
totterm=totterm1:^2
sst=sum(totterm)
R2=1-(sse/sst)
printf(" R2 ready "+st_global("c(current_time)")+ " \n")

 st_matrix("beta",beta)
 st_matrix("stderr",stderr)
 st_matrix("results",(beta,stderr,tees,pees))
 st_matrix("R2",R2)
}

end
```

The matrix inversion consumes the most of the process time, but the program keeps the user informed of every step. The calculations are based on the formulae stated in the introduction and as a last step the program estimates the R^2 and passes all the results back to STATA for display.

```
**********************************
** 3D FGLs estimation            **
** Peter Revesz 27.04.2013       **
** peterrevesz86@gmail.com       **
**********************************

* Header
*************************
capture noisily program drop rpfgls1

program define rpfgls1 , sortpreserve
        version 12
        #delimit ;
        syntax varlist(min=2 numeric) [if] [in] , Time(varlist numeric max=1)
                                                  Panels(varlist numeric
min=2 max=2)
                                                  [
                                                  REGOPTions(string asis)
                                                  ]
                                                  ;
        #delimit cr
        marksample touse
        markout `touse' `panels'
        markout `touse' `time'

        tempfile master
        save `master' , replace


* Check and order data
*****************************

 * No panelvars in regressvars.

        tokenize `panels'
        local token_i=1
```

```
		while  "``token_i''"!="" {
			local panel`token_i'="``token_i''"
			local ++token_i
		}

	quietly replace `touse'=0 if `panel1'==`panel2'

	keep if `touse'==1
	gen keept=`touse'



	local allvars="`varlist'"
	tokenize `varlist'
	local dependvar="`1'"
	local vartoken_i=1
	while "``vartoken_i''"!="" {
		foreach i of numlist 1/`token_i' {
		 if "`panel`i''"=="``vartoken_i''" {
		 display as error " Panelvariable found in the regression varlist." _newline "
Exiting with error."
		error 141
		 }
		}
	 local ++vartoken_i
	}

	macro shift
	local indepvars ="`*'"

	display as text "Variable check passed succesfully" _newline

	display "Panel variables	: `panels'"
	display "Time variable		: `time'"
	display "Dependent variable		: `dependvar' "
	display "Independent variables	: `indepvars'"
	display "Regression options	: `regoptions'" _newline
```

** Acquireing maxs

```
        *tempvar tvar
        quietly egen tvar=group(`time') if `touse'
        quietly sum tvar if `touse'
        local tmax=r(max)
        drop tvar

        *tempvar ivar
        quietly egen ivar=group(`panel1') if `touse'
        quietly sum ivar if `touse'
        local imax=r(max)
        drop ivar

        *tempvar jvar
        quietly egen jvar=group(`jpanel') if `touse'
        quietly sum jvar if `touse'
        local jmax=r(max)
        drop jvar

if `imax'>`jmax' {
        local nmax=`imax'
        }
        else {
        local nmax=`jmax'
        }
```

** Hasnocons

```
        tokenize `regoptions'
        local opttoken=1
        local nocons=0
        while "``opttoken'"!="" {
        if "``opttoken'"=="noc" local nocons=1
        if "``opttoken'"=="noco" local nocons=1
        if "``opttoken'"=="nocon" local nocons=1
        if "``opttoken'"=="nocons" local nocons=1
```

```
        if "``opttoken''"=="noconst" local nocons=1
        if "``opttoken''"=="noconsta" local nocons=1
        if "``opttoken''"=="noconstan" local nocons=1
        if "``opttoken''"=="noconstant" local nocons=1

        if "``opttoken''"=="h" local nocons=1
        if "``opttoken''"=="ha" local nocons=1
        if "``opttoken''"=="has" local nocons=1
        if "``opttoken''"=="hasc" local nocons=1
        if "``opttoken''"=="hasco" local nocons=1
        if "``opttoken''"=="hascon" local nocons=1
        if "``opttoken''"=="hascons" local nocons=1
        local ++opttoken
        }

        if `nocons'==1 display " Regression estimated without constant"
        if `nocons'==0 {
        display " Regression estimated with constant"
        gen cons=1
        }

