Peter Revesz
Any questions to: peterrevesz86@gmail.com 2013.04.27.

ESTIMATION OF 3D FIXED EFFECT PANEL DATA MODELS: PROGRAM DOCUMENTATION

This is the detailed documentation for the estimation of the 3 dimensional fixed effect unbalanced panel data models, using STATA, based on the paper by Laszlo Matyas and Laszlo Balazsi: *The Estimation of Multi-dimensional Fixed Effects Panel Data Models* (version February 18, 2013: http://econpapers.repec.org/paper/ceueconwp/2012_5f2.htm).

Do file: fetrans3.do

The program actually estimates the following equations, utilizing the structure D1, D2, D3 described in the paper (p. 15-16):

$$B = I_t - D1(D1'D1)^{-1}D1'$$

$$C = B - (BD2)[(BD2)'(BD2)]^{-}(BD2)'$$

$$E = C - (CD3)[(CD3)'(CD3)]^{-}(CD3)'$$

$$y_{trans} = E * y$$

Syntax

transform2 *variables to be transoformed , tpanel(time variable) ipanel(first panel variable) jpanel(second panel variable) prefix(name prefix for the new variables) blocksize(width of used matrix blocks)*

The need for this long program is to deal with the burden of the potentially very large amount of data. The estimation above calculates a $[T, T]$ matrix from two other of the same size, where T is the number of all the observations. In STATA environment this means $3 * T * T * 16\, byte$ memory capacity at least. In other programming environment it would still be $3 * T * T * 8\, byte$ from the second equation. In the case of a relatively small panel database of 10,000 observations STATA needs at least 4.5 GB of memory. It is easy to see that memory capacity actually rises with the power of 2 relative to the size of the database.

To solve this problem the program cuts the matrices into smaller pieces. To make estimations quicker each matrix is only fragmented in one dimension: a fragment either has all the rows with only a few columns, or vice versa.

The key to fragmentation is the blocksize option: this specifies how many total columns / rows should be included in each fragment. When using STATA-MP (multi processor), STATA automatically distributes the elemental estimations occurring during a matrix multiplication to all available cores. Also, the I/O processes reaching the HDD are the slowest processes on any computer during these estimations. Therefore it is optimal to use the largest “blocksize” for which your computer can provide enough memory. (That is approximately$4 * T^2 * 16\, byte$)

The strength of this program is the constant feedback and the structure with functions. All the functions display the state of the current step every 5% done. The structure enables to quickly change the starting point (and end point) of the estimations, what is necessary when using large databases.

```
capture noisily program drop transform2

program define transform2
  *version 12
  #delimit ;
  syntax varlist(min=1 numeric) , Tpanel(varlist numeric max=1)
                          [
                          BLOCKsize(integer 10)
                          PREfix(string)
                          Ipanel(varlist numeric max=1)
                          Jpanel(varlist numeric max=1)
                          ]
                          ;
  #delimit cr
```

The header part defines the syntax for the program. The program takes the following arguments:

- a varlist, that should contain the variables that must be transformed
- the time variable
- the first panel variable (i)
- the second panel variable (j)

As optional arguments:

- the prefix the will appear in front of the names of the transformed variables. If not defined it will be "trans_"
- the blocksize, which is the number of rows the computer keeps in parallel in the memory. The default value is 10 rows.

This value is directly connected with the memory capacity of the computer. As most of calculation are done with (T;blocksize) or (blocksize;T) sized matrixes, a big blocksize will mean large memory need. The program stores 4 of the above sized matrixes at a time, therefore be sure that memory of your computer exceeds 4*16*T*blocksize bytes. Where T is the number of observations, and Stata stores each number in 16 bytes. (double format).

```
* 2. Check for required characteristics

* save starting order
noisily display  "STARTED AT :"
noisily display  c(current_time)
quietly desc , varlist
local varorder=r(varlist)
local maxvar=r(k)
```

As reporting to the user is a priority in this program, the program starts with displaying the time of start. Then stores the order of variables for later use, and the number of variables as well.

```
tokenize `varlist'
local allvars `*'
local v=1

capture {
	foreach var of varlist `allvars' {
		count if `var'==.
		if r(N)>0 {
		display as error "Missing values found in `var'. Missing values deleted casewise. "
		noisily drop if `var'==.
		}
	}
}
```

Next step is parse the list of variables given, and sort out the observation where any of them has a missing value.

```
if "`prefix'"=="" {
	local prefix = "trans"
}
```

Check if a prefix is given in the options, if not, then use “trans” as a prefix.

```
*tempvar tvar
egen tvar=group(`tpanel')
capture count if `tpanel'==.
if r(N)>0 {
	display as error "Missing values in panel variable `tpanel'"
	exit
}
capture {
	sum tvar
	local tmax=r(max)
}
```

```
*tempvar ivar
capture count if `ipanel'==.
if r(N)>0 {
	display as error "Missing values in panel variable `ipanel'"
	exit
}
capture {
	egen ivar=group(`ipanel')
	sum ivar
	local imax=r(max)
}
*tempvar jvar
capture count if `jpanel'==.
if r(N)>0 {
	display as error "Missing values in panel variable `jpanel'"
	exit
}
capture {
	egen jvar=group(`jpanel')
	sum jvar
	local jmax=r(max)
}
*capture compare ivar jvar
```

Rescale the time and panel variables one by one. The lowest value occurred will be 1 in each variable , and each different value will be mark in an ascending order. This makes the program independent of the actual values of panel and time variables.
Count the number of different values in each panel variable.

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

Mark the highest of the number of different values in the panel variables as the “nmax”, the number used during the estimations as N.

```
tempfile master
save `master' , replace

keep ivar jvar tvar `allvars'
rename tvar `tpanel'
rename ivar `ipanel'
rename jvar `jpanel'
sort `tpanel' `ipanel' `jpanel'
putmata panelvars =(`tpanel' `ipanel' `jpanel' `var') , replace // changed statvars to panelvars!
putmata datavars =(`allvars'), replace
```

Save the database as it is, then keep only the panel variables, the time variable and the variables that should be transformed for further estimations.

Rename the variables – this is actually just technical.

Create and make reachable two matrices: the matrix of the panel variables and time variables and the variables that should be transformed.

```
****************
  mata: method(panelvars, datavars)
*****************
```

Start mata estimations. Continue when MATA is finished.

```
tokenize `allvars'
local testnr=1
while "``testnr''"!="" {
        local maxvar2=`testnr'
        local testnr=`testnr'+1
}
```

Count the number of variables to be transformed.

```
capture noisily {
        foreach varnr of numlist 1/`maxvar2' {
                display "finalysing started for variable `varnr' "
                display c(current_time)
                local var="``varnr''"
                rename `prefix'_`varnr' `prefix'_`var'
                sum `var', meanonly
                local z=r(mean)*10^-10
                replace `prefix'_`var'=round(`prefix'_`var',`z')
                display "finalysing finished for variable `var' "
                display c(current_time)
        }
}
```

For each variable transformed, rename the MATA outputted variable to use the predefined prefix. Round the variable, to the E-10-th value of the mean of the average after the transformation. Send feedback to the user after every variable finished this way.

```
rename `tpanel' tvar
rename `ipanel' ivar
rename `jpanel' jvar
merge 1:1 tvar ivar jvar using `master' , nogenerate noreport
save `master', replace

//varlist finish
```

Rename back the rescaled time and panel variables, and merge the new database back to the starting one.

```
  order `varorder'
  drop tvar ivar jvar
  noisily display  "FINISHED AT :"
  noisily display  c(current_time)
end
```

Reorder variables as they were in the beginning, drop the rescaled time and panel variables, and show finishing time to the user.

MATA PART

```
mata:
        mata clear
```

Clear any object from previous estimations.

THE FRAMEWORK FUNCTION.

