GMM and MINZ Program Libraries for MATLAB

Michael T. Cliff*
Krannert Graduate School of Management
Purdue University

March 2, 2003

This document accompanies the GMM and MINZ software libraries for MATLAB which complement and build from James LeSage's Econometrics Toolbox.[1] A brief overview of GMM estimation from a theoretical perspective[2] is followed by a discussion on how to use the GMM portion of the software. Since the `gmm` routine relies on the MINZ optimization library, a discussion of MINZ follows. Direct interaction with this optimization library is not necessary for most users. However, the libraries are written with flexibility in mind, so more advanced users are able to substitue their own routines if desired. As a general rule, the user does not need to change the provided code. Various options are selected by the arguments passed to the provided functions, not by editing the functions themselves. The idea is to use a common set of programs for different applications, eliminating the need to maintain several different versions of a program. Of course, it also means that care must be taken when modifying the central programs, as the changes will affect many different projects.

To make the discussion concrete, several demo programs are discussed in the final section. Some of these focus on the GMM portion of the code, others on the MINZ optimization library. You can think of each demo as being a seperate project and see how the central code is used to estimate a variety of models.

1 What is GMM?

GMM, the Generalized Method of Moments, is an econometric procedure for estimating the parameters of a model. Hansen (1982) developed GMM as an extension to the classical method of moments estimators dating back more than a century. The basic idea is to choose parameters of the model so as to match the moments of the model to those of the data as

*Address correspondence to Mike Cliff, Krannert Graduate School of Management, 1310 Krannert Building, Purdue University, West Lafayette, IN 47907–1310. Phone: (765) 496–7514; e-mail: `mcliff@mgmt.purdue.edu`. The software is available for download from `http://www.mgmt.purdue.edu/faculty/mcliff/progs.html`.

[1]The Econometrics Toolbox is available from `http://www.econ.utoledo.edu/matlab_gallery`.

[2]There are many excellent textbook treatments, including Cochrane (2001), Davidson and MacKinnon (1993), Greene (1997), and Hamilton (1994).

closely as possible. The moment conditions are chosen by the analyst based on the problem at hand. A weighting matrix determines the relative importance of matching each moment. Most common estimation procedures can be couched in this framework, including ordinary least squares, instrumental variables estimators, two-stage least squares, and in some cases maximum likelihood. An example provides an illustration of the similarity to OLS in the linear model.

It is important to realize the generality of GMM (hence the 'G'). Thus, saying that "I estimate the parameters via GMM" is essentially meaningless unless additional details are provided. Without any additional information, this statement is about as informative as saying "I use a computer to estimate the parameters" or "my estimates are based on econometrics."

A key advantage to GMM over other estimation procedures is that the statistical assumptions required for hypothesis testing are quite weak. Of course, nothing comes for free. The cost is a loss of efficency over methods such as Maximum Likelihood (MLE). One can view MLE as a limiting case of GMM: under MLE the distribution of errors is specifed so in a sense all of the moments are incorporated. The trouble with MLE is often that the errors may not follow a known distribution (such as the Normal, which is almost the universal standard in MLE). Thus, GMM offers a compromise between the efficiency of MLE and robustness to deviations from normality (or other distributional forms). Also note that, except for some special cases, the GMM results are asymptotic.

2 Theory behind GMM

GMM chooses the parameters which minimize the quadratic

$$J_T = \mathbf{m}(\boldsymbol{\theta})'\mathbf{W}\mathbf{m}(\boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{\theta}$ is a k-vector of parameters, $\mathbf{m}(\boldsymbol{\theta})$ is a L-vector of orthogonality conditions, and $\mathbf{W}$ is an $L \times L$ positive definite weighting matrix. The objective function has a least-squares flavor. You can see that "I use GMM to estimate $\boldsymbol{\theta}$" doesn't mean much without some additional details such as what $\mathbf{m}(\boldsymbol{\theta})$ and $\mathbf{W}$ look like. The next two subsections address these in turn.

2.1 Orthogonality Conditions

The moment conditions $\mathbf{m}(\boldsymbol{\theta})$ set means of functions of the data and parameters to zero. One simple restriction estimates the mean μ of data y_t

$$E[y_t] = \mu$$

giving the population orthogonality condition

$$E[y_t - \mu] = 0$$

and sample counterpart

$$\mathbf{m}(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} y_t - \mu.$$

Another restriction, on the variance (σ^2), is

$$E[(y_t - \mu)^2] = \sigma^2 \qquad \text{giving the system} \qquad E\begin{bmatrix} y_t - \mu \\ (y_t - \mu)^2 - \sigma^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Note that the moment condition for the mean is needed to estimate the variance. Similarly, a covariance restriction would be

$$E[(x_t - \mu_x)(y_t - \mu_y)] = \sigma_{x,y} \qquad \text{giving} \qquad E\begin{bmatrix} x_t - \mu_x \\ y_t - \mu_y \\ (x_t - \mu_x)(y_t - \mu_y) - \sigma_{x,y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

The terms μ_x, μ_y, σ_x, σ_y and $\sigma_{x,y}$ are parameters we wish to estimate, whereas x_t and y_t are data. An example is provided with the code to estimate the means and covariance matrix of a dataset.

A key ingredient to GMM is the specification of the moment, or orthogonality, conditions $\mathbf{m}(\boldsymbol{\theta})$. The moment conditions are commonly based on the error terms from an economic model. Consider a general model of the form

$$\mathbf{y}_{[T\times 1]} = \mathbf{f}(\mathbf{X}_{[T\times k]}; \boldsymbol{\theta}) + \boldsymbol{\varepsilon} \tag{2}$$

where $\mathbf{f}$ can be a nonlinear function. We then will need $L \geq k$ (independent) restrictions in order to identify the k-vector of parameters, $\boldsymbol{\theta}$. The moment conditions restrict unconditional means of the data to be zero. The population version of each of these restrictions ($\ell = 1, \ldots, L$) is of the form

$$E[m_\ell(\mathbf{y}, \mathbf{X}; \boldsymbol{\theta})] = 0.$$

The sample analog is

$$m_\ell(\mathbf{y}, \mathbf{X}; \hat{\boldsymbol{\theta}}) = \frac{1}{T}\sum_{t=1}^{T} m_{\ell,t}(\mathbf{y}_t, \mathbf{x}_t; \hat{\boldsymbol{\theta}})$$

where $\mathbf{y}_t$ and $\mathbf{x}_t$ denote row t of the matrices $\mathbf{y}$ and $\mathbf{X}$, transposed to be column vectors. Note that $\mathbf{m}_t$ (with the time subscript) indicates an observation-by-observation set of values, while $\mathbf{m}$ (no time subscript) indicates the moment (average) of the $\mathbf{m}_t$'s.