* 3. Generate residuals
*************************

        quietly regress `dependvar' `indepvars' if `touse', `regoptions'
        quietly predict resid , resid
        estimates store myregress
        quietly count if `touse'
        display "Number of observations `r(N)'"

****************

        mata: method()

****************
if `nocons'==1 {
        matrix rownames beta= `indepvars'
```

```
        *matrix colnames cov= covariance
        *matrix rownames cov= `indepvars'
        matrix rownames results= `indepvars'
        }
        else {
        matrix rownames beta= `indepvars' cons
        *matrix colnames cov= covariance
        *matrix rownames cov= `indepvars' cons
        matrix rownames results= `indepvars' cons
        matrix colnames results= "Beta" "Std_Err" "t" "P(t)"

        drop cons
        }

        quietly estimates restore myregress

        display _newline  "Regression result"
        regress

        display _newline "FGLS results"
        matrix list results

        display _newline "R2 of FGLS"
        matrix list R2

        merge 1:1  `panels' `time' using "`master'"   , noreport nogen
end

mata:
                mata clear

void function method()
{
if(strtoreal(st_local("nocons"))==0) {
st_view(X, . , st_macroexpand(st_local("indepvars")+" cons"),/* st_local("touse")*/ )
}
if(strtoreal(st_local("nocons"))==1) {
```

```
st_view(X, . , st_macroexpand(st_local("indepvars")), /* st_local("touse") */)
}
st_view(Y, . , st_macroexpand(st_local("panels")+" "+st_local("time")+" resid ")/* varmu varuu varvu varee ")*/, /*st_local("touse") */)
st_view(Depvar, . , st_macroexpand(st_local("dependvar")), /* st_local("touse") */)
bigt=rows(Y)

/**** VAR MU *****/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

p1_info=panelsetup(Y,1)
p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,2)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)

seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
```

```
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varmu=outside*inside

if(varmu<0) {
varmu=0
}
printf(" Varmu \n")
varmu


/**** VAR VU *****/

stata("sort "+st_local("panel1")+" "+st_local("time")+" "+st_local("panel2") )
p1_info=panelsetup(Y,1)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)
```

```
seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varvu=outside*inside

if(varvu<0) {
varvu=0
}
printf(" Var Vu \n")
varvu

/**** VAR UU *****/

stata("sort "+st_local("panel2")+" "+st_local("time")+" "+st_local("panel1") )
p1_info=panelsetup(Y,2)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
                p1_middlemean[i,2]=(sum(p2_innermean[.,2]))*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
```

```
	seconddenominator2=1/(panelstats(p1_info)[1]*bigt)



seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varuu=outside*inside
if(varuu<0) {
varuu=0
}
printf(" Var Uu \n")
varuu

/***** Var Ee ***********/

secondtotal=sum(Y[.,4]:^2)
second=secondtotal/bigt
varee=second-varmu-varvu-varuu
if(varee<0) {
varee=0
}
printf(" Var Ee \n")
varee

/***** STRUCTURE MATRIX **********/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

	printf("Structure Matrix Started  "+st_global("c(current_time)")+ " \n")
	readypercent=5

Sigma=J(bigt,bigt,.)
for(i=1;i<=bigt;i++) {

		rprow=J(1,bigt,0)
```

```
                /* Adding Varmu part -- Structure Matrix B*/
                /* Equality at the panel variables */
                keep1=Y[.,1]:==Y[i,1]
                keep2=Y[.,2]:==Y[i,2]
                keepmod=(keep1+keep2)[.,1]:==2
                rprow=rprow+varmu*keepmod'

                /* Adding Varuu part -- Structure Matrix C */
                /* Equality at the 2nd panel variable and the Inner order*/
                keep3=Y[.,3]:==Y[i,3]
                keepmod=(keep2+keep3)[.,1]:==2
                rprow=rprow+varuu*keepmod'

                /* Adding Varvu part -- Structure Matrix D */
                /* Equality at the 1st panel variable and the Inner order*/
                keepmod=(keep1+keep3)[.,1]:==2
                rprow=rprow+varvu*keepmod'

                /* Adding Varee part -- Structure Matrix It*/
                rprow[i]=rprow[i]+varee

