14.9 A GMM-CUE estimator using Mata's optimize() functions (with Mark E. Schaffer)

The problem: We would like to implement the continuously-updated generalized method of moments estimator (GMM-CUE) of Hansen et al. (1996)) in Mata.[20] 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 (`ml`, [R] **ml**). Although that is a workable solution, it can be very slow for large datasets with many regressors and instruments. In Stata version 10, a full-featured suite of optimization commands are available in Mata as `optimize()` ([M-5] **optimize**). We implement a simple IV-GMM estimator in Mata and use that as a model for implementing GMM-CUE.

The two-step GMM estimator for a linear instrumental variables regression model reduces to standard IV if we assume an *i.i.d.* error process, or if the equation is exactly identified with the number of instruments equal to the number of regressors.[21] 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`.

```
. type mygmm2s.ado
*! mygmm2s 1.0.2 MES/CFB 08apr2008
program mygmm2s, eclass
        version 10
/*
  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.
```

20. The rationale for this estimator is presented in Baum et al. (2007), pp. 477–479.
21. See Baum et al. (2007), pp. 467–469.

```
*/
        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 af
> terwards.
        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()` ([M-5] **st_tsrevar()**) function is used to deal with Stata's time series operators in any of the variable lists.

```
. type m_mygmm2s.mata
mata:mata clear
version 10
```

```
mata:
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
mata: mata mosave m_mygmm2s(), dir(PERSONAL) replace
```

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 (see Section 13.10.3).

```
. type m_myomega.mata
mata: mata clear
version 10
mata:
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` ([R] **ivregress**) or our own `ivreg2`.

To implement the GMM-CUE estimator,[22] we clone `mygmm2s.ado` to `mygmmcue.ado`. The ado-file code is very similar:

```
. type mygmmcue.ado
*! mygmmcue 1.0.2 MES/CFB 08apr2008
program mygmmcue, eclass
        version 10
        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)
// Prepare row/col names
// Our convention is that regressors are [endog   included exog]
// and instruments are                   [excluded exog  included exog]
        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` ([M-2] **declarations**) 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,

22. See Baum et al. (2007), pp. 477–480.

but treating them as `external` requires less housekeeping.[23] 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 (2007b), sets up a Mata *structure*, in our case named `S`, containing all elements of the problem.[24] 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 Section 13.8). 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()` toprovide 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()`.

```
. type m_mygmmcue.mata
mata: mata clear
version 10
mata:
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
```

23. As discussed in [M-5] **optimize()**, pp. 287–288, you may always pass up to nine extra arguments to evaluators. However, you must then keep track of those arguments and their order.
24. See Section 13.9.

```
        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())
        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`.

```
. type m_mycuecrit.mata
mata: mata clear
version 10
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.
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 (see Section 11.12) for each routine. First, let's check to see that our two-step routine works for both *i.i.d.* and heteroskedastic errors:

```
. cscript mygmm2s adofile mygmm2s
-------------------------------------------------------------------------------BEGIN mygmm2s

-> which mygmm2s
./mygmm2s.ado
*! mygmm2s 1.0.2 MES/CFB 08apr2008

. set more off

.
. mata: mata clear

. program drop _all

. use http://fmwww.bc.edu/ec-p/data/hayashi/griliches76.dta, clear
(Wages of Very Young Men, Zvi Griliches, J.Pol.Ec. 1976)

.
. quietly ivreg2 lw s expr tenure rns smsa  (iq=med kww age mrt), gmm2s

. savedresults save ivhomo e()

. mygmm2s lw, endog(iq) inexog(s expr tenure rns smsa) ///
>             exexog(med kww age mrt)

Two-step GMM results                                    Number of obs = 758
```

lw	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
iq	-.0115468	.0052169	-2.21	0.027	-.0217717	-.001322
s	.1373477	.0169631	8.10	0.000	.1041007	.1705947
expr	.0338041	.007268	4.65	0.000	.019559	.0480492
tenure	.040564	.0088553	4.58	0.000	.023208	.0579201
rns	-.1176984	.0353037	-3.33	0.001	-.1868924	-.0485043
smsa	.149983	.0314254	4.77	0.000	.0883903	.2115757
_cons	4.837875	.3448209	14.03	0.000	4.162038	5.513711

```
Sargan-Hansen J statistic:  61.137
Chi-sq(  3  )       P-val = 0.0000

. savedresults compare ivhomo e(), include(macros: depvar scalar: N j matrix: b
>  V) ///
>              tol(1e-7) verbose
comparing macro  e(depvar)
comparing scalar e(N)
comparing scalar e(j)
comparing matrix e(b)
comparing matrix e(V)

.
. quietly ivreg2 lw s expr tenure rns smsa  (iq=med kww age mrt), gmm2s robust

. savedresults save ivrobust e()

. quietly mygmm2s lw, endog(iq) inexog(s expr tenure rns smsa) ///
>                     exexog(med kww age mrt) robust

. savedresults compare ivrobust e(), include(macros: depvar scalar: N j matrix:
>  b V) ///
>              tol(1e-7) verbose
comparing macro  e(depvar)
comparing scalar e(N)
comparing scalar e(j)
comparing matrix e(b)
comparing matrix e(V)
```

Our `mygmm2s` routine returns the same results as `ivreg2` (Baum et al. (2007)) for the several objects included in the `savedresults compare` validation command.[25] Now we construct and run a similar script to validate the GMM-CUE routine:

```
. cscript mygmmcue adofile mygmmcue
-------------------------------------------------------------------------------BEGIN mygmmcue

-> which mygmmcue
./mygmmcue.ado
*! mygmmcue 1.0.2 MES/CFB 08apr2008

. set more off

. mata: mata clear

. program drop _all

. set rmsg on
r; t=0.00 9:57:59

. use http://fmwww.bc.edu/ec-p/data/hayashi/griliches76.dta, clear
(Wages of Very Young Men, Zvi Griliches, J.Pol.Ec. 1976)
r; t=0.18 9:57:59

.
. quietly ivreg2 lw s expr tenure rns smsa  (iq=med kww age mrt), cue
r; t=6.07 9:58:05

. savedresults save ivreg2cue e()
r; t=0.00 9:58:05

. 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
```

25. These functions of `ivreg2` have been certified against official Stata's `ivregress`.

```
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=0.51 9:58:06
. 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 9:58:06
.
. quietly ivreg2 lw s expr tenure rns smsa  (iq=med kww age mrt), cue robust
r; t=12.69 9:58:19
. savedresults save ivreg2cue e()
r; t=0.00 9:58:19
. 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
```

lw	Coef.	Robust 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=0.64 9:58:19
. 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 9:58:19
```

We find that the estimates produced by `mygmmcue` are reasonably close to those produced by the different 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.

Although this Mata-based GMM-CUE routine for linear instrumental variables models is merely a first stab at taking advantage of Mata's efficiency, it is evident that the approach has great potential for the development of more readable and efficient code.