GMM estimation in Mata

Using Stata's new optimizer to program estimators

Austin Nichols

July 24, 2008

optimize() is exciting stuff

- The new (as of Stata 10) `optimize` function in Mata is exciting.
- You can use it e.g. to find maxima of a function, solve a difficult nonlinear system of equations, or write a new estimator.
- Likely suspects: Generalized Methods of Moments (GMM) or Minimum Distance estimators (MDE).
  - More on GMM: Hansen (1982)
  - More on MDE: Chamberlain (1982, 1984)
  - More on both: Wooldridge (2002) chapter 14
- Today: a couple of quick examples of GMM estimators; see `ivpois` on SSC for a more detailed example.

The OLS model

Consider the most common regression framework:

$$y = X\beta + \varepsilon$$

where we assume $E(X'\varepsilon) = 0$ so our estimator $\widehat{\beta}$ is unbiased.

The usual approach is to define

$$\widehat{\beta}_{OLS} = (X'X)^{-1}(X'y)$$

that minimizes the sum of squared residuals

$$\sum (y - \widehat{y})^2 = \sum (y - X\widehat{\beta})^2$$

and has an easy solution.

The GMM model

Could also define $\widehat{\beta}_{GMM}$ that gets $E(X'\widehat{\varepsilon})$ as close to zero as possible in the sample (zero, in fact, with a constant) by minimizing the quadratic form

$$(X'e)'A(X'e)$$

for some weighting matrix A, where e is a function of the coefficient $e = y - Xb$.

This is the basic idea of GMM: if you know a population moment $g(\beta)$ is zero, try to get the analogous sample moment as close to zero as possible by minimizing its "square." Hansen (1982, 1984) discusses the asymptotic properties of this approach, and Hayashi (2000), Wooldridge (2002, ch.14), and Baum, Schaffer, and Stillman (2003, 2007) discuss the intuition and practical implementation.

The GMM family of models

In fact, each criterion function of the form

$$g(b)' \cdot A \cdot g(b)$$

defines a family of estimators, one for each weighting matrix A (including an identity matrix as one possibility). If A is chosen to be the inverse of the variance of $g(b)$ we get an efficient estimator. If A is chosen to be a consistent estimator of the inverse of the asymptotic variance of $g(b)$ we get an asymptotically efficient estimator—there are a few ways to do this (see Baum, Schaffer, and Stillman 2007 for a clear exposition).

The optimize() routine in Mata is surprisingly easy to use and all the documentation is on the web at http://stata.com/help.cgi?mata. That said, it can a bit confusing to set up a problem the first time.

First you need to set up a function with the criterion function (the function to be minimized):

```
mata:
 void i_crit(todo,b,crit,g,H)
 {
  external y,X,W
  m=X'(y-X*b')
  crit=(m*W*m')
 }
```

where the external declaration allows the y,X,W to be passed back and forth among Mata functions. `m` is the moment function $g(b)$ and `crit` is the criterion function. `g` and `H` are the gradient and Hessian, which we aren't calculating here.

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ gets the dep var “earnings”

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ makes a constant term

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ gets the RHS var “education”

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ computes the weighting matrix

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ makes a starting guess at a parameter vector (all zeros)

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ creates a "name" for the optimization problem

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ names the function to optimize

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ tells optimize to minimize, not maximize

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ d0 says we won't calculate gradient or Hessian

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ puts in the starting guess at a parameter vector

Then you can set up the problem in Mata and do the optimization in a handful of lines:

```
y = st_data(., "earnings")
cons=J(rows(y),1,1)
X = st_data(., "education"), cons
W=cholinv(X'X)
init=J(1,cols(X),0)
S=optimize_init()
optimize_init_evaluator(S, &i_crit())
optimize_init_which(S,"min")
optimize_init_evaluatortype(S,"d0")
optimize_init_params(S,init)
p=optimize(S)
```

