Implementing econometric estimators with Mata

Christopher F Baum and Mark E. Schaffer

Boston College and DIW Berlin / Heriot–Watt University

DC09 & UKSUG15, July-September 2009

In this talk we will describe the development of two econometric estimators using the capabilities of Mata, stressing how the combined Stata ado-file / Mata function environment is very well suited to performing such tasks.

The first estimator, an extension of Stata's `sureg` for linear seemingly unrelated regressions, might have been implemented in the same way if we had access to Stata 11.

The second, a nonlinear GMM (generalized method of moments) extension of the linear IV (instrumental variables) estimator, might be able to take advantage of some of Stata 11's new capabilities if we had written it with Stata 11 at hand. What we present today is based upon what is feasible in the Stata 10.1 environment, and is described at greater length in sections 14.8 and 14.9 of *An Introduction to Stata Programming* (ITSP), Stata Press, 2009.

In this talk we will describe the development of two econometric estimators using the capabilities of Mata, stressing how the combined Stata ado-file / Mata function environment is very well suited to performing such tasks.

The first estimator, an extension of Stata's `sureg` for linear seemingly unrelated regressions, might have been implemented in the same way if we had access to Stata 11.

The second, a nonlinear GMM (generalized method of moments) extension of the linear IV (instrumental variables) estimator, might be able to take advantage of some of Stata 11's new capabilities if we had written it with Stata 11 at hand. What we present today is based upon what is feasible in the Stata 10.1 environment, and is described at greater length in sections 14.8 and 14.9 of *An Introduction to Stata Programming* (ITSP), Stata Press, 2009.

In this talk we will describe the development of two econometric estimators using the capabilities of Mata, stressing how the combined Stata ado-file / Mata function environment is very well suited to performing such tasks.

The first estimator, an extension of Stata's `sureg` for linear seemingly unrelated regressions, might have been implemented in the same way if we had access to Stata 11.

The second, a nonlinear GMM (generalized method of moments) extension of the linear IV (instrumental variables) estimator, might be able to take advantage of some of Stata 11's new capabilities if we had written it with Stata 11 at hand. What we present today is based upon what is feasible in the Stata 10.1 environment, and is described at greater length in sections 14.8 and 14.9 of *An Introduction to Stata Programming* (ITSP), Stata Press, 2009.

Stata's seemingly unrelated regression (SUR) estimator (`sureg`) estimates a set of equations, employing the matrix of residual correlations for each equation to produce a refined estimate of its parameter vector.

The `sureg` estimator can be considered as a panel data estimator that operates in the *wide form*. It is common for panel data to be *unbalanced*, and Stata's `xt` commands handle unbalanced panels without difficulty. However, `sureg` discards any observation that is missing in *any* of its equations.

Stata's seemingly unrelated regression (SUR) estimator (`sureg`) estimates a set of equations, employing the matrix of residual correlations for each equation to produce a refined estimate of its parameter vector.

The `sureg` estimator can be considered as a panel data estimator that operates in the *wide form*. It is common for panel data to be *unbalanced*, and Stata's `xt` commands handle unbalanced panels without difficulty. However, `sureg` discards any observation that is missing in *any* of its equations.

We would like to use `sureg` without losing these observations. If we use `correlate` *varlist*, observations missing for *any* of the variables in the *varlist* will be dropped from the calculation. In contrast, the pairwise correlation command `pwcorr` *varlist* will compute correlations from all available observations for each pair of variables in turn.

The logic of `sureg` is that of `correlate`; but the correlations of residuals employed by `sureg` may be, as in `pwcorr`, calculated on a pairwise basis, allowing the estimator to be applied to a set of equations which may cover different time periods (as long as there is meaningful overlap).

I illustrate how Mata may be used to handle this quite sophisticated estimation problem. I worked with the code of official Stata's `reg3` to extract the parsing commands that set up the problem, and then wrote my own ado-file to perform the computations. It calls a Mata function that produces the estimates.

We would like to use `sureg` without losing these observations. If we use `correlate` *varlist*, observations missing for *any* of the variables in the *varlist* will be dropped from the calculation. In contrast, the pairwise correlation command `pwcorr` *varlist* will compute correlations from all available observations for each pair of variables in turn.

The logic of `sureg` is that of `correlate`; but the correlations of residuals employed by `sureg` may be, as in `pwcorr`, calculated on a pairwise basis, allowing the estimator to be applied to a set of equations which may cover different time periods (as long as there is meaningful overlap).

I illustrate how Mata may be used to handle this quite sophisticated estimation problem. I worked with the code of official Stata's `reg3` to extract the parsing commands that set up the problem, and then wrote my own ado-file to perform the computations. It calls a Mata function that produces the estimates.

We would like to use `sureg` without losing these observations. If we use `correlate` *varlist*, observations missing for *any* of the variables in the *varlist* will be dropped from the calculation. In contrast, the pairwise correlation command `pwcorr` *varlist* will compute correlations from all available observations for each pair of variables in turn.

The logic of `sureg` is that of `correlate`; but the correlations of residuals employed by `sureg` may be, as in `pwcorr`, calculated on a pairwise basis, allowing the estimator to be applied to a set of equations which may cover different time periods (as long as there is meaningful overlap).

I illustrate how Mata may be used to handle this quite sophisticated estimation problem. I worked with the code of official Stata's `reg3` to extract the parsing commands that set up the problem, and then wrote my own ado-file to perform the computations. It calls a Mata function that produces the estimates.