The moment conditions utilized, though somewhat arbitrary, are often guided by economic principles and the model of interest. For example, in finance the return on an asset this period is generally modeled as unpredictable by (orthogonal to) information in prior periods, so moment conditions often incorporate past returns, interest rates, etc.

Note that there must be at least as many moment conditions as there are parameters to achieve identification. If you have too few restrictions, you can "create" more by using

instruments. Returning to (2), suppose $E[\varepsilon_t \mathbf{x}_t] \neq \mathbf{0}$, but that $E[\varepsilon_t \mathbf{z}_t] = \mathbf{0}$. The $\mathbf{z}_t$'s are referred to as instruments. In sample, the model errors are

$$\mathbf{e}(\hat{\boldsymbol{\theta}}) = \mathbf{y} - \mathbf{f}(\mathbf{X}; \hat{\boldsymbol{\theta}}). \tag{3}$$

giving the moment conditions

$$\mathbf{m}(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} \mathbf{z}_t e(\mathbf{y}_t, \mathbf{x}_t; \boldsymbol{\theta}) = \frac{1}{T}\mathbf{Z}'\mathbf{e}(\mathbf{y}, \mathbf{X}; \hat{\boldsymbol{\theta}}). \tag{4}$$

This can actually be generalized for simultaneous equations by letting $\mathbf{e}_t$ represent the vector of residuals for each equation at time t, giving $\mathbf{m}_t = \boldsymbol{\varepsilon}_t \otimes \mathbf{z}_t$. The notation $\otimes$ indicates the Kroneker product, multiply every element of $\boldsymbol{\varepsilon}_t$ by $\mathbf{z}_t$ in (9).

This approach with the instruments changes the question of "which moments" to "which $\mathbf{Z}$." It is common to include a constant as an instrument to restrict the model errors to have mean zero. For purposes of using the software, understanding (3) and (4) is crucial. The user will need to provide an m-file which returns these objects. Equation 3 provides the errors from the economic model, like the residuals in OLS. Equation 4 returns the moments that are used to identify the parameters, like the normal equations in OLS. Nearly all other aspects of the software can proceed without the user's intervention.

Illustration: GMM vs. OLS

Suppose we have a simple model $y_t = \alpha + x_t\beta + \varepsilon_t$ and three observations of $\{x_t, y_t\}$: $\{0, 1\}, \{1, 3\}, \{2, 5\}$. We need (at least) two moment conditions to identify α and β. The natural ones to choose are $E[\varepsilon_t] = 0$ and $E[x_t\varepsilon_t] = 0$, the normal equations from OLS. In this case, $\mathbf{z}_t = [1 \;\; x_t]$. In sample,

$$\mathbf{m} = \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \frac{1}{3}\sum_{t=1}^{3} \mathbf{z}_t' e_t = \frac{1}{3}\sum_{t=1}^{3} \begin{bmatrix} 1e_t \\ x_t e_t \end{bmatrix} = \frac{1}{3}\sum_{t=1}^{3} \begin{bmatrix} 1(y_t - \alpha - x_t\beta) \\ x_t(y_t - \alpha - x_t\beta) \end{bmatrix}$$

In this case, the objective function is minimized when $\mathbf{m} = \mathbf{0}$, or

$$m_1 = \frac{1}{3}[(1 - \alpha - 0\beta) + (3 - \alpha - 1\beta) + (5 - \alpha - 2\beta)] = 3 - \alpha - \beta = 0 \tag{5}$$

$$m_2 = \frac{1}{3}[0(1 - \alpha - 0\beta) + 1(3 - \alpha - 1\beta) + 2(5 - \alpha - 2\beta) = \frac{1}{3}(13 - 3\alpha - 5\beta)] = 0 \tag{6}$$

Equation (5) gives $\alpha = 3 - \beta$, which can be substituted into (6) to get $\beta = 2$, implying $\alpha = 1$.

The above analysis ignored the presence of the weighting matrix $\mathbf{W}$ in the minimization of the objective function (assuming it is equal to the identity matrix). We will now consider the role of the weighting matrix.

2.2 Weighting Matrix

If there are as many moment conditions as parameters, the moments will all be perfectly matched and the objective function J_T in (1) will have a value of zero. This is referred to as the "just-identified" case. In the situation where there are more moment conditions than parameters ("over-identified") not all of the moment restrictions will be satisfied so a weighting matrix $\mathbf{W}$ determines the relative importance of the various moment conditions. An important contribution of Hansen (1982) is to point out that setting $\mathbf{W} = \mathbf{S}^{-1}$, the inverse of an asymptotic covariance matrix, is optimal in the sense that it yields $\hat{\boldsymbol{\theta}}$ with the smallest asymptotic variance. Intuitively, more weight is given to the moment conditions with less uncertainty. $\mathbf{S}$ is also know as the spectral density matrix evaluated at frequency zero. There are many approaches for estimating $\mathbf{S}$ which can account for various forms of heteroskedasticity and/or serial correlation, including White (1980), the Bartlett kernel used by Newey and West (1987), the Parzen kernel of Gallant (1987), the truncated kernel of Hansen (1982) and Hansen and Hodrick (1980), or the "automatic" bandwidth selection from Andrews and Monahan (1992) with Quadratic-Spectral or Tukey-Hanning kernels. Each of these methods is supported in the software.