This is the framework function, that calls the steps on by one. The steps are handmade functions to replace the basic MATA functions. The only reason for the replacement that the size of the database is too large to be processed solely in the computer memory, and the matrices have to be fragmented to pieces. The functions are:

- rpgen_XX – creates a predefined matrix, with the name in XX.
- Rprowtocol -- rearranges the fragmentation, from row wise pieces to column wise pieces. This step is needed for the matrix used on the right side of a multiplication.
- Rpmultiply – multiplies the first matrix (left side) with the second matrix (right side)
- Rpinv and rpqinv – symmetric inversion and QR inversion of the matrix.
- Rpunlink – delete the matrix
- Rpaltexport – create and export the transformed variable to the STATA environment
- Rpcheckcontent – not used during healthy running. Accessory function used for debugging.

```
void function method(real matrix panelvars, real matrix datavars)
{
        imax=strtoreal(st_local("nmax"))
        tmax=strtoreal(st_local("tmax"))
        blocksize=strtoreal(st_local("blocksize"))
        bigt=rows(panelvars)
```

Call the arguments from Stata:

- the largest number of different panel variables
- the number of different time values
- the program argument blocksize for the estimations
- the number of all the observations.

Call the steps (functions) one-by-one:

```
/*1*/	rpgen_d1row(panelvars, imax, bigt, blocksize)
/*2*/	rprowtocol("d1row", "d1col", bigt, imax*imax, blocksize)
/*3*/	rpmultiply("d1col", "d1col", "d1'd1row", imax*imax, imax*imax,blocksize)
/*4*/	rpinv("d1'd1row", "invd1'd1row", imax*imax,blocksize)
/*5*/	rpunlink("d1'd1row",imax*imax,blocksize)
/*6*/	rpmultiply("d1row","invd1'd1row","d1invd1'd1row",bigt, imax*imax,blocksize)
/*7*/	rpunlink("invd1'd1row",imax*imax,blocksize)
/*8*/	rpmultiply("d1invd1'd1row","d1row","d1invd1'd1d1'row",bigt, bigt,blocksize)
/*9*/	rpunlink("d1row",bigt,blocksize)
/*10*/	rpunlink("d1invd1'd1row",bigt,blocksize)
/*11*/	rpunlink("d1col",imax*imax, blocksize)
/*12*/	rpgen_b(bigt,blocksize)
/*13*/	rpunlink("d1invd1'd1d1'row",bigt,blocksize)
/*14*/	rpgen_d2row(panelvars, imax, bigt, tmax,blocksize)
/*15*/	rprowtocol("d2row", "d2col", bigt, tmax*imax,blocksize)
/*16*/	rpunlink("d2row", bigt,blocksize)
/*17*/	rpmultiply("b_row","d2col","bd2row",bigt, tmax*imax,blocksize)
/*18*/	rpunlink("d2col",tmax*imax,blocksize)
/*19*/	rprowtocol("bd2row", "bd2col", bigt, tmax*imax,blocksize)
/*20*/	rpmultiply("bd2col","bd2col","bd2'bd2row",tmax*imax, tmax*imax,blocksize)
/*21*/	rpqinv("bd2'bd2row", "qinvbd2'bd2row", tmax*imax, 1000,blocksize)
/*22*/	rpunlink("bd2'bd2row",tmax*imax,blocksize)
/*23*/	rpmultiply("bd2row","qinvbd2'bd2row","bd2qinvbd2'bd2row",bigt,
tmax*imax,blocksize)
/*24*/	rpunlink("qinvbd2'bd2row",tmax*imax,blocksize)
/*25*/	rpmultiply("bd2qinvbd2'bd2row","bd2row","bd2qinvbd2'bd2bd2'row",bigt,
bigt,blocksize)
```

```
/*26*/	rpunlink("bd2qinvbd2'bd2row",bigt,blocksize)
/*27*/	rpunlink("bd2row", bigt,blocksize)
/*28*/	rpunlink("bd2col",tmax*imax,blocksize)
/*29*/	rpgen_c(bigt,blocksize)
/*30*/	rpunlink("bd2qinvbd2'bd2bd2'row", bigt,blocksize)
/*31*/	rpunlink("b_row",bigt,blocksize)
/*32*/	rpgen_d3row(panelvars, bigt, tmax, imax,blocksize)
/*33*/	rprowtocol("d3row", "d3col", bigt, tmax*imax,blocksize)
/*34*/	rpunlink("d3row", bigt,blocksize)
/*35*/	rpmultiply("c_row","d3col","cd3row",bigt, tmax*imax,blocksize)
/*36*/	rpunlink("d3col",tmax*imax,blocksize)
/*37*/	rprowtocol("cd3row", "cd3col", bigt, tmax*imax,blocksize)
/*38*/	rpmultiply("cd3col","cd3col","cd3'cd3row",tmax*imax, tmax*imax,blocksize)
/*39*/	rpunlink("cd3col",tmax*imax,blocksize)
/*40*/	rpqinv("cd3'cd3row", "qinvcd3'cd3row", tmax*imax, 100,blocksize)
/*41*/	rpunlink("cd3'cd3row",tmax*imax,blocksize)
/*42*/rpmultiply("cd3row","qinvcd3'cd3row","cd3qinvcd3'cd3row",bigt,tmax*imax,blocksize)
```

```
/*43*/   rpunlink("qinvcd3'cd3row", tmax*imax, blocksize)
/*44*/   rpmultiply(
"cd3qinvcd3'cd3row","cd3row","cd3qinvcd3'cd3cd3'row",bigt,bigt,blocksize)
/*45*/   rpunlink("cd3qinvcd3'cd3row",bigt,blocksize)
/*46*/   rpgen_e(bigt,blocksize)
/*47*/   rpunlink("cd3qinvcd3'cd3cd3'row",bigt,blocksize)
/*48*/   rpunlink("cd3row",bigt,blocksize)
/*49*/   rpaltexport(datavars,bigt,blocksize)
/*50*/   rpunlink("e_row",bigt,blocksize)
/*51*/   rpunlink("c_row",bigt,blocksize)
}
```

CHECK CONTENT FUNCTION.

```
void function rpcheckcontent (string scalar mattocheck, real scalar i )
{
  fh = fopen(st_macroexpand(mattocheck+strofreal(i)+".myfile") , "r" , 1 )
                                rprow=fgetmatrix(fh)
                                rpcol=fgetmatrix(fh)

  printf("CONTENT CHECKING START "+mattocheck+strofreal(i)+"\n" )

  printf("rprow \n")
  rprow
  printf("rpcol \n")
  rpcol
  fclose(fh)
  printf("CONTENT CHECKING END "+mattocheck+strofreal(i)+"\n" )
}
```

The check content function is solely for debugging. The function opens the selected part of the selected matrix, and displays it to the user. As all file parts have a column vector and row vector format (with the same content) stored, both of them are displayed.

Syntax:

rpcheckcontent(name of the matrix , number of the checked part)

The function opens the selected file, gets the matrix, prints the start and end time, and prints the matrix.