For brevity, I do not reproduce the parsing code taken from `reg3.ado`. The syntax of my program, `suregub`, is identical to that of `sureg` but does not support all of its options. The basic syntax is:

```
suregub (depvar1 varlist1) (depvar2 varlist2) ... (depvarN varlistN)
```

The parsing code defines a local macro `eqlist` of equations to be estimated and a set of local macros `ind1 ...indN` which contains the right-hand-side variables for each of the *N* equations.

We first generate the OLS residuals for each equation by running `regress` and `predict, residual`. We find the maximum and minimum observation indices for each set of residuals with the `max()` and `min()` functions, respectively.

For brevity, I do not reproduce the parsing code taken from `reg3.ado`. The syntax of my program, `suregub`, is identical to that of `sureg` but does not support all of its options. The basic syntax is:

```
suregub (depvar1 varlist1) (depvar2 varlist2) ... (depvarN varlistN)
```

The parsing code defines a local macro `eqlist` of equations to be estimated and a set of local macros `ind1 ...indN` which contains the right-hand-side variables for each of the *N* equations.

We first generate the OLS residuals for each equation by running `regress` and `predict, residual`. We find the maximum and minimum observation indices for each set of residuals with the `max()` and `min()` functions, respectively.

For brevity, I do not reproduce the parsing code taken from `reg3.ado`. The syntax of my program, `suregub`, is identical to that of `sureg` but does not support all of its options. The basic syntax is:

```
suregub (depvar1 varlist1) (depvar2 varlist2) ... (depvarN varlistN)
```

The parsing code defines a local macro `eqlist` of equations to be estimated and a set of local macros `ind1 ...indN` which contains the right-hand-side variables for each of the N equations.

We first generate the OLS residuals for each equation by running `regress` and `predict, residual`. We find the maximum and minimum observation indices for each set of residuals with the `max()` and `min()` functions, respectively.

```
// generate residual series
local minn = .
local maxn = 0
forvalues i = 1/`neq´ {
        local dv : word `i´ of `eqlist´
        local eq`i´ = "`dv´ `ind`i´´"
        qui {
        regress `dv´ `ind`i´´
        tempvar touse`i´ es eps`i´
        predict double `eps`i´´ if e(sample), resid
        generate byte `touse`i´´ = cond(e(sample), 1, .)
        summarize `eps`i´´, meanonly
        local maxn = max(`maxn´, r(N))
        local minn = min(`minn´, r(N))
        }
}
```

Temporary matrix `sigma` contains the pairwise correlations of the equations' residuals. They are produced with calls to `correlate`. We then invoke the Mata function `mm_suregub`, passing it the number of equations (`neq`), the list of equations (`eqlist`) and the computed `sigma` matrix.

```
tempname sigma
matrix `sigma´ = J(`neq´, `neq´, 0)
// generate pairwise correlation matrix of residuals;
// for comparison with sureg, use divisor N
local neq1 = `neq´ - 1
forvalues i = 1/`neq1´  {
        forvalues j = 2/`neq´   {
                qui correlate `eps`i´´ `eps`j´´, cov
                mat `sigma´[`i´, `i´] = r(Var_1) * (r(N) - 1) / (r(N))
                mat `sigma´[`j´, `j´] = r(Var_2) * (r(N) - 1) / (r(N))
                mat `sigma´[`i´, `j´] = r(cov_12) * (r(N) - 1) / (r(N))
                mat `sigma´[`j´, `i´] = `sigma´[`i´, `j´]
        }
}
mata: mm_suregub(`neq´, "`eqlist´", "`sigma´")
```

The last block of code displays the header, posts the coefficient vector and VCE to the ereturn structure and uses `ereturn display` to produce the standard estimation output.

```
display _newline "Seemingly unrelated regression for an unbalanced panel"
display _newline "Minimum observations per unit = `minn´"
display    "Maximum observations per unit = `maxn´"
mat b = r(b)
mat V = r(V)
ereturn clear
ereturn post b V
ereturn local cmd "suregub"
ereturn local minobs `minn´
ereturn local maxobs `maxn´
ereturn display
end
```

The “heavy lifting” in this command is performed within the Mata routine. Its logic requires a number of matrices, but we cannot know how many matrices are needed until the routine is invoked. Thus, we make use of Mata’s *pointers* (in this case, rowvectors `eq, xx, yy` of pointers to real matrices). The use of pointers is described in ITSP, Section 13.8.

We also must work with the coefficient names and *matrix stripes* attached to Stata matrices so that the display of estimation results will work properly.

The “heavy lifting” in this command is performed within the Mata routine. Its logic requires a number of matrices, but we cannot know how many matrices are needed until the routine is invoked. Thus, we make use of Mata’s *pointers* (in this case, rowvectors `eq, xx, yy` of pointers to real matrices). The use of pointers is described in ITSP, Section 13.8.