The Spectral Density matrix for the kernel-based estimators (White, Hansen, Newey-West, and Gallant) is given by

$$\hat{\mathbf{S}} = \hat{\mathbf{S}}_0 + \sum_{j=1}^{J} w(j) \left[\hat{\mathbf{S}}_j + \hat{\mathbf{S}}_j'\right] \tag{7}$$

where

$$\hat{\mathbf{S}}_j = \frac{T}{T-k}\frac{1}{T}\sum_{t=j+1}^{T} \mathbf{m}_t(\hat{\boldsymbol{\theta}})\mathbf{m}_{t-j}(\hat{\boldsymbol{\theta}})' \tag{8}$$

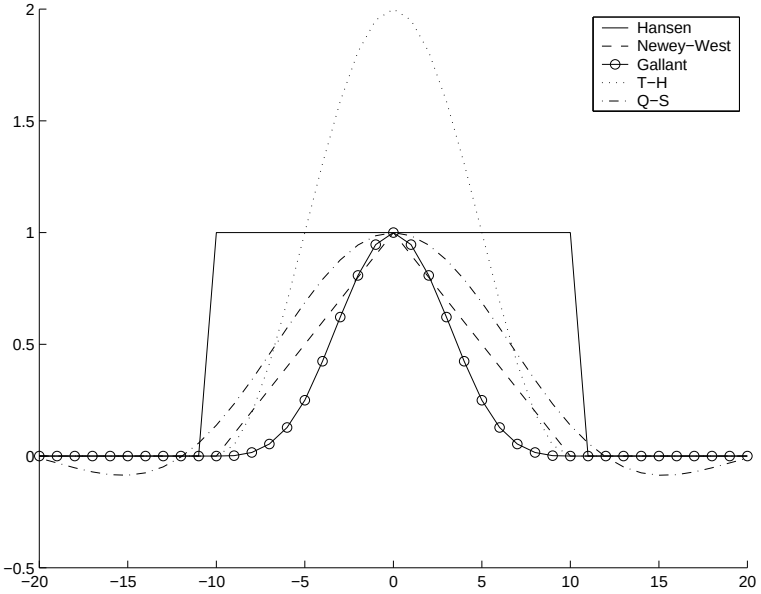
$$= \frac{1}{T-k}\sum_{t=j+1}^{T} \left[\boldsymbol{\varepsilon}_t \otimes \mathbf{z}_t\right]\left[\boldsymbol{\varepsilon}_{t-j} \otimes \mathbf{z}_{t-j}\right]' \tag{9}$$

$$= \frac{1}{T-k}\sum_{t=j+1}^{T} \left[\boldsymbol{\varepsilon}_t\boldsymbol{\varepsilon}_{t-j}' \otimes \mathbf{z}_t\mathbf{z}_{t-j}'\right] \tag{10}$$

The $\frac{T}{T-k}$ term is a small sample degrees of freedom correction. The term $w(j)$ is the kernel weight, and it is what distinguishes the various estimators. Terms beyond the lag truncation parameter J are given weights of zero in kernels other than the Quadratic-Spectral and Tukey-Hanning. Figure 1 shows an example of the weights assigned to each of the kernels.

In general, an "optimal" weighting matrix requires an estimate of the parameter vector, yet at the same time, estimating the parameters requires a weighting matrix. To solve this dependency, common practice is to set the initial weighting matrix to the identity then calculate the parameter estimates. A new weighting matrix is calculated with the last

Figure 1: Weights Used by Various Kernels



parameter estimates, then new parameter estimates with the updated weighting matrix.

$$\mathbf{W}_0 = \mathbf{I} \tag{11}$$

$$\hat{\boldsymbol{\theta}}_1 = \text{argmin } \mathbf{m}(\boldsymbol{\theta})'\mathbf{W}_0\mathbf{m}(\boldsymbol{\theta}) \tag{12}$$

$$\mathbf{W}_1 = f(\hat{\boldsymbol{\theta}}_1) \tag{13}$$

$$\hat{\boldsymbol{\theta}}_2 = \text{argmin } \mathbf{m}(\boldsymbol{\theta})'\mathbf{W}_1\mathbf{m}(\boldsymbol{\theta}) \tag{14}$$

The process can then be iterated futher by calculating $\mathbf{W}_2$ then minimizing to find $\hat{\boldsymbol{\theta}}_3$ and so on. In general, iterating to end with $\hat{\boldsymbol{\theta}}_n$ is called n-stage GMM. You can also iterate until the change in objective function is sufficiently small. I call this approach iterated GMM. The software is written so that the user can easily control this process.

A problem with using the Identity as the initial weighting matrix is that the estimation procedure may be sensitive to the scaling of the data. The objective function is $\mathbf{m}'\mathbf{m} = \sum_{\ell=1}^{L} m_\ell^2$. Consider a single-equation case. Since $m_\ell = \frac{1}{T}\sum_{t=1}^{T} e_t z_{\ell,t}$, the magnitudes of the moments obviously depends on the scaling of the elements of $\mathbf{z}_t$. One way to deal with this problem is to incorporate the magnitudes of the instruments in the weighting matrix by setting $\mathbf{W}_0 = (\mathbf{I}_N \otimes \mathbf{Z}'\mathbf{Z})^{-1}$. N refers to the number of equtions, taken to be one here. As more GMM iterations are used the difference between using the Identity matrix and the second moments of the instruments as a weighting matrix decreases. However, I have found that the approach using instruments can perform much better at fewer iterations. As an example, in one experiment I found that the Identity matrix generated p-values for the model fit (discussed in the next section) of 0.2955, 0.1849, and 0.0854 after 2, 3, and 10 stage GMM

estimation. When using the instruments to form the weighting matrix, the corresponding p-values were 0.0524, 0.0534, and 0.05. The difference in the two approaches will be somewhat problem-specific, but it seems that using the instruments should be at least as good as the Identity approach.

The simple example in the prior section minimized the objective function by setting the moments to zero. In the general case with over-identified systems or nonlinear models, this approach is not appropriate. Instead the function is minimized by setting the gradient of the objective function $\partial[\mathbf{m}(\boldsymbol{\theta})'\mathbf{W}\mathbf{m}(\boldsymbol{\theta})]/\partial\boldsymbol{\theta}$ to zero. The presence of the weighting matrix in the quadratic makes it more convenient in the program to work with the Jacobian of the moment conditions $\partial\mathbf{m}(\boldsymbol{\theta})/\partial\boldsymbol{\theta} = \mathbf{M}(\boldsymbol{\theta})$ than with the gradient of the objective function, $2\mathbf{M}(\boldsymbol{\theta})'\mathbf{W}\mathbf{m}(\boldsymbol{\theta})$. One way to think about what the estimation is doing in this case is to write the first order conditions for minimizing $\mathbf{m}(\boldsymbol{\theta})'\mathbf{W}\mathbf{m}(\boldsymbol{\theta})$ as

$$\mathbf{m}(\boldsymbol{\theta})'\mathbf{W}\mathbf{m}(\boldsymbol{\theta}) = \mathbf{A}\mathbf{m}(\boldsymbol{\theta}) = \mathbf{0}$$

where the matrix $\mathbf{A}$ is a set of weights that specify what linear combinations of $\mathbf{m}$ are set to zero.