▷ does the optimization and puts the parameter vector in p

That line optimize_init_evaluator tells the optimizer what Mata function is going to calculate the value of the criterion function, which is called the evaluator function in Mata. The criterion function can be scalar-valued for d-type evaluator functions, or vector-valued for v-type evaluator functions, and the d or v is modified with a number indicated how many derivatives it can calculate:

```
 type  Capabilities expected of evaluator()
-----------------------------------------------------
  0    can calculate f(p)
  1    can calculate f(p) and g=f'(p)
  2    can calculate f(p) and g=f'(p) and H=f''(p)
-----------------------------------------------------
```

A side note: instead of declaring the data matrices as externals, you can use a function to add an argument to the evaluator function:

```
 optimize_init_argument(S, 1, X)
```

which lowers your chance of making a mistake by using the same name for different objects.

A better way is to set up a Mata function that takes Stata variables as arguments, does the optimization, and stores the result in a Stata matrix:

```
void i_ols(string scalar lhs, string scalar rhs, string scalar ok)
{
 external y,X,W
 y = st_data(., tokens(lhs), ok)
 cons = J(rows(y),1,1)
 X = st_data(., tokens(rhs), ok), cons
 W = cholinv(X'X)
 init = st_matrix("b")
 S = optimize_init()
 optimize_init_evaluator(S, &i_crit())
 optimize_init_which(S,"min")
 optimize_init_evaluatortype(S,"d0")
 optimize_init_params(S,init)
 p = optimize(S)
 st_replacematrix("b",p)
 }
```

Then you can put both Mata functions at the end of a little ado file `ols.ado`:

```
prog ols, eclass
 version 10
 syntax varlist [if] [in]
 marksample touse
 gettoken lhs rhs : varlist
 mat b = J(1,`:word count `rhs' _cons',0)
 matname b `rhs' _cons, c(.)
 mata: i_ols("`lhs'", "`rhs'", "`touse'")
 eret post b, e(`touse') depname(`lhs')
 eret di
end
mata:
 void i_crit(todo,b,crit,g,H)
 {
  external y,X,W
  m=X'(y-X*b')
  crit=(m'*W*m)
 }
 void i_ols(string scalar lhs, string scalar rhs, string scalar ok)
 {
  external y,X,W
  y = st_data(., tokens(lhs), ok)
  cons = J(rows(y),1,1)
  X = st_data(., tokens(rhs), ok), cons
  W = cholinv(X'X)
  init = st_matrix("b")
  S = optimize_init()
  optimize_init_evaluator(S, &i_crit())
  optimize_init_which(S,"min")
  optimize_init_evaluatortype(S,"d0")
  optimize_init_params(S,init)
  p = optimize(S)
  st_replacematrix("b",p)
  }
end
```

Now you can run a GMM version of OLS, and bootstrap for standard errors:

```
use http://fmwww.bc.edu/ec-p/data/wooldridge/card, clear
bs: ols lwage educ
bs: reg lwage educ
```

Programming asymptotic standard error calculations for GMM is just a bit more work, but offers the advantage that once you calculate the gradient, you can improve the speed of optimization by choosing method `d1` (see help mf_optimize).

Note how easily you can shift to an instrumental variables (IV) model by assuming $E(Z'\varepsilon) = 0$ instead of $E(X'\varepsilon) = 0$.

Poisson regression

A Poisson regression assumes that the outcome is described by a conditional mean which is an exponentiated linear combination of X i.e.

$$E(y|X) = exp(X\beta)$$

so it is appropriate for a wide variety of models where the dependent variable is nonnegative (zero or positive), not just where the dependent variable measures counts of events. Wherever you might be inclined to take the logarithm of a nonnegative dependent variable y and use OLS, Poisson regression offers an alternative that includes observations where y is zero. Just as with a regression of log dependent variable on X, the interpretation of estimates is as marginal effects in percentage terms, e.g. a coefficient of 0.05 indicates a one-unit increase in X is associated with a 5% increase in y.