We also must work with the coefficient names and *matrix stripes* attached to Stata matrices so that the display of estimation results will work properly.

```
version 10.1
mata: mata clear
mata: mata set matastrict on
mata:
// mm_suregub 1.0.0  CFBaum 11aug2008
void mm_suregub(real scalar neq,
                string scalar eqlist,
                string scalar ssigma)
{
        real matrix isigma, tt, eqq, iota, XX, YY, xi, xj, yj, vee
        real vector beta
        real scalar nrow, ncol, i, ii, i2, jj, j, j2
        string scalar lt, touse, le, eqname, eqv
        string vector v, vars, stripe
        pointer (real matrix) rowvector eq
        pointer (real matrix) rowvector xx
        pointer (real matrix) rowvector yy

        eq = xx = yy = J(1, neq, NULL)
        isigma = invsym(st_matrix(ssigma))
        nrow = 0
        ncol = 0
        string rowvector coefname, eqn
        string matrix mstripe
```

The first section of the function loops over equations, setting up the appropriate contents of the dependent variable (`yy[i]`) and the right-hand-side variables (`xx[i]`) for each equation in turn. A constant term is assumed to be present in each equation.

```
// equation loop 1
        for(i = 1; i <= neq; i++) {
                lt = "touse" + strofreal(i)
                touse = st_local(lt)
                st_view(tt, ., touse)
                le = "eq" + strofreal(i)
                eqv = st_local(le)
                vars = tokens(eqv)
                v = vars[|1, .|]
// pull in full matrix, including missing values
                st_view(eqq, ., v)
                eq[i] = &(tt :* eqq)
// matrix eq[i] is [y|X] for ith eqn
                eqname = v[1]
                stripe = v[2::cols(v)], "_cons"
                coefname = coefname, stripe
                eqn = eqn, J(1, cols(v), eqname)
// form X, assuming constant term
                nrow = nrow + rows(*eq[i])
                iota = J(rows(*eq[i]), 1, 1)
                xx[i] = &((*eq[i])[| 1,2 \ .,. |], iota)
                ncol = ncol + cols(*xx[i])
// form y
                yy[i] = &(*eq[i])[.,1]
        }
```

In the second loop over equations, the elements of the full $\mathbf{X}'\mathbf{X}$ matrix are computed as scalar multiples of an element of the inverse of `sigma` times the cross-product of the i^{th} and j^{th} equations' regressor matrices. The full $\mathbf{y}'\mathbf{y}$ vector is built up from scalar multiples of an element of the inverse of `sigma` times the cross-product of the i^{th} equation's regressors and the j^{th} equation's values of `yy`.

```
        XX = J(ncol, ncol, 0)
        YY = J(ncol, 1, 0)
        ii = 0
// equation loop 2
        for(i=1; i<=neq; i++) {
        i2 = cols(*xx[i])
        xi = *xx[i]
        jj = 0
        for(j=1; j<=neq; j++) {
                xj = *xx[j]
                j2 = cols(*xx[j])
                yj = *yy[j]
                XX[| ii+1, jj+1 \ ii+i2, jj+j2 |] = isigma[i, j] :* cross(xi, xj)
                YY[| ii+1, 1 \ ii+i2, 1 |] = YY[| ii+1, 1 \ ii+i2, 1 |] + ///
                                                 isigma[i, j] :* cross(xi, yj)
                jj = jj + j2
        }
        ii = ii + i2
        }
```

The least squares solution is obtained with `invsym()` and the appropriate matrix row and column stripes are defined for the result matrices `r(b)` and `r(V)`.

```
// compute SUR beta (X´ [Sigma^-1 # I] X)^-1 (X´ [Sigma^-1 # I] y)
        vee = invsym(XX)
        beta = vee * YY
        st_matrix("r(b)", beta´)
        mstripe=eqn´, coefname´
        st_matrixcolstripe("r(b)", mstripe)
        st_matrix("r(V)", vee)
        st_matrixrowstripe("r(V)", mstripe)
        st_matrixcolstripe("r(V)", mstripe)
}
end
```

To validate the routine, we first apply it to a balanced panel, for which it should replicate standard `sureg` results if it has been programmed properly. As we have verified that `suregub` passes that test using a version of the `grunfeld.dta` dataset that has been reshaped to the wide format, we modify that dataset to create an unbalanced panel and use `suregub` to estimate four companies' investment equations.

```
webuse grunfeld, clear
drop in 75/80
drop in 41/43
drop in 18/20
keep if company <= 4
drop time
reshape wide invest mvalue kstock, i(year) j(company)
```

```
. suregub (invest1 mvalue1 kstock1) (invest2 mvalue2 kstock2) ///
>         (invest3 mvalue3 kstock3) (invest4 mvalue4 kstock4)
Seemingly unrelated regressions for an unbalanced panel
Min obs per unit = 14
Max obs per unit = 20
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
invest1      |
     mvalue1 |   .0787979   .0220396     3.58   0.000      .035601    .1219948
     kstock1 |    .245538    .044413     5.53   0.000     .1584901    .3325858
       _cons |   64.69007   96.85787     0.67   0.504    -125.1479     254.528
-------------+----------------------------------------------------------------
invest2      |
     mvalue2 |   .1729584   .0640754     2.70   0.007      .047373    .2985439
     kstock2 |   .4150819   .1279443     3.24   0.001     .1643156    .6658482
       _cons |   -53.8353   128.2932    -0.42   0.675    -305.2853    197.6147
-------------+----------------------------------------------------------------
invest3      |
     mvalue3 |   .0522683   .0191105     2.74   0.006     .0148124    .0897243
     kstock3 |   .1071995   .0287962     3.72   0.000     .0507599     .163639
       _cons |  -39.32897   37.37061    -1.05   0.293     -112.574    33.91607
-------------+----------------------------------------------------------------
invest4      |
     mvalue4 |   .0632339   .0142336     4.44   0.000     .0353364    .0911313
     kstock4 |   .1487322   .0825525     1.80   0.072    -.0130678    .3105321
       _cons |   12.50393   11.02009     1.13   0.257     -9.09504     34.1029
------------------------------------------------------------------------------
```

As the results from this user-written routine have been stored in the ereturn array, the normal post-estimation features are available:

```
. test [invest2]mvalue2 = [invest4]mvalue4

 ( 1)  [invest2]mvalue2 - [invest4]mvalue4 = 0

           chi2(  1) =    2.89
         Prob > chi2 =    0.0892
```

A number of additional features could be added to `suregub` to more closely match the behavior of `sureg`.