From the prior OLS example,

$$\mathbf{M} = \frac{1}{3}\sum_{t=1}^{3}\begin{bmatrix}1 & x_t \\ x_t & x_t^2\end{bmatrix} = \frac{1}{3}\begin{bmatrix}3 & 3 \\ 3 & 5\end{bmatrix}$$

so

$$\mathbf{M}'\mathbf{m} = \begin{bmatrix}1 & 1 \\ 1 & 5/3\end{bmatrix}\begin{bmatrix}3-\alpha-\beta \\ (13-3\alpha-5\beta)/3\end{bmatrix} = \begin{bmatrix}0 \\ 0\end{bmatrix}.$$

Solving gives the same answer as before

$$\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \begin{bmatrix}3 & 4 \\ 9 & 14\end{bmatrix}^{-1}\begin{bmatrix}11 \\ 37\end{bmatrix} = \begin{bmatrix}1 \\ 2\end{bmatrix}.$$

2.3 Hypothesis Testing with GMM

2.3.1 Covariance Matrices

One of the nice properties of GMM is that hypothesis testing is still possible in the presence of heteroskedastic and/or serially correlated errors. I should point out that the reason for this rests in choosing $\mathbf{S}$ to take care of the heteroskedasticity/serial correlation. If your estimate of $\mathbf{S}$ only corrects for heteroskedasticity [e.g., White (1980)], then your standard errors will not be robust to serial correlation.

The distribution of the GMM estimator is given by

$$\hat{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\theta}, \mathbf{V}/T), \quad \text{where} \quad \mathbf{V} = [\mathbf{M}'\mathbf{W}\mathbf{M}]^{-1}\mathbf{M}'\mathbf{W}\mathbf{S}\mathbf{W}\mathbf{M}[\mathbf{M}'\mathbf{W}\mathbf{M}]^{-1}.$$

When the weighting matrix is optimal ($\mathbf{W} = \mathbf{S}^{-1}$), $\mathbf{V} = [\mathbf{M}'\mathbf{S}^{-1}\mathbf{M}]^{-1}$. The reduced expression for $\mathbf{V}$ is also the Gauss-Newton approximation to the inverse Hessian of the objective

function, a feature exploited in the code. In addition to the covariance matrix of the parameter estimates, the GMM framework also provides a covariance matrix of the moment conditions

$$[\mathbf{I} - \mathbf{M}(\mathbf{M}'\mathbf{W}\mathbf{M})^{-1}\mathbf{M}'\mathbf{W}]\mathbf{S}[\mathbf{I} - \mathbf{M}(\mathbf{M}'\mathbf{W}\mathbf{M})^{-1}\mathbf{M}'\mathbf{W}]'/T \tag{15}$$

This matrix also simplifies for the optimal weighting matrix, giving $[\mathbf{S} - \mathbf{M}\mathbf{V}\mathbf{M}']/T$.

2.3.2 Model Fit

A natural question to ask is how well the model fits the data. If the model is "just-identified" there is one parameter for each restriction so the restrictions can be satisfied exactly. If the model is over-identified it will not be possible to set every moment to zero. So the question is how far from zero are we. The answer is provided by the "test of over-identifying restrictions," often denoted TJ_T. This test statistic is distributed χ^2_{L-k} under the null.

If the optimal weighting matrix is used, the test of model fit is simply TJ_T as noted. If a suboptimal weighting matrix is used, it is necessary to use (15) to find $T\mathbf{m}(\boldsymbol{\theta})'[\text{cov}(\mathbf{m}(\boldsymbol{\theta}))]^{+}\mathbf{m}(\boldsymbol{\theta})$ instead of TJ_T. The notation $[\cdot]^{+}$ indicates a pseudo-inverse, since the covariance matrix is singular. The covariance matrix of the moments also allows one to calculate t-statistics to test individual moments.

2.3.3 Three Classic Tests

The three classic test statistics, likelihood ratio (LR), Lagrange multiplier (LM), and Wald (W) can be implemented using the results of `gmm`. The tests are based on *nesting*, meaning the null hypothesis is a special case of the alternative. For example, in the model

$$y_t = \alpha + \beta_1 x_{1,t} + \beta_2 x_{2,t}$$

the null hypothesis $\beta_2 = 0$ can be tested against the alternative $\beta_2 \neq 0$. All three tests are asymptotically χ^2 with degrees of freedom equal to the number of restrictions. This section briefly discusses the theory behind the tests. Section 5.9 shows how to implement these tests.

Likelihood Ratio Test (LRT)

The LR test estimates the model in both restricted and unrestricted form. The essence of the test is to see how much the loss function (J_T) increases when the restrictions are imposed. The test is implemented by multiplying the difference in the restricted and unrestricted objective functions by the sample size. It is important to estimate the restricted model using the same weighting matrix as the unrestricted model.