THE ROW TO COL FUNCTION

```
void function rprowtocol(string scalar oldmatname, string scalar newmatname, real scalar colsize,
real scalar rowsize, real scalar blocksize)
{
		real matrix rprow
		real matrix rpcol
		printf("transforming " + oldmatname + "\n")
	readypercent=5
	for(j=1;j<=ceil(rowsize/blocksize);j++) {
		width=blocksize
		if(j==ceil(rowsize/blocksize)) {
		width=rowsize-(j-1)*blocksize
		}
		rpcol=J(colsize,width,.)
		for(i=1;i<=ceil(colsize/blocksize);i++) {
			fh = fopen(st_macroexpand(oldmatname+strofreal(i)+".myfile") , "r" , 1 )
```

```
			rprow=fgetmatrix(fh)
			a1j=(j-1)*blocksize+1
			b1j=j*blocksize
			if(b1j>rowsize) {
			b1j=rowsize
			}
			x=rprow[|1,a1j \ .,b1j|]
			fclose(fh)
			a2i=(i-1)*blocksize+1
			b2i=i*blocksize
			if(b2i>colsize) {
			b2i=colsize
			}
			rpcol[| a2i,1 \ b2i ,. |]=x
		}
		unlink(st_macroexpand(newmatname+strofreal(j)+".myfile"))
		fh2 = fopen(st_macroexpand(newmatname+strofreal(j)+".myfile") , "w" , 1 )
```

```
		rprow=rpcol'
		fputmatrix(fh2, rprow)
		fputmatrix(fh2, rpcol)
		fclose(fh2)

		if(mod(j,round(0.05*ceil(rowsize/blocksize)))==0) {
		printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
		readypercent=readypercent+5
		}
	}
}
```

The row to col function is the abbreviation of Row to Column. This is a necessary help function, needed when a matrix currently stored fragmented row wise has to be used through its columns.

Syntax:

Rprowtocol(Name of the origin matrix, Name of the new matrix, Number of Columns in the matrix, Number of Rows in the matrix, the blocksize for estimations)

The function opens the row wise fragments, copies a blocksize*blocksize part from it, and copies it to column vector that will be a fragment of the column wise representation of the same matrix. The loop is done until all the column wise fragments are full.

THE MULTIPLY FUNCTION

```
void function rpmultiply(string scalar rowname , string scalar colname, string scalar endrowname,
real scalar newcolsize, real scalar newrowsize, real scalar blocksize)
{
		real rowvector rprow, useasrow, rpnewrow
		real colvector rpcol
		printf("multiplying "+ rowname+ " with "+ colname+ "\n")
  readypercent=5
  for(i=1;i<=ceil(newcolsize/blocksize);i++) {
		height=blocksize
		if(i==ceil(newcolsize/blocksize)) {
			height=newcolsize-(i-1)*blocksize
		}
		rpnewrow=J(height,newrowsize,.)
		fh1 = fopen(st_macroexpand(rowname+strofreal(i)+".myfile") , "r" , 1 )
		rprow=fgetmatrix(fh1)
		fclose(fh1)
		useasrow=rprow
```

```
		for(j=1;j<=ceil(newrowsize/blocksize);j++) {
			fh2 = fopen(st_macroexpand(colname+strofreal(j)+".myfile") , "r" , 1 )
			rprow=fgetmatrix(fh2)
			rpcol=fgetmatrix(fh2)
			fclose(fh2)
			a1i=(i-1)*blocksize+1
			a1j=(j-1)*blocksize+1
			b1i=i*blocksize
			if(b1i>newrowsize) {
				b1i=newrowsize
			}
			b1j=j*blocksize
			if(b1j>newrowsize) {
				b1j=newrowsize
			}
			rpnewrow[|1,a1j \ ., b1j |]=useasrow*rpcol
		}
```

```
            rprow=rpnewrow
            unlink(st_macroexpand(endrowname+strofreal(i)+".myfile"))
            fh3 = fopen(st_macroexpand(endrowname+strofreal(i)+".myfile") , "w" , 1 )
            fputmatrix(fh3, rprow)
            rpcol=rprow'
            fputmatrix(fh3, rpcol)
            fclose(fh3)
            /*****/
            if(mod(i,round(0.05*ceil(newcolsize/blocksize)))==0) {
            printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
            readypercent=readypercent+5
            }
  }
}
```

The multiply function is to replace the MATA built in "*" function with a function that deals with the fragmented files.

*Syntax LeftMatrix * RighMatrix = NewMatrix*

rpmultiply(Name of the matrix on the left, Name of the matrix on the right, Name of the new matrix – stored row wise, Number of columns in the new matrix, Number of row in the new matrix ,Blocksize for estimations)

The function first invokes a row wised stored fragment of the left matrix, then calls all the column wise stored fragments of the matrix on the right one-by-one, to calculate each row wise fragment of the new matrix. The loop is continued until all the rows of the new matrix are ready.

THE INVERSION FUNCTIONS

```
void function rpinv(string scalar oldmatname, string scalar newmatname, real scalar rownumber,
real scalar blocksize) // N2*N2
{
            printf("inverting " + oldmatname + " \n")
            real matrix mattoinv, invmat
            real rowvector rprow
            real colvector rpcol
  mattoinv=J(rownumber, rownumber,.)
  for(i=1;i<=ceil(rownumber/blocksize);i++) {
            fh1 = fopen(st_macroexpand(oldmatname+strofreal(i)+".myfile") , "r" , 1 )
            rprow=fgetmatrix(fh1)
            fclose(fh1)
            ai=(i-1)*blocksize+1
            bi=i*blocksize
            if(bi>rownumber){
                        bi=rownumber
            }
            mattoinv[ai::bi,.]=rprow
  }
```

```
  invmat=invsym(mattoinv)
  for(i=1;i<=ceil(rownumber/blocksize);i++) {
        ai=(i-1)*blocksize+1
        bi=i*blocksize
        if(bi>rownumber){
        bi=rownumber
        }
        rprow=invmat[ai::bi,.]
        unlink(st_macroexpand(newmatname+strofreal(i)+".myfile"))
        fh3 = fopen(st_macroexpand(newmatname+strofreal(i)+".myfile") , "w" , 1 )
        fputmatrix(fh3, rprow)
        rpcol=rprow'
        fputmatrix(fh3, rpcol)
        fclose(fh3)
  }
}
```

```
void function rpqinv(string scalar oldmatname, string scalar newmatname, real scalar
rownumber, real scalar rpedittozero, real scalar blocksize) // N2*N2
{
        printf(" inverting " + oldmatname + " \n")
        real matrix mattoinv, invmat
        real rowvector rprow
        real colvector rpcol
  mattoinv=J(rownumber, rownumber,.)
  for(i=1;i<=ceil(rownumber/blocksize);i++) {
        fh1 = fopen(st_macroexpand(oldmatname+strofreal(i)+".myfile") , "r" , 1 )
        rprow=fgetmatrix(fh1)
        fclose(fh1)
        ai=(i-1)*blocksize+1
        bi=i*blocksize
        if(bi>rownumber){
        bi=rownumber
        }
        mattoinv[ai::bi,.]=rprow
  }
  invmat=edittozero(qrinv(mattoinv),rpedittozero)
  for(i=1;i<=ceil(rownumber/blocksize);i++) {
        ai=(i-1)*blocksize+1
        bi=i*blocksize
        if(bi>rownumber){
              bi=rownumber
        }
        rprow=invmat[ai::bi,.]
        unlink(st_macroexpand(newmatname+strofreal(i)+".myfile"))
        fh3 = fopen(st_macroexpand(newmatname+strofreal(i)+".myfile") , "w" , 1 )
        fputmatrix(fh3, rprow)
        rpcol=rprow'
        fputmatrix(fh3, rpcol)
        fclose(fh3)
  }
}
```

These functions are to estimate the inverse matrixes. “rpinvsym” is to calculate the inverse of a symmetric positive definite matrix. “rpqinv” is to calculate the inverse on any matrix, using the QR method.

Syntax

rpinvsym(Name of matrix to invert, Name of the result matrix, number of rows, blocksize)

rpqinv(Nam of matrix to invert, name of the result matrix, number of rows, tolerance coefficient, blocksize)

Both functions are built on the same design: Read all the fraction files of the matrix to invert and store the whole matrix in memory. Use the MATA built-in function to estimate the appropriate inverse (“invsym” or “qrinv”). Then turn the matrix into the same number of fractions as before.

The function rpqinv calls and extra argument, the tolerance coefficient: the MATA built-in function qrinv estimates the inverse to the double precision: this results that the inverse matrix consists extremely small numbers instead of zeros, where appropriate. The tolerance options defines the level of tolerance: what are the “biggest” numbers that MATA should still treat as 0. Currently 10^-13 is the highest value (1000*10^-16 where 10^-16 is the computing unit), and this could only be changed by modifying the program. No user option is built for this, mainly for safety reasons.