As the results from this user-written routine have been stored in the ereturn array, the normal post-estimation features are available:

```
. test [invest2]mvalue2 = [invest4]mvalue4

 ( 1)  [invest2]mvalue2 - [invest4]mvalue4 = 0

           chi2(  1) =    2.89
         Prob > chi2 =    0.0892
```

A number of additional features could be added to `suregub` to more closely match the behavior of `sureg`.

We turn now to our second estimator: the continuously-updated generalized method of moments estimator (GMM-CUE) of Hansen, Heaton, Yaron (1996). This estimator is described in Baum, Schaffer, Stillman, *Stata Journal* 7:4, 2007. This is an estimator of a linear instrumental variables model that requires numerical optimization for its solution.

We have implemented this estimator for `ivreg2` in Stata's ado-file language using the maximum likelihood commands of the `ml` suite. Although that is a workable solution, it can be very slow for large datasets with many regressors and instruments. In Stata versions 10 and 11, a full-featured suite of optimization commands are available in Mata as `optimize()`. We implement a simple IV-GMM estimator in Mata and use that as a model for implementing GMM-CUE.

We turn now to our second estimator: the continuously-updated generalized method of moments estimator (GMM-CUE) of Hansen, Heaton, Yaron (1996). This estimator is described in Baum, Schaffer, Stillman, *Stata Journal* 7:4, 2007. This is an estimator of a linear instrumental variables model that requires numerical optimization for its solution.

We have implemented this estimator for `ivreg2` in Stata's ado-file language using the maximum likelihood commands of the `ml` suite. Although that is a workable solution, it can be very slow for large datasets with many regressors and instruments. In Stata versions 10 and 11, a full-featured suite of optimization commands are available in Mata as `optimize()`. We implement a simple IV-GMM estimator in Mata and use that as a model for implementing GMM-CUE.

The following ado-file, `mygmm2s.ado`, accepts a dependent variable and three additional optional variable lists: for endogenous variables, included instruments and excluded instruments. A constant is automatically included in the regression and in the instrument matrix.

There is a single option, `robust`, which specifies whether we are assuming *i.i.d.* errors or allowing for arbitrary heteroskedasticity. The routine calls Mata function `m_mygmm2s()` and receives results back in the return list. Estimation results are assembled and `post`ed to the official locations so that we may make use of Stata's `ereturn display` command and enable the use of post-estimation commands such as `test` and `lincom`.

The following ado-file, `mygmm2s.ado`, accepts a dependent variable and three additional optional variable lists: for endogenous variables, included instruments and excluded instruments. A constant is automatically included in the regression and in the instrument matrix.

There is a single option, `robust`, which specifies whether we are assuming *i.i.d.* errors or allowing for arbitrary heteroskedasticity. The routine calls Mata function `m_mygmm2s()` and receives results back in the return list. Estimation results are assembled and `post`ed to the official locations so that we may make use of Stata's `ereturn display` command and enable the use of post-estimation commands such as `test` and `lincom`.

```
*! mygmm2s 1.0.2 MES/CFB 11aug2008
program mygmm2s, eclass
        version 10.1
/*
  Our standard syntax:
  mygmm2s y, endog(varlist1) inexog(varlist2) exexog(varlist3)  [robust]
  where varlist1 contains endogenous regressors
        varlist2 contains exogenous regressors (included instruments)
        varlist3 contains excluded instruments
  Without robust, efficient GMM is IV. With robust, efficient GMM is 2-step
  efficient GMM, robust to arbitrary heteroskedasticity.
  To accommodate time-series operators in the options, add the "ts"
*/
        syntax varname(ts) [if] [in] [, endog(varlist ts) inexog(varlist ts) ///
                 exexog(varlist ts) robust ]
        local depvar `varlist´
```

```
/*
   marksample handles the variables in `varlist´ automatically, but not the
   variables listed in the options `endog´, `inexog´ and so on. -markout- sets
   `touse´ to 0 for any observations where the variables listed are missing.
*/
        marksample touse
        markout `touse´ `endog´ `inexog´ `exexog´
// These are the local macros that our Stata program will use
        tempname b V omega
// Call the Mata routine. All results will be waiting for us in "r()" macros.
        mata: m_mygmm2s("`depvar´", "`endog´", "`inexog´", ///
                        "`exexog´", "`touse´", "`robust´")
```

```
// Move the basic results from r() macros into Stata matrices.
       mat `b´ = r(beta)
       mat `V´ = r(V)
       mat `omega´ = r(omega)
// Prepare row/col names.
// Our convention is that regressors are [endog   included exog]
// and instruments are                   [excluded exog  included exog]
// Constant is added by default and is the last column.
       local vnames `endog´ `inexog´ _cons
       matrix rownames `V´ = `vnames´
       matrix colnames `V´ = `vnames´
       matrix colnames `b´ = `vnames´
       local vnames2 `exexog´ `inexog´ _cons
       matrix rownames `omega´ = `vnames2´
       matrix colnames `omega´ = `vnames2´
```

```
// We need the number of observations before we post our results.
        local N = r(N)
        ereturn post `b´ `V´, depname(`depvar´) obs(`N´) esample(`touse´)
// Store remaining estimation results as e() macros accessible to the user.
        ereturn matrix omega `omega´
        ereturn local depvar = "`depvar´"
        ereturn scalar N = r(N)
        ereturn scalar j = r(j)
        ereturn scalar L = r(L)
        ereturn scalar K = r(K)
        if "`robust´" != "" {
            ereturn local vcetype "Robust"
        }
        display _newline "Two-step GMM results" _col(60) "Number of obs = " e(N)
        ereturn display
        display "Sargan-Hansen J statistic: " %7.3f e(j)
        display "Chi-sq(" %3.0f e(L)-e(K) "  )       P-val = " ///
                %5.4f chiprob(e(L)-e(K), e(j)) _newline
end
```