                Sigma[i,.]=rprow

                if(mod(i,round(0.05*bigt))==0) {
                printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
                readypercent=readypercent+5
                }
}
printf("100 percent  "+st_global("c(current_time)")+ " \n")

invSigma=invsym(Sigma)
printf(" Sigma inverted "+st_global("c(current_time)")+ " \n")
Sigma=.
printf(" Sigma dropped "+st_global("c(current_time)")+ " \n")
XS=cross(X,invSigma)
printf(" XS ready "+st_global("c(current_time)")+ " \n")
```

```
XSX=XS*X
printf(" XSX "+st_global("c(current_time)")+ " \n")
XS=.
printf(" XS dropped "+st_global("c(current_time)")+ " \n")
invXSX=invsym(XSX)
printf(" XSX inverted "+st_global("c(current_time)")+ " \n")
XSX=.
printf(" XSX dropped "+st_global("c(current_time)")+ " \n")
cov=diagonal(invXSX)
printf(" cov ready "+st_global("c(current_time)")+ " \n")
stderr=(cov:^0.5)
printf(" std. errors ready "+st_global("c(current_time)")+ " \n")
beta=invXSX*cross(X,invSigma)*Depvar
printf(" beta ready "+st_global("c(current_time)")+ " \n")
tees=beta:/stderr
pees=tden(bigt-1,tees)
printf(" T and P(T) ready "+st_global("c(current_time)")+ " \n")
yhat=X*beta
errterm1=Depvar:-yhat
errterm=errterm1:^2
sse=sum(errterm)
ymean=mean(Depvar)
totterm1=Depvar:-ymean
totterm=totterm1:^2
sst=sum(totterm)
R2=1-(sse/sst)
printf(" R2 ready "+st_global("c(current_time)")+ " \n")

 st_matrix("beta",beta)
 st_matrix("stderr",stderr)
 st_matrix("results",(beta,stderr,tees,pees))
 st_matrix("R2",R2)
}

End
```

```
************************************
** Random effect covariance matrix **
** Peter Revesz 04.27.2013       **
** peterrevesz86@gmail.com     **
************************************

* Header
**************************
capture noisily program drop recov1

program define recov1 , sortpreserve
        version 12
        #delimit ;
        syntax varlist(min=2 numeric) [if] [in] , Time(varlist numeric max=1)
                                                  Panels(varlist numeric
min=2 max=2)
                                                  [
                                                  REGOPTions(string asis)
                                                  ]
                                                  ;
        #delimit cr
        marksample touse
        markout `touse' `panels'
        markout `touse' `time'

        tempfile master
        save `master' , replace


* Check and order data
****************************

 * No panelvars in regressvars.

        tokenize `panels'
```

```
	local token_i=1
		while  "``token_i''"!="" {
			local panel`token_i'="``token_i''"
			local ++token_i
		}

	quietly replace `touse'=0 if `panel1'==`panel2'

	keep if `touse'==1
	gen keept=`touse'



	local allvars="`varlist'"
	tokenize `varlist'
	local dependvar="`1'"
	local vartoken_i=1
	while "``vartoken_i''"!="" {
		foreach i of numlist 1/`token_i' {
		 if "`panel`i''"=="``vartoken_i''" {
		 display as error " Panelvariable found in the regression varlist." _newline "
Exiting with error."
		 error 141
		 }
		 if "`time'"=="``vartoken_i''" {
		 display as error " Time variable found in the regression varlist." _newline "
Exiting with error."
		 error 141
		 }
		}
	 local ++vartoken_i
	}

	macro shift
	local indepvars ="`*'"

	display as text "Variable check passed succesfully" _newline

	display "Panel variables          : `panels'"
```

```
        display "Time variable          : `time'"
        display "Dependent variable             : `dependvar' "
        display "Independent variables          : `indepvars'"
        display "Regression options     : `regoptions'" _newline

** Acquireing max's

        *tempvar tvar
        quietly egen tvar=group(`time') if `touse'
        quietly sum tvar if `touse'
        local tmax=r(max)
        drop tvar

        *tempvar ivar
        quietly egen ivar=group(`panel1') if `touse'
        quietly sum ivar if `touse'
        local imax=r(max)
        drop ivar

        *tempvar jvar
        quietly egen jvar=group(`jpanel') if `touse'
        quietly sum jvar if `touse'
        local jmax=r(max)
        drop jvar

if `imax'>`jmax' {
        local nmax=`imax'
        }
        else {
        local nmax=`jmax'
        }