Wald Test (W)

The Wald test estimates the unrestricted version of the model only. Conceptually, you see how many standard errors your restriction is from zero, much like a t-test. In a linear model with one restriction, the Wald test is simply a squared t-test. More generally, the test can accomodate multivariate restrictions, possibly involving linear combinations of the variables. These restrictions are of the form $\mathbf{R}\mathbf{b} = \mathbf{r}$, where $\mathbf{R}$ is $q \times k$ and $\mathbf{r}$ is a q-vector with

q denoting the number of restrictions and k denoting the number of parameters. Denote the variance of $\mathbf{b} = [\alpha\beta_1\beta_2]'$ as $\mathbf{\Sigma}$, so

$$\text{var}(\mathbf{Rb} - \mathbf{r}) = \mathbf{R\Sigma R}'$$

The test statistic is

$$W = (\mathbf{Rb} - \mathbf{r})'(\mathbf{R\Sigma R}')^{-1}(\mathbf{Rb} - \mathbf{r})$$

In the above example, $\mathbf{R} = [0\ 0\ 1]$ and $\mathbf{r} = 0$ so the test becomes $\hat{\beta}_2^2/\widehat{\text{var}}(\beta_2)$, a t^2 as claimed. To test if $\beta_1 = 0$ and $\beta_2 = 1$, set $\mathbf{R} = [0\ 1\ 0;\ 0\ 0\ 1]$ and $\mathbf{r} = [0;\ 1]$. To test $\beta_1 + \beta_2 = 0$ set $\mathbf{R} = [0\ 1\ 1]$ and $\mathbf{r}$ to zero.

A caution is in order when using the Wald test for nonlinear models. The test is not invariant to the paramterization of the model. You can get different inferences by simply changing the way you write the model. This lack of invariance means you should probably not use the test in the nonlinear case.

Lagrange Multiplier Test (LM)

The LM test estimates the model in its restricted form, then checks to see how well the unrestricted model fits the data *at the restricted parameter values*. This approach can be convenient when the full unrestricted model is difficult to estimate.

Implementation using the GMM package is somewhat more complicated than with the other tests because it is necessary to take the restricted estimates and reevaluate the moment conditions and Jacobian for the full model at the restricted values. The test statistic is $LM = T(\mathbf{m}'\mathbf{WM})(\mathbf{M}'\mathbf{WM})^{-1}(\mathbf{M}'\mathbf{Wm})$.

3 The GMM Package

This section covers the particulars of the GMM code. I start this section by looking at the simple OLS example from Section 2.1. This example is very simple and shows the minimum amount of information the user must provide. I show how to formulate this problem in terms of MATLAB code. Section 3.2 then discusses the code in more detail.

3.1 Implementation Example

The model and data are

$$\begin{bmatrix}1\\3\\5\end{bmatrix} = \begin{bmatrix}1\\1\\1\end{bmatrix}\alpha + \begin{bmatrix}0\\1\\2\end{bmatrix}\beta + \begin{bmatrix}e_1\\e_2\\e_3\end{bmatrix}$$

We need to define the moment conditions to use. Previously, we decided to focus on the OLS normal equations, meaning we can use $\mathbf{Z} = \mathbf{X}$ (which includes the vector of ones here) and we want $\mathbf{X}'\mathbf{e} = \mathbf{0}$.

```
y = [1; 3; 5];
X = [1 0; 1 1; 1 2];
b0 = [0; 0];
gmmopt.infoz.momt='lingmmm';
gmmopt.gmmit = 1;
out = gmm(b0,gmmopt,y,X,X);
```

The first three lines just set up the data and the parameter starting values. The fourth line defines the m-file for the moment conditions, `lingmmm.m` (note the file extension is suppressed). The fifth line says to use one-step GMM estimation (for simplicity). The last line calls the `gmm()` function, passing the parameter starting values, the `gmmopt` variable containing the reference to our moment conditions, the "dependent" variable $\mathbf{y}$, the "independent" variable(s) $\mathbf{X}$, and the instruments $\mathbf{Z}$.

An edited version of the output is

```
------------------  GMM PARAMETER ESTIMATES  -----------------
Parameter          Coeff     Std Err    Null   t-stat     p-val
parameter 1    1.000000   0.000000    0.00
parameter 2    2.000000   0.000000    0.00
```

Focusing only on the parameter estimates for now, we see that indeed, $\hat{\alpha} = 1.0$ and $\hat{\beta} = 2.0$. Note that this example is a little unusual since the error terms are identically zero so the standard errors are zero.

Before proceeding, lets take a look at the moment conditions in the file lingmm.m The file (modified somewhat to facilitate exposition for the single equation case) is

```
function [m,e] = lingmmm(b,infoz,stat,y,x,z)
e = y - x*b;
m = z'*e/rows(e);
```

The first line defines the function and its input/output arguments. The next line calculates the model residuals, like (3). The last line forms the moment conditions for the normal equations, like (4). Although this sample code is very simple, it illustrates all of the necessary features of the moment conditions m-file.

3.2 GMM Package Details

The GMM package is comprised of several m-files, including an optimization library MINZ. The user typically does not need to work directly with most of these files, but they are described for completeness. The user will need to create his own m-file specifying the model. With this m-file in hand, the GMM estimation is as simple as `gmm(b0,gmmopt,Y,X,Z,Win)`. A vector of parameter starting values is given in `b0`. The `gmmopt` argument is a MATLAB structure variable.[3] The structure `gmmopt` has as its fields several variables specific to GMM,

[3]This is a special type of construct that allows packaging of variables, both numeric and string, of differing dimensions as fields in a single object. A field is referenced by `gmmopt.fieldname`. Do a `help struct` in MATLAB for more information.

but also another entire structure, `infoz`, which contains much of the information needed by the optimization routine. The GMM-specific portions of `gmmopt` are discussed in this section. See Section 4 for more details on the optimization options `gmmopt.infoz`.

The arguments `Y`, `X`, and `Z` contain the data from which the model is estimated. The distinction between `Y` and `X` may be somewhat artificial in some situations, but should have a natural interpretation in regression settings. It is important to put the instruments in `Z`. For OLS problems, just repeat `X` for the `Z` argument. Each of the three data inputs should have the same number of observations (rows). You may use different numbers of columns as needed, and in some cases may find it convenient to make use of the multi-dimensional features in MATLAB. For example, I typically accomodate a system of equations by using columns of `Y`, and have also stacked matrices into 3-dimensional objects to handle lagged variables in `X`. However you decide to organize your data, it must be consistent with the m-file you provide for the moment conditions. Finally, `Win` is an optional argument for a user-defined matrix for use as the initial weighting matrix. This is only used if you do not want to use an "optimal" weighting matrix.

The function `gmm` returns two structures, `gmmout` and `gmmopt`. The former provides a collection of the results from the estimation. The latter is an update of the `gmmopt` structure provided on input. You can accept this output structure with some other name to avoid overwriting your original `gmmopt`.