The Mata function receives the names of variables to be included in the regression and creates view matrices for `Y` (the dependent variable), `X1` (the endogenous variables), `X2` (the exogenous regressors or included instruments) and `Z1` (the excluded instruments). The `st_tsrevar()` function is used to deal with Stata's time series operators in any of the variable lists.

```
mata:mata clear
version 10.1
mata: mata set matastrict on
mata:
// m_mygmm2s 1.0.0 MES/CFB 11aug2008
void m_mygmm2s(string scalar yname,
               string scalar endognames,
               string scalar inexognames,
               string scalar exexognames,
               string scalar touse,
               string scalar robust)
{
        real matrix Y, X1, X2, Z1, X, Z, QZZ, QZX, W, omega, V
        real vector cons, beta_iv, beta_gmm, e, gbar
        real scalar K, L, N, j
```

```
// Use st_tsrevar in case any variables use Stata´s time-series operators.
        st_view(Y, ., st_tsrevar(tokens(yname)), touse)
        st_view(X1, ., st_tsrevar(tokens(endognames)), touse)
        st_view(X2, ., st_tsrevar(tokens(inexognames)), touse)
        st_view(Z1, ., st_tsrevar(tokens(exexognames)), touse)
// Our convention is that regressors are [endog   included exog]
// and instruments are                   [excluded exog  included exog]
// Constant is added by default and is the last column.
        cons = J(rows(X2), 1, 1)
        X2 = X2, cons
        X = X1, X2
        Z = Z1, X2
        K = cols(X)
        L = cols(Z)
        N = rows(Y)
        QZZ = 1/N * quadcross(Z, Z)
        QZX = 1/N * quadcross(Z, X)
```

```
// First step of 2-step feasible efficient GMM: IV (2SLS).  Weighting matrix
// is inv of Z´Z (or QZZ).
        W = invsym(QZZ)
        beta_iv = (invsym(X´Z * W * Z´X) * X´Z * W * Z´Y)
// By convention, Stata parameter vectors are row vectors
        beta_iv = beta_iv´
// Use first-step residuals to calculate optimal weighting matrix for 2-step FE GMM
        omega = m_myomega(beta_iv, Y, X, Z, robust)
// Second step of 2-step feasible efficient GMM: IV (2SLS).  Weighting matrix
// is inv of Z´Z (or QZZ).
        W = invsym(omega)
        beta_gmm = (invsym(X´Z * W * Z´X) * X´Z * W * Z´Y)
// By convention, Stata parameter vectors are row vectors
        beta_gmm = beta_gmm´
```

```
// Sargan-Hansen J statistic: first we calculate the second-step residuals
       e = Y - X * beta_gmm´
// Calculate gbar = 1/N * Z´*e
       gbar = 1/N * quadcross(Z, e)
       j = N * gbar´ * W * gbar
// Sandwich var-cov matrix (no finite-sample correction)
// Reduces to classical var-cov matrix if Omega is not robust form.
// But the GMM estimator is "root-N consistent", and technically we do
// inference on sqrt(N)*beta.  By convention we work with beta, so we adjust
// the var-cov matrix instead:
       V = 1/N * invsym(QZX´ * W * QZX)
// Easiest way of returning results to Stata: as r-class macros.
       st_matrix("r(beta)", beta_gmm)
       st_matrix("r(V)", V)
       st_matrix("r(omega)", omega)
       st_numscalar("r(j)", j)
       st_numscalar("r(N)", N)
       st_numscalar("r(L)", L)
       st_numscalar("r(K)", K)
}
end
```

This function in turn calls an additional Mata function, `m_myomega()`, to compute the appropriate covariance matrix. This is a `real matrix` function, as it will return its result, the real matrix `omega`, to the calling function.

Because we will reuse the `m_myomega()` function in our GMM-CUE program, we place it in a separate file, `m_myomega.mata`, with instructions to compile it into a `.mo` file.

This function in turn calls an additional Mata function, `m_myomega()`, to compute the appropriate covariance matrix. This is a `real matrix` function, as it will return its result, the real matrix `omega`, to the calling function.