THE UNLINK FUNCTION

```
void function rpunlink(string scalar mattodel, real scalar rownumber, real scalar blocksize)
{
  printf(" unlinking " + mattodel +st_global("c(current_time)") +"\n")
  readypercent=5
  for(i=1;i<=ceil(rownumber/blocksize);i++) {
        unlink(st_macroexpand(mattodel+strofreal(i)+".myfile"))
        if(mod(i,round(0.05*ceil(rownumber/blocksize)))==0) {
              printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
              readypercent=readypercent+5
        }
  }
}
```

The unlink function is to delete the matrices stored in the fraction files.

Syntax

Rpunlink(Name of matrix to delete, number of rows of the matrix, blocksize)

The functions uses the MATA built-in delete function to each and every faction of the matrix.

THE D-X GENERATING FUNCTION

D1 GENERATING

```
void function rpgen_d1row(real matrix panelvars, real scalar imax ,real scalar bigt, real scalar
blocksize)
{
		real scalar helpi, helpj, orderids
		real rowvector rprow
		real colvector rpcol
  printf("generating D1 \n ")
  readypercent=5
  filecounter=1
  for(i=1;i<=bigt;i++) {
		if(mod(i,blocksize)==1) {
			height=blocksize
			if((i-1)+height>bigt) {
				height=bigt-i+1
			}
			rprow=J(height,imax*imax,.)
		}
		helpi=panelvars[i,2]
		helpj=panelvars[i,3]
		orderids=(helpi-1)*imax+helpj
		rprowtemp=e(orderids, imax*imax )
		rowposition=i-(filecounter-1)*blocksize
		if(mod(i,blocksize)>=1) {
			rprow[rowposition,.]=rprowtemp
		}
		if(mod(i,blocksize)==0) {
			rprow[rowposition,.]=rprowtemp
			unlink(st_macroexpand("d1row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d1row"+strofreal(filecounter)+".myfile") , "w" , 1
)
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
			filecounter=filecounter+1
		}
		if(i==bigt) {
			unlink(st_macroexpand("d1row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d1row"+strofreal(filecounter)+".myfile") , "w" , 1
)
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
		}
		if(mod(i,round(0.05*bigt))==0) {
		printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
		readypercent=readypercent+5
		}
  }
}
```

The D1 generating function generates the structure matrix named D1 in the corresponding paper. This matrix is the structure for the effect γ_{ij}.

The D2 generating function generates the structure matrix named D2 in the corresponding paper. This matrix is the structure for the effect α_{it}.

The D3 generating function generates the structure matrix named D3 in the corresponding paper. This matrix is the structure for the effect α_{jt}.

Syntax

rpgen_dXrow(Matrix of the panel variables, largest number of different panel values, total number of observations, blocksize)

The function builds the fragment files of the D*X* matrix. First creates a $[blocksize, N^2]$ sized matrix, (or $[rest\ , N^2]$ sized, if the rest of the lines are less than the blocksize) and then fills this matrix row by row.

Each row is actually a unit vector, with the element 1 at the position:

- $N * (i - 1) + j$ in D1 matrix
- $N * (t - 1) + i$ in D2 matrix
- $N * (t - 1) + j$ in D3 matrix

where N is the maximum of the number different values of the panels, and “i” and “j” are the indices of the panels and “t” is index of time. (Remember, each observation was marked as y_{ijt} in the paper)

```
void function rpgen_d2row(real matrix panelvars, real scalar imax ,real scalar bigt, real scalar
tmax, real scalar blocksize)
{
		real scalar helpi, helpt, orderit
		real rowvector rprow
		real colvector rpcol
  printf("generating D2 \n ")
  readypercent=5
  filecounter=1
  for(i=1;i<=bigt;i++) {
		if(mod(i,blocksize)==1) {
			height=blocksize
			if((i-1)+height>bigt) {
				height=bigt-i+1
			}
			rprow=J(height,tmax*imax,.)
		}
		helpt=panelvars[i,1]
		helpi=panelvars[i,2]
		orderit=(helpt-1)*imax+helpi
		rprowtemp=e(orderit, tmax*imax )
		rowposition=i-(filecounter-1)*blocksize
		if(mod(i,blocksize)>=1) {
			rprow[rowposition,.]=rprowtemp
		}
		if(mod(i,blocksize)==0) {
			rprow[rowposition,.]=rprowtemp
			unlink(st_macroexpand("d2row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d2row"+strofreal(filecounter)+".myfile") , "w" , 1
)
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
			filecounter=filecounter+1
		}
		if(i==bigt) {
			unlink(st_macroexpand("d2row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d2row"+strofreal(filecounter)+".myfile") , "w" , 1
)
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
		}
		if(mod(i,round(0.05*bigt))==0) {
			printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
			readypercent=readypercent+5
		}
  }
}
```

```
void function rpgen_d3row(real matrix panelvars, real scalar bigt , real scalar tmax, real scalar
imax, real scalar blocksize)
{
		printf("generating D3 \n")
  readypercent=5
  filecounter=1
  for(i=1;i<=bigt;i++) {
		if(mod(i,blocksize)==1) {
			height=blocksize
			if((i-1)+height>bigt) {
				height=bigt-i+1
			}
			rprow=J(height,tmax*imax,.)
		}
		helpt=panelvars[i,1]
		helpj=panelvars[i,3]
		orderjt=(helpt-1)*imax+helpj
		rprowtemp=e(orderjt, tmax*imax )
		rowposition=i-(filecounter-1)*blocksize
		if(mod(i,blocksize)>=1) {
			rprow[rowposition,.]=rprowtemp
		}
		if(mod(i,blocksize)==0) {
			rprow[rowposition,.]=rprowtemp
			unlink(st_macroexpand("d3row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d3row"+strofreal(filecounter)+".myfile") , "w" , 1
)
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
			filecounter=filecounter+1
		}
		if(i==bigt) {
			unlink(st_macroexpand("d3row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d3row"+strofreal(filecounter)+".myfile") , "w" , 1
)
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
		}
		if(mod(i,round(0.05*bigt))==0) {
			printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
			readypercent=readypercent+5
		}
  }
}
```

THE B, C, E GENERATING FUNCTION

B GENERATING

```
void function rpgen_b(real scalar bigt, real scalar blocksize)
{
  printf("generating B \n")
  readypercent=5
  for(i=1;i<=ceil(bigt/blocksize);i++) {
        fh1 = fopen(st_macroexpand("d1invd1'd1d1'row"+strofreal(i)+".myfile") , "r" , 1 )
        rprow=fgetmatrix(fh1)
        fclose(fh1)
        height=blocksize
        if(i==ceil(bigt/blocksize)){
              height=bigt-(i-1)*blocksize
        }
        rpnewrow=J(height,bigt,.)
        for(k=1;k<=height;k++) {
              l=(i-1)*blocksize+k
              rpnewrow[k,.]=e(l,bigt)-rprow[k,.]
        }
        rprow=rpnewrow
        unlink(st_macroexpand("b_row"+strofreal(i)+".myfile"))
        fh3 = fopen(st_macroexpand("b_row"+strofreal(i)+".myfile") , "w" , 1 )
        fputmatrix(fh3, rprow)
        rpcol=rprow'
        fputmatrix(fh3, rpcol)
        fclose(fh3)
        if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
              printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
              readypercent=readypercent+5
        }
  }
}
```

The B generating matrix finishes the steps with linked first dimension.