The files comprising the GMM library are described below. Most features of the software are described, although the user need not be concerned with much of the detail for most applications. The key thing to remember is that the user must provide the file with the moment conditions of interest, and tell the software the name of this file (without the `.m` extension) in `gmmopt.infoz.momt`. If the user has a linear model, the user can reference the m-file `lingmmm` which is provided as part of the library.

A sample of the output is shown in Figure 2. The default is to print a report with parameter estimates, standard errors, t-statistics, and p-values. The user can give the routine a vector of values for the null hypothesis of interest if values other than zero are desired. For over-identified models, similar information about the moment conditions is printed, along with the χ^2 test for the over-identifying restrictions. A heading section describes the estimation problem. In between the heading and the results there is some information about the optimization process.

Provided files in the GMM package

1. `gmm`

   The part of the online documentation for `gmm` describing the `gmmopt` structure appears below. Note that the `.infoz.momt` field is the only one required. This is where you specify the file containing your moment conditions. Defaults for others appear in brackets. `.infoz.jake` is the m-file containg the Jacobian of the moments, $\mathbf{M}(\boldsymbol{\theta})$. If left blank then numerical derivatives are used. `.infoz.hess` controls the Hessian calculation in the optimization routine. More details on it are provided on page 21.

Figure 2: Sample GMM Output

```
============================================================
                    GMM ESTIMATION PROGRAM
============================================================

2 Parameters, 8 Moment Conditions
2 Equation Model, 4 Instruments
329 Observations
2 Passes, Max., 100 Iterations/Pass
Search Direction:         BFGS
Derivatives:              Analytical (gmmexj)
Initial Weighting Matrix: inv(Z'Z)
Weighting Matrix:         Optimal
Spectral Density Matrix:  Newey-West (12 lags)


                   STARTING GMM ITERATION  1
  Weights Attached to Moments
         Moment 1   Moment 2   Moment 3   Moment 4   Moment 5   Moment 6
beta      -0.7716     0.5981     0.0063     0.6678    -0.1722     0.6539
gamma     29.1716   -43.2613     1.4210    13.1716    30.1899   -43.0929

        Moment 7   Moment 8
beta     -0.0219     0.0395
gamma     1.3827    12.0175

Ill-Conditioning Tolerance Set to 1000
Parameter Convergence Tolerance Set to 1e-4
Objective Function Convergence Tolerance Set to 1e-7
Gradient Convergence Tolerance Set to 1e-7
INITIAL HESSIAN = I
------------------------------------------------------------

        ITER      cond(H)  *    Step                 Obj Fcn
  1       1.00e+00    1.000000     0.0000056255
  2       9.16e+00    1.000000     0.0000002832
  3       9.16e+00    1.000000     0.0000002832
  CONVERGENCE CRITERIA MET: Change in Objective Function
         beta    gamma
 b1    1.0103   4.9999

                   STARTING GMM ITERATION  2
  Weights Attached to Moments
         Moment 1   Moment 2   Moment 3   Moment 4   Moment 5   Moment 6
beta      -1.5903    -3.1187    -1.0759     5.9184     0.6884    16.2865
gamma    -15.0070    -5.4496    -0.4973    21.0482    29.6968   -28.5222

        Moment 7   Moment 8
beta     11.4508   -27.5593
gamma    11.4646   -11.7335


------------------------------------------------------------

        ITER      cond(H)  *    Step                 Obj Fcn
  1       1.00e+00    0.000300     0.0618081047
  2       7.86e+01    1.000000     0.0562342258
[lines cut for brevity]
 10       1.28e+03 *  1.000000     0.0282889094
 11       1.27e+03 *  1.000000     0.0282889094
  CONVERGENCE CRITERIA MET: Change in Objective Function


           EVALUATING S at FINAL PARAMETER ESTIMATES

  ------------------  GMM PARAMETER ESTIMATES  ------------------
  Parameter       Coeff    Std Err    Null   t-stat     p-val
  beta         1.001973   0.001539    1.00     1.28    0.1998
  gamma        1.245958   0.511788    0.00     2.43    0.0149

  ------------------  GMM MOMENT CONDITIONS  -------------------
                    Moment    Std Err   t-stat     p-val
     Moment 1     0.003338   0.002118     1.58    0.1151
     Moment 2     0.003356   0.002122     1.58    0.1137
     Moment 3     0.003465   0.002118     1.64    0.1019
     Moment 4     0.003356   0.002120     1.58    0.1134
     Moment 5    -0.000334   0.000375    -0.89    0.3723
     Moment 6    -0.000330   0.000377    -0.88    0.3814
     Moment 7    -0.000339   0.000369    -0.92    0.3577
     Moment 8    -0.000331   0.000376    -0.88    0.3785

      J-stat = 9.3071    Prob[Chi-sq.(6) > J] = 0.1570
============================================================
```

```
% gmmopt  structure of gmm options                                    [default]
%   gmmopt.infoz    Nested structure of infoz needed in MINZ
%   gmmopt.infoz.momt  Filename of moment conditions                 REQUIRED
%   gmmopt.infoz.jake  Filename of Jacobian of moment cond           ['numz']
%   gmmopt.infoz.hess  Hessian updating (see gmmS function)          ['gn']
%
%   gmmopt.gmmit  Number of GMM iterations (NaN is iterated)         [2]
%   gmmopt.maxit  Cap on number of GMM iterations                    [25]
%   gmmopt.tol    Convergence criteria for iter. GMM                 [1e-7]
%   gmmopt.W0     Initial GMM weighting matrix                       ['Z']
%       'I' = Identity, 'Z' = Instruments (Z'Z), 'C' = Calculate from b,
%       'Win' = Fixed passed as Win, myfile = user's own m-file
%   gmmopt.W      Subsequent GMM Weighting Matrix                    ['S']
%       'S' = inverse Spectral Density from gmmS
%       myfile = user's m-file name
%   gmmopt.S      Type of Spectral Density matrix                    ['NW']
%       'W'=White, 'NW'=Newey-West (Bartlett), 'G'=Gallant (Parzen)
%       'H'=Hansen (Truncated), 'AM'=Andrews-Monahan, 'P'=Plain (OLS)
%       myfile = user's m-file
%   gmmopt.aminfo structure if gmmopt.S='AM'.   See ANDMON.M
%   gmmopt.lags   Lags used in truncated kernel for S                [nobs^(1/3)]
%   gmmopt.wtvec  User-provided vector of wts for truncated kernel
%   gmmopt.Strim  Contols demeaning of moments in calc of S          [1]
%       0 = none, 1 = demean e, 2 = demean Z'e
%   gmmopt.Slast  1 to recalc S at final param est. 2 updates W      [1]
%   gmmopt.null   Vector of null hypotheses for t-stats              [0]
%   gmmopt.prt    Fid for printing (0=none,1=screen,else file)       [1]
%   gmmopt.plot   1 does some plots, else suppress                   [1]
%   gmmopt.vname  Optional k-vector of parameter names
```