Because we will reuse the `m_myomega()` function in our GMM-CUE program, we place it in a separate file, `m_myomega.mata`, with instructions to compile it into a `.mo` file.

```
mata: mata clear
version 10.1
mata: mata set matastrict on
mata:
//  m_myomega 1.0.0 MES/CFB 11aug2008
real matrix m_myomega(real rowvector beta,
                      real colvector Y,
                      real matrix X,
                      real matrix Z,
                      string scalar robust)
{
        real matrix QZZ, omega
        real vector e, e2
        real scalar N, sigma2

// Calculate residuals from the coefficient estimates
                N = rows(Z)
                e = Y - X * beta´
                if (robust=="") {
// Compute classical, non-robust covariance matrix
                    QZZ = 1/N * quadcross(Z, Z)
                    sigma2 = 1/N * quadcross(e, e)
                    omega = sigma2 * QZZ
                }
                else {
// Compute heteroskedasticity-consistent covariance matrix
                    e2 = e:^2
                    omega = 1/N * quadcross(Z, e2, Z)
                }
                _makesymmetric(omega)
                return (omega)
}
end
mata: mata mosave m_myomega(), dir(PERSONAL) replace
```

This gives us a working Mata implementation of an instrumental variables estimator and an IV-GMM estimator (accounting for arbitrary heteroskedasticity), and we can verify that its results match those of `ivregress` or our own `ivreg2`. To implement the GMM-CUE estimator, we clone `mygmm2s.ado` to `mygmmcue.ado`. The ado-file code is very similar:

```
*! mygmmcue 1.0.2 MES/CFB 11aug2008
program mygmmcue, eclass
        version 10.1
        syntax varname(ts) [if] [in] [ , endog(varlist ts) ///
                inexog(varlist ts) exexog(varlist ts) robust ]
        local depvar `varlist´
        marksample touse
        markout `touse´ `endog´ `inexog´ `exexog´
        tempname b V omega
        mata: m_mygmmcue("`depvar´", "`endog´", "`inexog´", ///
                         "`exexog´", "`touse´", "`robust´")
        mat `b´ = r(beta)
        mat `V´ = r(V)
        mat `omega´=r(omega)
        local vnames `endog´ `inexog´ _cons
        matrix rownames `V´ = `vnames´
        matrix colnames `V´ = `vnames´
        matrix colnames `b´ = `vnames´
        local vnames2 `exexog´ `inexog´ _cons
        matrix rownames `omega´ = `vnames2´
        matrix colnames `omega´ = `vnames2´
        local N = r(N)
        ereturn post `b´ `V´, depname(`depvar´) obs(`N´) esample(`touse´)
        ereturn matrix omega `omega´
        ereturn local depvar = "`depvar´"
        ereturn scalar N = r(N)
        ereturn scalar j = r(j)
        ereturn scalar L = r(L)
        ereturn scalar K = r(K)
        if "`robust´" != "" ereturn local vcetype "Robust"
        display _newline "GMM-CUE estimates" _col(60) "Number of obs = " e(N)
        ereturn display
        display "Sargan-Hansen J statistic: " %7.3f e(j)
        display "Chi-sq(" %3.0f e(L)-e(K) "  )        P-val = " ///
                %5.4f chiprob(e(L)-e(K), e(j)) _newline
end
```

We now consider how the Mata function must be modified to incorporate the numerical optimization routines. We must first make use of Mata's `external` declaration to specify that the elements needed within our objective function evaluator are visible to that routine. We could also pass those arguments to the evaluation routine, but treating them as `external` requires less housekeeping. As in the standard two-step GMM routine, we derive first-step estimates of the regression parameters from a standard GMM estimation.

We then use Mata's `optimize()` functions to set up the optimization problem. The `optimize_init()` call, as described in Gould, *Stata Journal* 7:4, sets up a Mata *structure* named `S` containing all elements of the problem. In a call to `optimize_init_evaluator()`, we specify that the evaluation routine is a Mata function, `m_mycuecrit()`, by providing a *pointer* to the function (see ITSP, Section 13.8).

```
mata:
//  m_mygmmcue 1.0.0 MES/CFB 11aug2008
void m_mygmmcue(string scalar yname,
                string scalar endognames,
                string scalar inexognames,
                string scalar exexognames,
                string scalar touse,
                string scalar robust)
{
        real matrix X1, X2, Z1, QZZ, QZX, W, V
        real vector cons, beta_iv, beta_cue
        real scalar K, L, N, S, j

// In order for the optimization objective function to find various variables
// and data they have to be set as externals.  This means subroutines can
// find them without having to have them passed to the subroutines as arguments
> .
// robustflag is the robust argument recreated as an external Mata scalar.
        external Y, X, Z, e, omega, robustflag
        robustflag = robust
```

```
       st_view(Y, ., st_tsrevar(tokens(yname)), touse)
       st_view(X1, ., st_tsrevar(tokens(endognames)), touse)
       st_view(X2, ., st_tsrevar(tokens(inexognames)), touse)
       st_view(Z1, ., st_tsrevar(tokens(exexognames)), touse)
// Our convention is that regressors are [endog          included exog]
// and instruments are                   [excluded exog  included exog]
// The constant is added by default and is the last column.
       cons = J(rows(X2), 1, 1)
       X2 = X2, cons
       X = X1, X2
       Z = Z1, X2
       K = cols(X)
       L = cols(Z)
       N = rows(Y)
       QZZ = 1/N * quadcross(Z, Z)
       QZX = 1/N * quadcross(Z, X)
```

```
// First step of CUE GMM: IV (2SLS).  Use beta_iv as the initial values for
// the numerical optimization.
       W = invsym(QZZ)
       beta_iv = invsym(X´Z * W *Z´X) * X´Z * W * Z´Y
// Stata convention is that parameter vectors are row vectors, and optimizers
// require this, so must conform to this in what follows.
       beta_iv = beta_iv´
// What follows is how to set out an optimization in Stata.  First, initialize
// the optimization structure in the variable S.  Then tell Mata where the
// objective function is, that it´s a minimization, that it´s a "d0" type of
// objective function (no analytical derivatives or Hessians), and that the
// initial values for the parameter vector are in beta_iv.  Finally, optimize.
       S = optimize_init()
       optimize_init_evaluator(S, &m_mycuecrit())
```

We call `optimize_init_which()` to indicate that we are minimizing (rather than maximizing) the objective function, and use `optimize_init_evaluatortype()` that our evaluation routine is a type `d0` evaluator. Finally, we call `optimize_init_params()` to provide starting values for the parameters from the instrumental variables coefficient vector `beta_iv`.