** Hasnocons

        tokenize `regoptions'
        local opttoken=1
        local nocons=0
```

```
        while "``opttoken''"!="" {
        if "``opttoken''"=="noc" local nocons=1
        if "``opttoken''"=="noco" local nocons=1
        if "``opttoken''"=="nocon" local nocons=1
        if "``opttoken''"=="nocons" local nocons=1
        if "``opttoken''"=="noconst" local nocons=1
        if "``opttoken''"=="noconsta" local nocons=1
        if "``opttoken''"=="noconstan" local nocons=1
        if "``opttoken''"=="noconstant" local nocons=1

        if "``opttoken''"=="h" local nocons=1
        if "``opttoken''"=="ha" local nocons=1
        if "``opttoken''"=="has" local nocons=1
        if "``opttoken''"=="hasc" local nocons=1
        if "``opttoken''"=="hasco" local nocons=1
        if "``opttoken''"=="hascon" local nocons=1
        if "``opttoken''"=="hascons" local nocons=1
        local ++opttoken
        }

        if `nocons'==1 display " Regression estimated without constant"
        if `nocons'==0 {
        display " Regression estimated with constant"
        gen cons=1
        }

* 3. Generate residuals
*************************

        quietly regress `dependvar' `indepvars' if `touse', `regoptions'
        quietly predict resid , resid
        estimates store myregress
        quietly count if `touse'
        display "Number of observations `r(N)'"

***************
```

```
        mata: method()

**********
        quietly estimates restore myregress



        if `nocons'==1 {
        matrix rownames omega= `indepvars'
        matrix colnames omega= `indepvars'
        matrix rownames stderr= `indepvars'
        }
        else {
        matrix rownames omega= `indepvars' cons
        matrix colnames omega= `indepvars' cons
        matrix rownames stderr= `indepvars' cons
        drop cons
        }

        display _newline  "Regression result"
        regress
        display _newline "Newly estimated Std. Error Matrix"
        matrix list stderr

        display _newline "Original Variance-Covariance Matrix"
        matrix list e(V)

        display _newline "Newly estimated Variance Covariance matrix"
        matrix list omega

        merge 1:1  `panels' `time' using "`master'"   , noreport nogen
end

***********************************
* START MATA
*****************************

mata:
```

```
		mata clear

void function method()
{
if(strtoreal(st_local("nocons"))==0) {
st_view(X, . , st_macroexpand(st_local("indepvars")+" cons"), )
}
if(strtoreal(st_local("nocons"))==1) {
st_view(X, . , st_macroexpand(st_local("indepvars")), )
}
st_view(Y, . , st_macroexpand(st_local("panels")+" "+st_local("time")+" resid "), )

bigt=rows(Y)

/**** VAR MU *****/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

p1_info=panelsetup(Y,1)
p1_middlemean=J(panelstats(p1_info)[1],2,.)
	for(i=1;i<=panelstats(p1_info)[1];i++){
		panelsubview(SY1,Y,i,p1_info)

		 p2_info=panelsetup(SY1,2)
		 p2_innermean=J(panelstats(p2_info)[1],2,.)

		for(j=1;j<=panelstats(p2_info)[1];j++){
			panelsubview(SY2,SY1,j,p2_info)
			p2_innermean[j,1]=mean(SY2[.,4])
			p2_innermean[j,2]=1/rows(SY2)
		}

		p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
		p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
	}

	firstpart=mean(p1_middlemean[.,1])
```

```
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)

seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varmu=outside*inside

if(varmu<0) {
varmu=0
}
printf(" Varmu \n")
varmu


/**** VAR VU *****/

stata("sort "+st_local("panel1")+" "+st_local("time")+" "+st_local("panel2") )
p1_info=panelsetup(Y,1)


p1_middlemean=J(panelstats(p1_info)[1],2,.)
        for(i=1;i<=panelstats(p1_info)[1];i++){
                panelsubview(SY1,Y,i,p1_info)

                 p2_info=panelsetup(SY1,3)
                 p2_innermean=J(panelstats(p2_info)[1],2,.)