Syntax:

rpgen_b(number of total observations, blocksize)

Matrix B is the result of the equation:

$$B = I_t - D1(D1'D1)^{-1}D1'$$

The matrix then reads the fragments of $D1(D1'D1)D1'$ matrix, and calculates the corresponding fragment of B with a simply with subtracting each (n-th) row from a unit row vector (with 1 element in the n-th position).

```
void function rpgen_c(real scalar bigt, real scalar blocksize)
{
		printf("generating C \n")
  readypercent=5
  for(i=1;i<=ceil(bigt/blocksize);i++) {
		fh1 = fopen(st_macroexpand("bd2qinvbd2'bd2bd2'row"+strofreal(i)+".myfile") , "r" , 1
)
		rprow=fgetmatrix(fh1)
		userow1=rprow
		fclose(fh1)
		fh2 = fopen(st_macroexpand("b_row"+strofreal(i)+".myfile") , "r" , 1 )
		rprow=fgetmatrix(fh2)
		userow2=rprow
		fclose(fh2)
		rprow=userow2-userow1
		unlink(st_macroexpand("c_row"+strofreal(i)+".myfile"))
		fh3 = fopen(st_macroexpand("c_row"+strofreal(i)+".myfile") , "w" , 1 )
		fputmatrix(fh3, rprow)
		rpcol=rprow'
		fputmatrix(fh3, rpcol)
		fclose(fh3)
		if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
			printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
		readypercent=readypercent+5
		}
  }
}
```

The C generating matrix finishes the steps with linked second dimension.

Syntax:

rpgen_C(number of total observations, blocksize)

Matrix C is the result of the equation:

$$C = B - (BD2)[(BD2)'(BD2)]^{-}(BD2)'$$

The estimations of the rows of matrix C is similar to matrix B, except the subtraction is from the corresponding rows of matrix B. This means that although matrix B is previously in the beginning of the steps with the 2nd dimension, it has to be preserved until the end of the second dimension.

Worth noting again, that matrix B, C and $(BD2)[(BD2)'(BD2)](BD2)'$ are all $[T,T]$ matrices, where T is the total number of observations. This (and generating matrix E) is the largest pressure and load on the computer, with the need to store (partially in memory and on HDD) $3 * T * T * 16\ byte$ of data. With database a small database of 10,000 observations this is 4.5 GB data!

```
void function rpgen_e(real scalar bigt, real scalar blocksize)
{
  printf("generating E \n")
  readypercent=5
  for(i=1;i<=ceil(bigt/blocksize);i++) {
        fh1 = fopen(st_macroexpand("cd3qinvcd3'cd3cd3'row"+strofreal(i)+".myfile") , "r" , 1 )
        rprow=fgetmatrix(fh1)
        userow1=rprow
        fclose(fh1)
        fh2 = fopen(st_macroexpand("c_row"+strofreal(i)+".myfile") , "r" , 1 )
        rprow=fgetmatrix(fh2)
        userow2=rprow
        fclose(fh2)
        rprow=userow2-userow1
        unlink(st_macroexpand("e_row"+strofreal(i)+".myfile"))
        fh3 = fopen(st_macroexpand("e_row"+strofreal(i)+".myfile") , "w" , 1 )
        fputmatrix(fh3, rprow)
        rpcol=rprow'
        fputmatrix(fh3, rpcol)
        fclose(fh3)
        if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
              printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
              readypercent=readypercent+5
        }
  }
}
```

The E matrix generating finishes the steps with linked third dimension and the total estimating process.

Syntax:

rpgen_E(number of total observations, blocksize)

Matrix E is the result of the equation:

$$E = C - (CD3)[(CD3)'(CD3)]^{-}(CD3)'$$

TRANSFORMING AND EXPORTING FUNCTION

```
void function rpaltexport(real matrix datavars, real scalar bigt, real scalar blocksize)
{
  printf("Exporting started \n")
  readypercent=5
  maxvar=cols(datavars)
  mytostata=J(bigt, maxvar,.)
  for(i=1;i<=ceil(bigt/blocksize);i++) {
        fh1 = fopen(st_macroexpand("e_row"+strofreal(i)+".myfile") , "r" , 1 )
        rprow=fgetmatrix(fh1)
        useasrow=rprow
        fclose(fh1)
        ai=(i-1)*blocksize+1
        bi=i*blocksize
        if(bi>bigt){
        bi=bigt
        }
        mytostata[|ai,1\bi,.|]=useasrow*datavars
        if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
              printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
              readypercent=readypercent+5
        }
  }
  mytostata=edittozero(mytostata,1000)
  for(k=1;k<=maxvar;k++) {
        st_addvar("double", st_local("prefix")+"_"+strofreal(k))
        st_store(.,(st_local("prefix")+"_"+strofreal(k)),mytostata[.,k])
        printf("Exported the "+strofreal(k)+"th variable \n")
  }
  printf("Exporting finished \n ")
}
End
```

This last step executes the transformation using the given E transformation matrix and export the data back to the STATA database.

Syntax

rpaltexport(Matrix with the actual variables, number of total observations, blocksize)

The program in this last step executes the $y_{trans} = E * y$ multiplication from the fragmented E matrix. The transformed variables are then put to the database, where the last STATA statements finish process with a few trimming steps.

DO FILE

```
***************************************************
** 3 dimensional fixed effect transformation   ***
** Peter Revesz 27.04.2013                      ***
** peterrevesz86@gmail.com                      ***
***************************************************


capture noisily program drop transform2

program define transform2
	*version 12
	#delimit ;
	syntax varlist(min=1 numeric) [if] , Tpanel(varlist numeric max=1)
						[
						BLOCKsize(integer 10)
						PREfix(string)
						Ipanel(varlist numeric max=1)
						Jpanel(varlist numeric max=1)
						Kpanel(varlist numeric max=1)
						]
						;
	#delimit cr

	* 2. Check for required characteristics

	* save starting order
	noisily display  "STARTED AT :"
	noisily display  c(current_time)
	quietly desc , varlist
	local varorder=r(varlist)
	local maxvar=r(k)
```

```
	tokenize `varlist'
	local allvars `*'
	local v=1

	capture {
		foreach var of varlist `allvars' {
			count if `var'==.
			if r(N)>0 {
			display as error "Missing values found in `var'. Missing values deleted casewise. "
			noisily drop if `var'==.
			}
		}
	}

	if "`prefix'"=="" {
	local prefix = "trans"
	}

	*tempvar tvar
	egen tvar=group(`tpanel')
	capture count if `tpanel'==.
	if r(N)>0 {
	display as error "Missing values in panel variable `tpanel'"
	exit
	}
	capture {
	sum tvar
	local tmax=r(max)
	}
	*tempvar ivar
	capture count if `ipanel'==.
	if r(N)>0 {
	display as error "Missing values in panel variable `ipanel'"
	exit
	}
	capture {
```

```
egen ivar=group(`ipanel')
sum ivar
local imax=r(max)
}
*tempvar jvar
capture count if `jpanel'==.
if r(N)>0 {
display as error "Missing values in panel variable `jpanel'"
exit
}
capture {
egen jvar=group(`jpanel')
sum jvar
local jmax=r(max)
}
*capture compare ivar jvar

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

tempfile master
save `master' , replace

keep ivar jvar tvar `allvars'
rename tvar `tpanel'
rename ivar `ipanel'
rename jvar `jpanel'
sort `tpanel' `ipanel' `jpanel'
putmata panelvars =(`tpanel' `ipanel' `jpanel' `var') , replace // changed statvars to panelvars!
putmata datavars =(`allvars'), replace

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

```
	mata: method(panelvars, datavars)
*****************


	tokenize `allvars'
	local testnr=1
	while "``testnr''"!="" {
		local maxvar2=`testnr'
		local testnr=`testnr'+1
	}

	capture noisily {
		foreach varnr of numlist 1/`maxvar2' {
			display "finalysing started for variable `varnr' "
			display c(current_time)
			local var="``varnr''"
			rename `prefix'_`varnr' `prefix'_`var'
			sum `var', meanonly
			local z=r(mean)*10^-10
			replace `prefix'_`var'=round(`prefix'_`var',`z')
			display "finalysing finished for variable `var' "
			display c(current_time)
		}
	}
	rename `tpanel' tvar
	rename `ipanel' ivar
	rename `jpanel' jvar
	merge 1:1 tvar ivar jvar using `master' , nogenerate noreport
	save `master', replace