The `optimize()` function invokes the optimization routine, returning its results in the parameter rowvector `beta_cue`. The optimal value of the objective function is retrieved with `optimize_result_value()`.

We call `optimize_init_which()` to indicate that we are minimizing (rather than maximizing) the objective function, and use `optimize_init_evaluatortype()` that our evaluation routine is a type `d0` evaluator. Finally, we call `optimize_init_params()` to provide starting values for the parameters from the instrumental variables coefficient vector `beta_iv`.

The `optimize()` function invokes the optimization routine, returning its results in the parameter rowvector `beta_cue`. The optimal value of the objective function is retrieved with `optimize_result_value()`.

```
        optimize_init_which(S, "min")
        optimize_init_evaluatortype(S, "d0")
        optimize_init_params(S, beta_iv)
        beta_cue = optimize(S)
// The last omega is the CUE omega, and the last evaluation of the GMM
// objective function is J.
        W = invsym(omega)
        j = optimize_result_value(S)
        V = 1/N * invsym(QZX´ * W * QZX)
        st_matrix("r(beta)", beta_cue)
        st_matrix("r(V)", V)
        st_matrix("r(omega)", omega)
        st_numscalar("r(j)", j)
        st_numscalar("r(N)", N)
        st_numscalar("r(L)", L)
        st_numscalar("r(K)", K)
}
end
mata: mata mosave m_mygmmcue(), dir(PERSONAL) replace
```

Let us now examine the evaluation routine. Given values for the parameter vector `beta`, it computes a new value of the `omega` matrix (Ω, the covariance matrix of orthogonality conditions) and a new set of residuals `e`, which are also a function of `beta`.

The `j` statistic, which is the minimized value of the objective function, is then computed, depending on the updated residuals `e` and the weighting matrix `W = ` Ω^{-1}, a function of the updated estimates of `beta`.

Let us now examine the evaluation routine. Given values for the parameter vector `beta`, it computes a new value of the `omega` matrix (Ω, the covariance matrix of orthogonality conditions) and a new set of residuals `e`, which are also a function of `beta`.

The `j` statistic, which is the minimized value of the objective function, is then computed, depending on the updated residuals `e` and the weighting matrix `W` $= \Omega^{-1}$, a function of the updated estimates of `beta`.

```
mata:
// GMM-CUE evaluator function.
// Handles only d0-type optimization; todo, g and H are just ignored.
// beta is the parameter set over which we optimize, and
// j is the objective function to minimize.
// m_mycuecrit 1.0.0 MES/CFB 11aug2008
void m_mycuecrit(todo, beta, j, g, H)
{
        external Y, X, Z, e, omega, robustflag
        real matrix W
        real vector gbar
        real scalar N
        omega = m_myomega(beta, Y, X, Z, robustflag)
        W = invsym(omega)
        N = rows(Z)
        e = Y - X * beta´
// Calculate gbar=Z´*e/N
        gbar = 1/N * quadcross(Z,e)
        j = N * gbar´ * W * gbar
}
end
mata: mata mosave m_mycuecrit(), dir(PERSONAL) replace
```

Our Mata-based GMM-CUE routine is now complete. To validate both the two-step GMM routine and its GMM-CUE counterpart, we write a simple certification script for each routine. First, let's check to see that our two-step routine works for both *i.i.d.* and heteroskedastic errors. Our `mygmm2s` routine returns the same results as `ivreg2` from `SSC` for the several objects included in the `savedresults compare` validation command, and these functions of `ivreg2` have been certified against official Stata's `ivregress`.

Now we construct and run a similar certification script to validate the GMM-CUE routine:

Our Mata-based GMM-CUE routine is now complete. To validate both the two-step GMM routine and its GMM-CUE counterpart, we write a simple certification script for each routine. First, let's check to see that our two-step routine works for both *i.i.d.* and heteroskedastic errors. Our `mygmm2s` routine returns the same results as `ivreg2` from `SSC` for the several objects included in the `savedresults compare` validation command, and these functions of `ivreg2` have been certified against official Stata's `ivregress`.