                for(j=1;j<=panelstats(p2_info)[1];j++){
                        panelsubview(SY2,SY1,j,p2_info)
                        p2_innermean[j,1]=mean(SY2[.,4])
                        p2_innermean[j,2]=1/rows(SY2)
                }

                p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
```

```
		p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
	}



	firstpart=mean(p1_middlemean[.,1])
	seconddenominator1=sum(p1_middlemean[.,2])
	seconddenominator2=1/(panelstats(p1_info)[1]*bigt)



seconddenominator=seconddenominator1*seconddenominator2
secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varvu=outside*inside

if(varvu<0) {
varvu=0
}
printf(" Var Vu \n")
varvu



/**** VAR UU *****/

stata("sort "+st_local("panel2")+" "+st_local("time")+" "+st_local("panel1") )
p1_info=panelsetup(Y,2)

p1_middlemean=J(panelstats(p1_info)[1],2,.)
	for(i=1;i<=panelstats(p1_info)[1];i++){
		panelsubview(SY1,Y,i,p1_info)

		 p2_info=panelsetup(SY1,3)
		 p2_innermean=J(panelstats(p2_info)[1],2,.)

		for(j=1;j<=panelstats(p2_info)[1];j++){
			panelsubview(SY2,SY1,j,p2_info)
```

```
                p2_innermean[j,1]=mean(SY2[.,4])
                p2_innermean[j,2]=1/rows(SY2)
            }

            p1_middlemean[i,1]=mean(p2_innermean[.,1]:^2)
            p1_middlemean[i,2]=sum(p2_innermean[.,2])*(1/rows(SY1))
        }

        firstpart=mean(p1_middlemean[.,1])
        seconddenominator1=sum(p1_middlemean[.,2])
        seconddenominator2=1/(panelstats(p1_info)[1]*bigt)


seconddenominator=seconddenominator1*seconddenominator2

secondtotal=sum(Y[.,4]:^2)
second=(secondtotal*seconddenominator)
inside=firstpart-second
outside=1/(1-(seconddenominator1*(1/panelstats(p1_info)[1])))
varuu=outside*inside
if(varuu<0) {
varuu=0
}
printf(" Var Uu \n")
varuu

/***** Var Ee ***********/

secondtotal=sum(Y[.,4]:^2)
second=secondtotal/bigt
varee=second-varmu-varvu-varuu
if(varee<0) {
varee=0
}
printf(" Var Ee \n")
varee
```

```
/***** STRUCTURE MATRIX **********/

stata("sort "+st_local("panel1")+" "+st_local("panel2")+" "+st_local("time") )

Xmod=J(bigt,cols(X),.)

        printf("Structure Matrix Started  "+st_global("c(current_time)")+ " \n")
        readypercent=5
        for(i=1;i<=bigt;i++) {

                rprow=J(1,bigt,0)

                /* Adding Varmu part -- Structure Matrix B*/
                /* Equality at the panel variables */
                keep1=Y[.,1]:==Y[i,1]
                keep2=Y[.,2]:==Y[i,2]
                keepmod=(keep1+keep2)[.,1]:==2
                rprow=rprow+varmu*keepmod'

                /* Adding Varuu part -- Structure Matrix C */
                /* Equality at the 2nd panel variable and the Inner order*/
                keep3=Y[.,3]:==Y[i,3]
                keepmod=(keep2+keep3)[.,1]:==2
                rprow=rprow+varuu*keepmod'

                /* Adding Varvu part -- Structure Matrix D */
                /* Equality at the 1st panel variable and the Inner order*/
                keepmod=(keep1+keep3)[.,1]:==2
                rprow=rprow+varvu*keepmod'

                /* Adding Varee part -- Structure Matrix It*/
                rprow[i]=rprow[i]+varee

                /*Replaceing row in Xmod */

                Xmod[i,.]=rprow*X
```

```
                if(mod(i,round(0.05*bigt))==0) {
                printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
                readypercent=readypercent+5
                }
        }
        printf("100 percent  "+st_global("c(current_time)")+ " \n")

/** Final Estimations */
XX=invsym(cross(X,X))

 middle=cross(X,Xmod)
 omega=XX*middle*XX

cov=diagonal(omega)
stderr=(cov:^0.5)

 st_matrix("omega",omega)
 st_matrix("stderr",stderr)

}

end
```