The `.gmmit` option determines the number of iterations through the GMM procedure. Note that the number of GMM iterations is different from the number of iterations in the optimization (`.infoz.maxit`). Iterated GMM is achieved by setting `.gmmit=NaN`. The user controls the maximum number of iterations and the convergence criteria for the objective function with `.maxit` and `.tol`.

On each iteration a new weighting matrix is calculated. The initial weighting matrix is determined by setting `.W0`; the default is to use the instruments, $(\mathbf{I}_N \otimes \mathbf{Z}'\mathbf{Z})^{-1}$. The user can also choose the Identity matrix (`gmmopt.W0='I'`) or to calculate a weighting matrix from the starting parameter values (`'C'`). Alternatively, the user can specify their own weighting matrix. To use a fixed matrix of numbers as the initial weighting matrix, set `gmmopt.W0='Win'` and pass the desired matrix as the last argument to the `gmm()` function. To use a matrix calculated by an m-file as the initial weighting matrix, pass the name of this file (without the `.m` extension). For example, to user the file `userw.m`, set `gmmopt.W0='userw'`. The user can also control the weighting matrix used in subsequent iterations. The default (`gmmopt.W='S'`) is to use the inverse of the spectral density matrix calculated by the `gmmS` function. Alternatively, the user can specify a different function to calculate a weighting matrix, (e.g., `gmmopt.W='userw'`). This can be, but does not have to be, the same function as the initial weighting

matrix. If a user's own m-file is provided, the input and output arguments must follow the convention `W = userw(b,gmmopt,Y,X,Z)`. Of course the name of the m-file does not have to be `userw.m`, but must match the name specified in `gmmopt.W0` and/or `.W`.

If using an optimal weighting matrix, the user can specify the desired method for estimating the Spectral Density matrix in `gmmopt.S`. The options are Identity (`'I'`), White (`'W'`), Hansen (`'H'`, truncated kernel), Newey-West (`'NW'`, Bartlett kernel), Gallant (`'G'`, Parzen kernel), or the Andrews-Monahan procedure (`'AM'`). If the covariance matrix is not positive definite[4] an error flag is returned. In the case of Newey-West, Gallant, or Hansen, the number of lags is specified in `.lags`. The default is $T^{1/3}$. One additional feature that applies to the truncated kernel is the ability to select specific lags in the spectral density matrix. To do so, the user passes a vector of the weights associated with each lag to `gmmopt.wtvec`, starting with lag 1 (the contemporaneous, or lag 0, is assumed to get a weight of 1). This can be useful for modeling seasonalities. For example, if there is a quarterly seasonal in monthly data, a user might use `gmmopt.wtvec = [0 0 1]'`. Keep in mind that $\mathbf{S}$ computed in this fashion may not be positive definite. Another feature the user can control is demeaning of the moment conditions in calculation of $\mathbf{S}$. Under the null, $E[\mathbf{e}_t] = 0$ and $E[\mathbf{e}_t \otimes \mathbf{z}_t] = 0$ so it should not matter whether one uses $E[\mathbf{m}_t\mathbf{m}_t']$ or $E[\mathbf{m}_t\mathbf{m}_t'] - E[\mathbf{m}_t]E[\mathbf{m}_t']$. In practice, one may wish to allow for non-zeros means. Accordingly, the field `.Strim` makes no correction if zero, demeans $\mathbf{e}_t$ if set to one (default), or demeans $\mathbf{e}_t \otimes \mathbf{z}_t$ if set to two.

The Andrews-Monahan method has some additional options associated with it in the `gmmopt.aminfo` structure. Specifically, the user can choose to the time series model used to prewhiten the residuals: AR(1) equation-by-equation (`.aminfo.p=1`), MA(q) for each equation (set `.aminfo.q` to q), or an ARMA(1,1) model by making the obvious settings to the `.p` and `.q` fields. Setting `.aminfo.vardum=1` overrides the equation-by-equation settings and estimates a VAR(1) model. The Andrews-Monahan procedure "automatically" calculates the bandwidth[5] and uses either the Quadratic-Spectral kernel as the default (`.aminfo.kernel='QS'`) or the Tukey-Hanning kernel (`.aminfo.kernel='TH'`). A modification of this method [Andrews (1991)] uses the time series models to determine the bandwidth but does not use the pre-whitened residuals. This procedure is achieved by setting `.aminfo.nowhite=1`.

The final parameter estimates are used to recalculate the spectral density matrix estimate if the field `gmmopt.Slast` is set to one (default). Thus, the $\mathbf{S}$ is the formulas in

[4]This is possible with the truncated kernel. A common solution is to switch to a method such as Newey-West when this occurs. The criteria for positive definiteness is $\lambda_{\min} > \varepsilon\lambda_{\max}\text{cols}(\mathbf{S})$, where λ_i is an eigenvalue of the Spectral Density matrix $\mathbf{S}$ and ε is the machine precision. This is the same criteria as used by the MATLAB function `rank()`.

[5]The procedure is automatic in that it uses specific formulas for the calculation. There are still some decisions the analyst must make, such as the specification of the time series model. The bandwith is a function of the weighted value of the parameters of the time series model. The weights are set to unity for coefficients other than the intercept in Andrews (1991) and Andrews and Monahan (1992), resulting in a bandwidth which is sensitive to the scaling of the data. I follow the recommendation of den Haan and Levin (1999) and use the inverse of the standard deviation of each moment as the weight to account for scaling.