	//varlist finish

	order `varorder'
	drop tvar ivar jvar
	noisily display  "FINISHED AT :"
	noisily display  c(current_time)
end
```

```
mata:
        mata clear

void function method(real matrix panelvars, real matrix datavars)
{
        imax=strtoreal(st_local("nmax"))
        tmax=strtoreal(st_local("tmax"))
        blocksize=strtoreal(st_local("blocksize"))
        bigt=rows(panelvars)

    /*1*/  rpgen_d1row(panelvars, imax, bigt, blocksize)
    /*2*/  rprowtocol("d1row", "d1col", bigt, imax*imax, blocksize)
    /*3*/  rpmultiply("d1col", "d1col", "d1'd1row", imax*imax, imax*imax,blocksize)
    /*4*/  rpinv("d1'd1row", "invd1'd1row", imax*imax,blocksize)
    /*5*/  rpunlink("d1'd1row",imax*imax,blocksize)
    /*6*/  rpmultiply("d1row","invd1'd1row","d1invd1'd1row",bigt, imax*imax,blocksize)
    /*7*/  rpunlink("invd1'd1row",imax*imax,blocksize)
    /*8*/  rpmultiply("d1invd1'd1row","d1row","d1invd1'd1d1'row",bigt, bigt,blocksize)
    /*9*/  rpunlink("d1row",bigt,blocksize)
    /*10*/ rpunlink("d1invd1'd1row",bigt,blocksize)
    /*11*/ rpunlink("d1col",imax*imax, blocksize)
    /*12*/ rpgen_b(bigt,blocksize)
    /*13*/ rpunlink("d1invd1'd1d1'row",bigt,blocksize)
    /*14*/ rpgen_d2row(panelvars, imax, bigt, tmax,blocksize)
    /*15*/ rprowtocol("d2row", "d2col", bigt, tmax*imax,blocksize)
    /*16*/ rpunlink("d2row", bigt,blocksize)
    /*17*/ rpmultiply("b_row","d2col","bd2row",bigt, tmax*imax,blocksize)
    /*18*/ rpunlink("d2col",tmax*imax,blocksize)
    /*19*/ rprowtocol("bd2row", "bd2col", bigt, tmax*imax,blocksize)
    /*20*/ rpmultiply("bd2col","bd2col","bd2'bd2row",tmax*imax, tmax*imax,blocksize)
    /*21*/ rpqinv("bd2'bd2row", "qinvbd2'bd2row", tmax*imax, 1000,blocksize)
    /*22*/ rpunlink("bd2'bd2row",tmax*imax,blocksize)
    /*23*/ rpmultiply("bd2row","qinvbd2'bd2row","bd2qinvbd2'bd2row",bigt,
tmax*imax,blocksize)
    /*24*/ rpunlink("qinvbd2'bd2row",tmax*imax,blocksize)
    /*25*/ rpmultiply("bd2qinvbd2'bd2row","bd2row","bd2qinvbd2'bd2bd2'row",bigt,
bigt,blocksize)
    /*26*/ rpunlink("bd2qinvbd2'bd2row",bigt,blocksize)
```

```
        /*27*/ rpunlink("bd2row", bigt,blocksize)
        /*28*/ rpunlink("bd2col",tmax*imax,blocksize)
        /*29*/ rpgen_c(bigt,blocksize)
        /*30*/ rpunlink("bd2qinvbd2'bd2bd2'row", bigt,blocksize)
        /*31*/ rpunlink("b_row",bigt,blocksize)
        /*32*/ rpgen_d3row(panelvars, bigt, tmax, imax,blocksize)
        /*33*/ rprowtocol("d3row", "d3col", bigt, tmax*imax,blocksize)
        /*34*/ rpunlink("d3row", bigt,blocksize)
        /*35*/ rpmultiply("c_row","d3col","cd3row",bigt, tmax*imax,blocksize)
        /*36*/ rpunlink("d3col",tmax*imax,blocksize)
        /*37*/ rprowtocol("cd3row", "cd3col", bigt, tmax*imax,blocksize)
        /*38*/ rpmultiply("cd3col","cd3col","cd3'cd3row",tmax*imax, tmax*imax,blocksize)
        /*39*/ rpunlink("cd3col",tmax*imax,blocksize)
        /*40*/ rpqinv("cd3'cd3row", "qinvcd3'cd3row", tmax*imax, 100,blocksize)
        /*41*/ rpunlink("cd3'cd3row",tmax*imax,blocksize)
        /*42*/
        rpmultiply("cd3row","qinvcd3'cd3row","cd3qinvcd3'cd3row",bigt,tmax*imax,blocksize)
        /*43*/ rpunlink("qinvcd3'cd3row", tmax*imax, blocksize)
        /*44*/ rpmultiply(
"cd3qinvcd3'cd3row","cd3row","cd3qinvcd3'cd3cd3'row",bigt,bigt,blocksize)
        /*45*/ rpunlink("cd3qinvcd3'cd3row",bigt,blocksize)
        /*46*/ rpgen_e(bigt,blocksize)
        /*47*/ rpunlink("cd3qinvcd3'cd3cd3'row",bigt,blocksize)
        /*48*/ rpunlink("cd3row",bigt,blocksize)
        /*49*/ rpaltexport(datavars,bigt,blocksize)
        /*50*/ rpunlink("e_row",bigt,blocksize)
        /*51*/ rpunlink("c_row",bigt,blocksize)
}
/***********************************/


void function rpcheckcontent (string scalar mattocheck, real scalar i )
{
        fh = fopen(st_macroexpand(mattocheck+strofreal(i)+".myfile") , "r" , 1 )
                        rprow=fgetmatrix(fh)
                        rpcol=fgetmatrix(fh)


        printf("CONTENT CHECKING START "+mattocheck+strofreal(i)+"\n" )
```

```
        printf("rprow \n")
        rprow
        printf("rpcol \n")
        rpcol
        fclose(fh)
        printf("CONTENT CHECKING END "+mattocheck+strofreal(i)+"\n" )
}
```

```
void function rprowtocol(string scalar oldmatname, string scalar newmatname, real scalar colsize, real scalar rowsize, real scalar blocksize)
{
                real matrix rprow
                real matrix rpcol
                printf("transforming " + oldmatname + "\n")
        readypercent=5
        for(j=1;j<=ceil(rowsize/blocksize);j++) {
                width=blocksize
                if(j==ceil(rowsize/blocksize)) {
                width=rowsize-(j-1)*blocksize
                }
                rpcol=J(colsize,width,.)
                for(i=1;i<=ceil(colsize/blocksize);i++) {
                        fh = fopen(st_macroexpand(oldmatname+strofreal(i)+".myfile") , "r" , 1 )
                        rprow=fgetmatrix(fh)
                        a1j=(j-1)*blocksize+1
                        b1j=j*blocksize
                        if(b1j>rowsize) {
                        b1j=rowsize
                        }
                        x=rprow[|1,a1j \ .,b1j|]
                        fclose(fh)
                        a2i=(i-1)*blocksize+1
                        b2i=i*blocksize
                        if(b2i>colsize) {
                        b2i=colsize
                        }
                        rpcol[| a2i,1 \ b2i ,. |]=x
```

```
                }
                unlink(st_macroexpand(newmatname+strofreal(j)+".myfile"))
                fh2 = fopen(st_macroexpand(newmatname+strofreal(j)+".myfile") , "w" , 1 )
                rprow=rpcol'
                fputmatrix(fh2, rprow)
                fputmatrix(fh2, rpcol)
                fclose(fh2)

                if(mod(j,round(0.05*ceil(rowsize/blocksize)))==0) {
                printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
                readypercent=readypercent+5
                }
        }
}
```

void function rpmultiply(string scalar rowname , string scalar colname, string scalar endrowname, real scalar newcolsize, real scalar newrowsize, real scalar blocksize)