GMM IV Poisson

Mullahy (1997) proposed a GMM estimator suitable for endogenous X. If we assume

$$y = exp(X\beta)\varepsilon$$

then $y \cdot exp(-X\beta) - 1$ should be orthogonal to a set of instruments Z:

$$E[Z'(y \cdot exp(-X\beta) - 1)] = 0$$

For this case, wherever you might be inclined to take the logarithm of a nonnegative dependent variable y and use IV, the GMM estimator offers an alternative that includes observations where y is zero.

Multiply or add

Given $E[y|X] = exp(X\beta)$, one can assume either an additive error or a multiplicative error, which produce different versions of the moment conditions. The additive form for the error posits that $y = exp(X\beta) + u$ and gives moment conditions of the form $Z'(y - exp(X\beta)) = 0$, whereas the multiplicative form posits $y = exp(X\beta)u$ and gives moment conditions of the form $E[Z'(y \cdot exp(-X\beta) - 1))] = 0$ for instruments Z (where Z includes all exogenous variables, both included and excluded instruments).

Angrist (2001) shows that in a model with endogenous binary treatment and a binary instrument, the latter procedure (assuming a multiplicative error) estimates a proportional local average treatment effect (LATE) parameter in models with no covariates. The latter is also more intuitively appealing and congruent with Poisson and GLM, and the assumption can be rewritten $y = exp(X\beta)u = exp(X\beta)exp(v) = exp(X\beta + v)$ so $ln(y) = X\beta + v$ (assuming $y > 0$) to provide the natural link to OLS. Windmeijer (2006) contains a useful discussion and further related models.

Moment conditions

Recall that if we assume

$$y = exp(X\beta)\varepsilon$$

then $y \cdot exp(-X\beta) - 1$ should be orthogonal to a set of instruments Z:

$$E[Z'(y \cdot exp(-X\beta) - 1)] = 0$$

This translates very easily into the Mata code:

```
mata:
 void i_civp(todo,b,crit,g,H)
 {
  external y,X,Z,W
  m=Z'((y:*exp(-X*b'):- 1))
  crit=(m'*W*m)
 }
```

There are two changes: the `m=` something different, and we add `Z` to the external declaration.

optimize() code

The program that calls optimize() merely adds Z:

```
void i_ivp(string scalar lhs, string scalar rhs, string scalar z, strin
 {
  external y,X,Z,W
  y = st_data(., tokens(lhs), ok)
  cons = J(rows(y),1,1)
  X = st_data(., tokens(rhs), ok), cons
  Z = st_data(., tokens(z), ok), cons
  W = cholinv(Z'Z)
  init = st_matrix("b")
  S = optimize_init()
  optimize_init_evaluator(S, &i_civp())
  optimize_init_which(S,"min")
  optimize_init_evaluatortype(S,"d0")
  optimize_init_params(S,init)
  p = optimize(S)
  st_replacematrix("b",p)
  }
```

optimize() code

But the main program in a do-file has a little bit more work to do:

```
prog ivp, eclass
 version 10
 syntax varlist [if] [in] [, exog(varlist) endog(varlist)]
 marksample touse
 markout ‘touse’ ‘exog’ ‘endog’
 gettoken lhs varlist:varlist
 loc rhs: list varlist | endog
 loc z: list varlist | exog
 loc z: list z - endog
 mat b = J(1,‘:word count ‘rhs’ _cons’,0)
 matname b ‘rhs’ _cons, c(.)
 mata: i_ivp("‘lhs’", "‘rhs’", "‘z’", "‘touse’")
 eret post b, e(‘touse’) depname(‘lhs’)
 eret di
end
mata:
 void i_civp(todo,b,crit,g,H)
 {
  external y,X,Z,W
  m=Z’((y:*exp(-X*b’):- 1))
  crit=(m’*W*m)
 }
 void i_ivp(string scalar lhs, string scalar rhs, string scalar z, string scalar ok)
 {
  external y,X,Z,W
  y = st_data(., tokens(lhs), ok)
  cons = J(rows(y),1,1)
  X = st_data(., tokens(rhs), ok), cons
  Z = st_data(., tokens(z), ok), cons
  W = cholinv(Z’Z)
  init = st_matrix("b")
  S = optimize_init()
```