Now we construct and run a similar certification script to validate the GMM-CUE routine:

```
. quietly ivreg2 lw s expr tenure rns smsa  (iq=med kww age mrt), cue
r; t=7.18 9:39:41

. savedresults save ivreg2cue e()
r; t=0.00 9:39:41

. mygmmcue lw, endog(iq) inexog(s expr tenure rns smsa) ///
>             exexog(med kww age mrt)
Iteration 0:  f(p) =  61.136598
Iteration 1:  f(p) =  32.923655
Iteration 2:  f(p) =   32.83694
Iteration 3:  f(p) =  32.832195
Iteration 4:  f(p) =  32.831616
Iteration 5:  f(p) =  32.831615

GMM-CUE estimates                                     Number of obs = 758
--------------------------------------------------------------------------
          lw |     Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+------------------------------------------------------------
          iq |  -.0755427   .0132447    -5.70   0.000    -.1015018   -.0495837
           s |   .3296909   .0430661     7.66   0.000      .245283    .4140989
        expr |   .0098901   .0184522     0.54   0.592    -.0262755    .0460558
      tenure |   .0679955   .0224819     3.02   0.002     .0239317    .1120594
         rns |  -.3040223   .0896296    -3.39   0.001    -.4796931   -.1283515
        smsa |   .2071594   .0797833     2.60   0.009      .050787    .3635318
       _cons |   8.907018   .8754361    10.17   0.000     7.191194    10.62284
--------------------------------------------------------------------------

Sargan-Hansen J statistic:   32.832
Chi-sq(  3  )         P-val = 0.0000

r; t=0.57 9:39:42
```

```
. savedresults compare ivreg2cue e(), include(macros: depvar scalar: N j matrix
> : b V) ///
>               tol(1e-4) verbose
comparing macro  e(depvar)
comparing scalar e(N)
comparing scalar e(j)
comparing matrix e(b)
comparing matrix e(V)
r; t=0.01 9:39:42

.
. quietly ivreg2 lw s expr tenure rns smsa  (iq=med kww age mrt), cue robust
r; t=13.46 9:39:55
. savedresults save ivreg2cue e()
r; t=0.00 9:39:55


. mygmmcue lw, endog(iq) inexog(s expr tenure rns smsa) ///
           exexog(med kww age mrt) robust
Iteration 0:  f(p) =  52.768916
Iteration 1:  f(p) =  28.946591  (not concave)
Iteration 2:  f(p) =  27.417939  (not concave)
Iteration 3:  f(p) =  27.041838
Iteration 4:  f(p) =  26.508996
Iteration 5:  f(p) =  26.420853
Iteration 6:  f(p) =  26.420648
Iteration 7:  f(p) =  26.420637
Iteration 8:  f(p) =  26.420637
```

```
GMM-CUE estimates                                   Number of obs = 758
------------------------------------------------------------------------------
             |               Robust
          lw |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          iq |  -.0770701   .0147825    -5.21   0.000    -.1060433    -.048097
           s |   .3348492   .0469881     7.13   0.000     .2427542    .4269441
        expr |   .0197632   .0199592     0.99   0.322     -.019356    .0588825
      tenure |   .0857848   .0242331     3.54   0.000     .0382888    .1332807
         rns |  -.3209864    .091536    -3.51   0.000    -.5003937   -.1415791
        smsa |    .255257   .0837255     3.05   0.002      .091158     .419356
       _cons |   8.943698   .9742228     9.18   0.000     7.034257    10.85314
------------------------------------------------------------------------------
Sargan-Hansen J statistic:  26.421
Chi-sq(  3  )        P-val = 0.0000
r; t=0.68 9:39:56
. savedresults compare ivreg2cue e(), include(macros: depvar scalar: N j matrix
> : b V) ///
>              tol(1e-4) verbose
comparing macro  e(depvar)
comparing scalar e(N)
comparing scalar e(j)
comparing matrix e(b)
comparing matrix e(V)
```

We find that the estimates produced by `mygmmcue` are reasonably close to those produced by the `ml`-based optimization routine employed by `ivreg2`. As we sought to speed up the calculation of GMM-CUE estimates, we are pleased to see the timings displayed by `set rmsg on`. For the non-robust estimates, `ivreg2` took 6.07 seconds, while `mygmmcue` took 0.51 seconds: a twelve-fold speedup.

For the robust CUE estimates, `ivreg2` required 12.69 seconds, compared to 0.64 seconds for `mygmmcue`: an amazing twenty times faster. Calculation of the robust covariance matrix using Mata's matrix operations is apparently much more efficient from a computational standpoint.

We look forward to further improvements in our ability to implement estimators such as GMM-CUE in Stata version 11, with its support for a general-purpose GMM command.

We find that the estimates produced by `mygmmcue` are reasonably close to those produced by the `ml`-based optimization routine employed by `ivreg2`. As we sought to speed up the calculation of GMM-CUE estimates, we are pleased to see the timings displayed by `set rmsg on`. For the non-robust estimates, `ivreg2` took 6.07 seconds, while `mygmmcue` took 0.51 seconds: a twelve-fold speedup.

For the robust CUE estimates, `ivreg2` required 12.69 seconds, compared to 0.64 seconds for `mygmmcue`: an amazing twenty times faster. Calculation of the robust covariance matrix using Mata's matrix operations is apparently much more efficient from a computational standpoint.

We look forward to further improvements in our ability to implement estimators such as GMM-CUE in Stata version 11, with its support for a general-purpose GMM command.

We find that the estimates produced by `mygmmcue` are reasonably close to those produced by the `ml`−based optimization routine employed by `ivreg2`. As we sought to speed up the calculation of GMM-CUE estimates, we are pleased to see the timings displayed by `set rmsg on`. For the non-robust estimates, `ivreg2` took 6.07 seconds, while `mygmmcue` took 0.51 seconds: a twelve-fold speedup.

For the robust CUE estimates, `ivreg2` required 12.69 seconds, compared to 0.64 seconds for `mygmmcue`: an amazing twenty times faster. Calculation of the robust covariance matrix using Mata's matrix operations is apparently much more efficient from a computational standpoint.

We look forward to further improvements in our ability to implement estimators such as GMM-CUE in Stata version 11, with its support for a general-purpose GMM command.