```
{
                real rowvector rprow, useasrow, rpnewrow
                real colvector rpcol
                printf("multiplying "+ rowname+ " with "+ colname+ " \n")
        readypercent=5
        for(i=1;i<=ceil(newcolsize/blocksize);i++) {
                height=blocksize
                if(i==ceil(newcolsize/blocksize)) {
                        height=newcolsize-(i-1)*blocksize
                }
                rpnewrow=J(height,newrowsize,.)
                fh1 = fopen(st_macroexpand(rowname+strofreal(i)+".myfile") , "r" , 1 )
                rprow=fgetmatrix(fh1)
                fclose(fh1)
                useasrow=rprow
                for(j=1;j<=ceil(newrowsize/blocksize);j++) {
                        fh2 = fopen(st_macroexpand(colname+strofreal(j)+".myfile") , "r" , 1 )
                        rprow=fgetmatrix(fh2)
                        rpcol=fgetmatrix(fh2)
                        fclose(fh2)
                        a1i=(i-1)*blocksize+1
```

```
                a1j=(j-1)*blocksize+1
                b1i=i*blocksize
                if(b1i>newrowsize) {
                        b1i=newrowsize
                }
                b1j=j*blocksize
                if(b1j>newrowsize) {
                        b1j=newrowsize
                }
                rpnewrow[|1,a1j \ ., b1j |]=useasrow*rpcol
        }
        rprow=rpnewrow
        unlink(st_macroexpand(endrowname+strofreal(i)+".myfile"))
        fh3 = fopen(st_macroexpand(endrowname+strofreal(i)+".myfile") , "w" , 1 )
        fputmatrix(fh3, rprow)
        rpcol=rprow'
        fputmatrix(fh3, rpcol)
        fclose(fh3)
        /*****/
        if(mod(i,round(0.05*ceil(newcolsize/blocksize)))==0) {
        printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
        readypercent=readypercent+5
        }
    }
}
void function rpinv(string scalar oldmatname, string scalar newmatname, real scalar rownumber, real scalar blocksize) // N2*N2
{
        printf("inverting  " + oldmatname + " \n")
        real matrix mattoinv, invmat
        real rowvector rprow
        real colvector rpcol
    mattoinv=J(rownumber, rownumber,.)
    for(i=1;i<=ceil(rownumber/blocksize);i++) {
        fh1 = fopen(st_macroexpand(oldmatname+strofreal(i)+".myfile") , "r" , 1 )
        rprow=fgetmatrix(fh1)
        fclose(fh1)
```

```
            ai=(i-1)*blocksize+1
            bi=i*blocksize
            if(bi>rownumber){
                  bi=rownumber
            }
            mattoinv[ai::bi,.]=rprow
      }
      invmat=invsym(mattoinv)
      for(i=1;i<=ceil(rownumber/blocksize);i++) {
            ai=(i-1)*blocksize+1
            bi=i*blocksize
            if(bi>rownumber){
            bi=rownumber
            }
            rprow=invmat[ai::bi,.]
            unlink(st_macroexpand(newmatname+strofreal(i)+".myfile"))
            fh3 = fopen(st_macroexpand(newmatname+strofreal(i)+".myfile") , "w" , 1 )
            fputmatrix(fh3, rprow)
            rpcol=rprow'
            fputmatrix(fh3, rpcol)
            fclose(fh3)
      }
}
void function rpqinv(string scalar oldmatname, string scalar newmatname, real scalar
rownumber, real scalar rpedittozero, real scalar blocksize) // N2*N2
{
            printf(" inverting " + oldmatname + " \n")
            real matrix mattoinv, invmat
            real rowvector rprow
            real colvector rpcol
      mattoinv=J(rownumber, rownumber,.)
      for(i=1;i<=ceil(rownumber/blocksize);i++) {
            fh1 = fopen(st_macroexpand(oldmatname+strofreal(i)+".myfile") , "r" , 1 )
            rprow=fgetmatrix(fh1)
            fclose(fh1)
            ai=(i-1)*blocksize+1
            bi=i*blocksize
```

Any questions to: peterrevesz86@gmail.com

```
            if(bi>rownumber){
            bi=rownumber
            }
            mattoinv[ai::bi,.]=rprow
      }
      invmat=edittozero(qrinv(mattoinv),rpedittozero)
      for(i=1;i<=ceil(rownumber/blocksize);i++) {
            ai=(i-1)*blocksize+1
            bi=i*blocksize
            if(bi>rownumber){
                  bi=rownumber
            }
            rprow=invmat[ai::bi,.]
            unlink(st_macroexpand(newmatname+strofreal(i)+".myfile"))
            fh3 = fopen(st_macroexpand(newmatname+strofreal(i)+".myfile") , "w" , 1 )
            fputmatrix(fh3, rprow)
            rpcol=rprow'
            fputmatrix(fh3, rpcol)
            fclose(fh3)
      }
}
void function rpunlink(string scalar mattodel, real scalar rownumber, real scalar blocksize)
{
      printf(" unlinking " + mattodel +st_global("c(current_time)") +"\n")
      readypercent=5
      for(i=1;i<=ceil(rownumber/blocksize);i++) {
            unlink(st_macroexpand(mattodel+strofreal(i)+".myfile"))
            if(mod(i,round(0.05*ceil(rownumber/blocksize)))==0) {
                  printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                  readypercent=readypercent+5
            }
      }
}
void function rpgen_d1row(real matrix panelvars, real scalar imax ,real scalar bigt, real scalar
blocksize)
{
            real scalar helpi, helpj, orderids
```

```
        real rowvector rprow
        real colvector rpcol
    printf("generating D1 \n ")
    readypercent=5
    filecounter=1
    for(i=1;i<=bigt;i++) {
        if(mod(i,blocksize)==1) {
            height=blocksize
            if((i-1)+height>bigt) {
                height=bigt-i+1
            }
            rprow=J(height,imax*imax,.)
        }
        helpi=panelvars[i,2]
        helpj=panelvars[i,3]
        orderids=(helpi-1)*imax+helpj
        rprowtemp=e(orderids, imax*imax )
        rowposition=i-(filecounter-1)*blocksize
        if(mod(i,blocksize)>=1) {
            rprow[rowposition,.]=rprowtemp
        }
        if(mod(i,blocksize)==0) {
            rprow[rowposition,.]=rprowtemp
            unlink(st_macroexpand("d1row"+strofreal(filecounter)+".myfile"))
            fh = fopen(st_macroexpand("d1row"+strofreal(filecounter)+".myfile") ,
"w" , 1 )
            fputmatrix(fh, rprow)
            rpcol=rprow'
            fputmatrix(fh, rpcol)
            fclose(fh)
            filecounter=filecounter+1
        }
        if(i==bigt) {
            unlink(st_macroexpand("d1row"+strofreal(filecounter)+".myfile"))
            fh = fopen(st_macroexpand("d1row"+strofreal(filecounter)+".myfile") ,
"w" , 1 )
            fputmatrix(fh, rprow)
            rpcol=rprow'
```