optimize() code

If you save the program as ivp.ado, you can run a GMM-IV-Poisson model easily and compare:

```
use http://fmwww.bc.edu/ec-p/data/wooldridge/card, clear
ssc inst ivpois, replace
ssc inst ivreg2, replace
bs: ivp wage educ, endog(educ) exog(nearc4)
bs: ivpois wage educ, endog(educ) exog(nearc4)
ivpois wage educ, endog(educ) exog(nearc4)
ivreg2 lwage (educ=nearc4)
```

The SSC program `ivpois` is just a longer version of `ivp` with some extras (checking for various errors, collinearity, etc.), but not nearly as developed as `ivreg2` (also on SSC).

Structures

A better way to handle some of the passing of arguments and functions is to define a Mata structure (see Gould 2007 and help M-2 struct).

We could put all the vectors, matrices, scalars like `rows(y)`, etc. in one structure and refer to that structure in the various functions we might need. No time for a detailed look at that approach today.

quadcross

I should be using `quadcross` for all the matrix multiplication above.

But this:

```
m=X'(y-X*b')
```

is a bit easier to read than:

```
m=quadcross(X,(y-quadcross(X',b')))
```

Syntax choices for a hypothetical `gmm`

The fact that the only moving parts so far have been the moment condition m and the form of the weight matrix W suggests that a simple way to create a general `gmm` command is to ask the user to supply a varlist, or possibly more than one, and to supply a moment condition in terms of y and X, in the spirit of `twoway function`. A temporary file can be written out via the `file` command just like the above ado files, but with the appropriate text inserted.

Of course, a more sophisticated version of the command would request a gradient, do some checking of whether the user has made mistakes, etc. A different approach would require the user to compile a Mata function ahead of time, and supply the name of the function. I don't think these approaches are *necessarily* incompatible, but I'd like to poll Stata users present to see what makes sense to them. To you, I mean.

References

Angrist, Joshua D. 2001. "Estimation of limited dependent variable models with dummy endogenous regressors: simple strategies for empirical practice." *Journal of Business and Economic Statistics*, 19:2-16.

Baum, Christopher F., Mark E. Schaffer, and Steven Stillman. 2003. "Instrumental variables and GMM: Estimation and testing." *Stata Journal*, 3(1): 1-31.

Baum, Christopher F., Mark E. Schaffer, and Steven Stillman. 2007. "Enhanced routines for instrumental variables/GMM estimation and testing." *Stata Journal*, 7(4): 465–506.

Chamberlain, G. 1982. "Multivariate regression models for panel data." *Journal of Econometrics*, 18: 5–46.

Chamberlain, G. 1984. "Panel Data." in Griliches, Zvi and M.D. Intrilligator, eds. *Handbook of Econometrics*, Vol.2 Ch.22: 1247–1318.

References

Gould, William. 2007. "Mata Matters: Structures." *Stata Journal*, 7(4): 556–570.

Hansen, Lars Peter. 1982. "Large Sample Properties of Generalized Method of Moments Estimators." *Econometrica*, 50: 1029–1054.

Hayashi, Fumio. 2000. *Econometrics*. 1st ed. Princeton, NJ: Princeton University Press.

Mullahy, John. 1997. "Instrumental-Variable Estimation of Count Data Models: Applications to Models of Cigarette Smoking Behavior." The Review of Economics and Statistics, 79(4):586-593.

Windmeijer, Frank. 2006. "GMM for Panel Count Data Models." Discussion Paper No. 06/591, Department of Economics, University of Bristol.

Wooldridge, J.M. 2002.
Econometric Analysis of Cross Section and Panel Data. Cambridge, MA: MIT Press.