```
                    fputmatrix(fh, rpcol)
                    fclose(fh)
            }
            if(mod(i,round(0.05*bigt))==0) {
            printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ " \n")
            readypercent=readypercent+5
            }
      }
}
void function rpgen_d2row(real matrix panelvars, real scalar imax ,real scalar bigt, real scalar tmax, real scalar blocksize)
{
            real scalar helpi, helpt, orderit
            real rowvector rprow
            real colvector rpcol
      printf("generating D2 \n ")
      readypercent=5
      filecounter=1
      for(i=1;i<=bigt;i++) {
            if(mod(i,blocksize)==1) {
                    height=blocksize
                    if((i-1)+height>bigt) {
                            height=bigt-i+1
                    }
                    rprow=J(height,tmax*imax,.)
            }
            helpt=panelvars[i,1]
            helpi=panelvars[i,2]
            orderit=(helpt-1)*imax+helpi
            rprowtemp=e(orderit, tmax*imax )
            rowposition=i-(filecounter-1)*blocksize
            if(mod(i,blocksize)>=1) {
                    rprow[rowposition,.]=rprowtemp
            }
            if(mod(i,blocksize)==0) {
                    rprow[rowposition,.]=rprowtemp
                    unlink(st_macroexpand("d2row"+strofreal(filecounter)+".myfile"))
```

```
                        fh = fopen(st_macroexpand("d2row"+strofreal(filecounter)+".myfile") ,
"w" , 1 )
                        fputmatrix(fh, rprow)
                        rpcol=rprow'
                        fputmatrix(fh, rpcol)
                        fclose(fh)
                        filecounter=filecounter+1
                }
                if(i==bigt) {
                        unlink(st_macroexpand("d2row"+strofreal(filecounter)+".myfile"))
                        fh = fopen(st_macroexpand("d2row"+strofreal(filecounter)+".myfile") ,
"w" , 1 )
                        fputmatrix(fh, rprow)
                        rpcol=rprow'
                        fputmatrix(fh, rpcol)
                        fclose(fh)
                }
                if(mod(i,round(0.05*bigt))==0) {
                        printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                        readypercent=readypercent+5
                }
        }
}
void function rpgen_d3row(real matrix panelvars, real scalar bigt , real scalar tmax, real scalar
imax, real scalar blocksize)
{
                printf("generating D3 \n")
        readypercent=5
        filecounter=1
        for(i=1;i<=bigt;i++) {
                if(mod(i,blocksize)==1) {
                        height=blocksize
                        if((i-1)+height>bigt) {
                                height=bigt-i+1
                        }
                        rprow=J(height,tmax*imax,.)
                }
```

```
		helpt=panelvars[i,1]
		helpj=panelvars[i,3]
		orderjt=(helpt-1)*imax+helpj
		rprowtemp=e(orderjt, tmax*imax )
		rowposition=i-(filecounter-1)*blocksize
		if(mod(i,blocksize)>=1) {
			rprow[rowposition,.]=rprowtemp
		}
		if(mod(i,blocksize)==0) {
			rprow[rowposition,.]=rprowtemp
			unlink(st_macroexpand("d3row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d3row"+strofreal(filecounter)+".myfile") ,
"w" , 1 )
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
			filecounter=filecounter+1
		}
		if(i==bigt) {
			unlink(st_macroexpand("d3row"+strofreal(filecounter)+".myfile"))
			fh = fopen(st_macroexpand("d3row"+strofreal(filecounter)+".myfile") ,
"w" , 1 )
			fputmatrix(fh, rprow)
			rpcol=rprow'
			fputmatrix(fh, rpcol)
			fclose(fh)
		}
		if(mod(i,round(0.05*bigt))==0) {
			printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
			readypercent=readypercent+5
		}
	}
}
void function rpgen_b(real scalar bigt, real scalar blocksize)
{
	printf("generating B \n")
```

```
	readypercent=5
	for(i=1;i<=ceil(bigt/blocksize);i++) {
		fh1 = fopen(st_macroexpand("d1invd1'd1d1'row"+strofreal(i)+".myfile") , "r" , 1
)
		rprow=fgetmatrix(fh1)
		fclose(fh1)
		height=blocksize
		if(i==ceil(bigt/blocksize)){
			height=bigt-(i-1)*blocksize
		}
		rpnewrow=J(height,bigt,.)
		for(k=1;k<=height;k++) {
			l=(i-1)*blocksize+k
			rpnewrow[k,.]=e(l,bigt)-rprow[k,.]
		}
		rprow=rpnewrow
		unlink(st_macroexpand("b_row"+strofreal(i)+".myfile"))
		fh3 = fopen(st_macroexpand("b_row"+strofreal(i)+".myfile") , "w" , 1 )
		fputmatrix(fh3, rprow)
		rpcol=rprow'
		fputmatrix(fh3, rpcol)
		fclose(fh3)
		if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
			printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
			readypercent=readypercent+5
		}
	}
}
void function rpgen_c(real scalar bigt, real scalar blocksize)
{
		printf("generating C \n")
	readypercent=5
	for(i=1;i<=ceil(bigt/blocksize);i++) {
		fh1 = fopen(st_macroexpand("bd2qinvbd2'bd2bd2'row"+strofreal(i)+".myfile") ,
"r" , 1 )
		rprow=fgetmatrix(fh1)
		userow1=rprow
```

```
                fclose(fh1)
                fh2 = fopen(st_macroexpand("b_row"+strofreal(i)+".myfile") , "r" , 1 )
                rprow=fgetmatrix(fh2)
                userow2=rprow
                fclose(fh2)
                rprow=userow2-userow1
                unlink(st_macroexpand("c_row"+strofreal(i)+".myfile"))
                fh3 = fopen(st_macroexpand("c_row"+strofreal(i)+".myfile") , "w" , 1 )
                fputmatrix(fh3, rprow)
                rpcol=rprow'
                fputmatrix(fh3, rpcol)
                fclose(fh3)
                if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
                        printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                readypercent=readypercent+5
                }
        }
}
void function rpgen_e(real scalar bigt, real scalar blocksize)
{
        printf("generating E \n")
        readypercent=5
        for(i=1;i<=ceil(bigt/blocksize);i++) {
                fh1 = fopen(st_macroexpand("cd3qinvcd3'cd3cd3'row"+strofreal(i)+".myfile") ,
"r" , 1 )
                rprow=fgetmatrix(fh1)
                userow1=rprow
                fclose(fh1)
                fh2 = fopen(st_macroexpand("c_row"+strofreal(i)+".myfile") , "r" , 1 )
                rprow=fgetmatrix(fh2)
                userow2=rprow
                fclose(fh2)
                rprow=userow2-userow1
                unlink(st_macroexpand("e_row"+strofreal(i)+".myfile"))
                fh3 = fopen(st_macroexpand("e_row"+strofreal(i)+".myfile") , "w" , 1 )
                fputmatrix(fh3, rprow)
                rpcol=rprow'
```

```
                fputmatrix(fh3, rpcol)
                fclose(fh3)
                if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
                        printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                        readypercent=readypercent+5
                }
        }
}
void function rpaltexport(real matrix datavars, real scalar bigt, real scalar blocksize)
{
        printf("Exporting started \n")
        readypercent=5
        maxvar=cols(datavars)
        mytostata=J(bigt, maxvar,.)
        for(i=1;i<=ceil(bigt/blocksize);i++) {
                fh1 = fopen(st_macroexpand("e_row"+strofreal(i)+".myfile") , "r" , 1 )
                rprow=fgetmatrix(fh1)
                useasrow=rprow
                fclose(fh1)
                ai=(i-1)*blocksize+1
                bi=i*blocksize
                if(bi>bigt){
                bi=bigt
                }
                mytostata[|ai,1\bi,.|]=useasrow*datavars
                if(mod(i,round(0.05*ceil(bigt/blocksize)))==0) {
                        printf(strofreal(readypercent)+"percent "+st_global("c(current_time)")+ "
\n")
                        readypercent=readypercent+5
                }
        }
        mytostata=edittozero(mytostata,1000)
        for(k=1;k<=maxvar;k++) {
                st_addvar("double", st_local("prefix")+"_"+strofreal(k))
                st_store(.,(st_local("prefix")+"_"+strofreal(k)),mytostata[.,k])
                printf("Exported the "+strofreal(k)+"th variable \n")
        }
```

```
	printf("Exporting finished \n ")
}
end
```