Section 2.3 do not collapse to the simplified form since **W** is the inverse of **S** from the *prior* parameter estimates, not the final **S**. The updating of **S** in this fashion is similar to the simple case of using Newey-West standard errors for a least squares problem: the OLS estimates are used to calculate the covariance matrix. To force recalculation of **W**, set `gmmopt.Slast=2`. Any other value will update neither **W** nor **S**.

A few other options in `gmmopt` control some of the fluff. `.null` sets the null hypotheses for parameter values. The default is a vector of zeros. `.vname` is a string matrix of parameter names, one per row. If nothing is given the parameters are simply labeled sequentially as "Parameter k." Finally, printing is controlled via `gmmopt.prt`. Setting it to zero suppresses printing, one prints to the screen, and a higher value prints to a file as specified by `fopen()`. The optimization printing (the sections of Figure 2 with iteration histories) is controlled separately but in a similar fashion by `gmmopt.infoz.prt`. Two diagnostic plots are generated when `gmmopt.plot=1`. One shows the absolute value of the weights in **W**. Negative weights are market with an "X." The second graph shows the kernel weights used in calculating **S**.

```
% gmmout  results structure
%  gmmout.f      function value
%  gmmout.J      chi-square stat for model fit
%  gmmout.p      p-value for model fit
%  gmmout.b      coefficient estimates
%  gmmout.se     standard errors for each parameter
%  gmmout.bcov   cov matrix of parameter estimates
%  gmmout.t      t-stats for parms = null
%  gmmout.pb     p-values for coefficients
%  gmmout.m      moments
%  gmmout.mse    standard errors of moments
%  gmmout.varm   covariance matrix of moments
%  gmmout.mt     t-stats for moments = 0
%  gmmout.mp     p-vals for moments
%  gmmout.nobs   number of observations
%  gmmout.north  number of orthogonality conditions
%  gmmout.neq    number of equations
%  gmmout.nz     number of instruments
%  gmmout.nvar   number of parameters
%  gmmout.df     degrees of freedom for model
%  gmmout.stat   stat structure from MINZ
%  gmmout.null   vector of null hypotheses for parameter values
%  gmmout.W      weighting matrix
%  gmmout.S      spectral density matrix
%  gmmout.eflag  error flag for spectral density matrix
%  gmmout.ithist  History of MINZ iterations
```

The `gmm` function returns the results from the estimation packaged in a structure variable called `gmmout`. These fields should be largely self-explanatory. Of note, the `.stat` structure contains information on the status of the optimization procedure. This structure is discussed in more detail in Section 4. A second argument returned by `gmm` is

the `gmmopt` structure, with any updates during the estimation. A few fields are added to the input structure to facilitate printing.

2. `gmmS`

   This file calculates the Spectral Density matrix **S**. The type of estimate is specified by `gmmopt.S`. The default is a Newey-West matrix (`'NW'`) with $T^{1/3}$ lags. Different lags are specified in `gmmopt.lags`. The preceeding discussion of the `gmm.m` file provides additional details. The output heading indicates which method was used along with the lag structure.

3. `prt_gmm`

   Controls printing of header information, GMM iteration history, and results. This file uses the fields `hprt` and `eprt` (added by `gmm`) to determine whether to print the header summary information or the final parameter estimates.

4. `lsfunc`

   The objective function used by the MINZ optimization routine ($\mathbf{m'Wm}$). `gmm` knows to assign this filename to the `gmmopt.infoz.func` field, so the user doesn't need to worry about this.

5. `lsgrad`

   The gradient used by the minz optimization routine ($2\mathbf{M'Wm}$). Again, this assignment is handled by the program.

6. `lingmmm`

   Moment conditions for use in a linear model. Of note, the user must give the dependent variable(s) `Y`, independent variables `X`, *and* the instruments `Z`. It is fine to use the same data for `Z` as `X`. Instrumental variables style estimation requires using different data. Systems of equations like $\mathbf{y}_i = \mathbf{X}\boldsymbol{\beta}_i + \mathbf{e}_i$ can be handled by setting $\mathbf{Y} = [\mathbf{y}_1 \ldots \mathbf{y}_N]$.

7. `lingmmj`

   Jacobian of moment conditions for use in a linear model.

8. `lingmmh`

   Calculates analytic second derivatives of objective function (Hessian) for linear model. As for all files dealing with the Hessian, the file returns the *inverse* Hessian as the field `stat.Hi`. This is discussed in more detail in Section 4.

9. `msdm`

   Moment conditions to calculate the means and covariance matrix of a dataset. For a $T \times k$ matrix of data $\mathbf{Y}$, takes as parameters k means, followed by vech(cov($\mathbf{y}$)).

10. `msdj`

    Jacobian for `msdm`.

User-defined files needed for GMM

1. Moment Conditions

   The user *must* specify a file containing the moment conditions of interest. This function takes the form `[m,e] = yourfunc(b,infoz,stat,Y,X,Z)`. Of course, you name the function and the m-file it lives in something meaningful to you, not "yourfunc." The function needs to return the moment conditions $\mathbf{m}$ and the model errors $\mathbf{e}$ For a N-equation model, the model errors are a $T \times N$ matrix formed by concatenating columns of the errors in (3). The moment conditions are the L-vector formed by taking the matrix $\mathbf{Z}'\mathbf{e}/T$ and stacking the columns. The inputs to the function are the parameter values `b`, the structures `infoz` and `stat` containing information about the estimation procedure (these need not be used by the function but must be included as arguments), and the data $\mathbf{Y}, \mathbf{X}, \mathbf{Z}$. The file with the moment conditions is referenced by setting `gmmopt.infoz.momt = 'filename'`, or equivalently, by setting `infoz.momt = 'filename'` and then `gmmopt.infoz = infoz`.

   We can look at the provided `lingmm` file as an example. The file starts with the line `function [m,e] = lingmmm(b,infoz,stat,y,x,z)`. This tells MATLAB that the file is a function (as opposed to a script) and the function takes the arguments listed and returns `m` and `e`.

   If your moments require additional data or information, you can pass these through as fields in the `infoz` structure. For example, you may have moment conditions that depend on the value of some constants, which you may like to change occasionally. If your moments need to know such a value μ, you can pass this number to the estimation procedure by setting `gmmopt.infoz.mu=` your desired value. Your moment conditions m-file would then receive this like `mu = infoz.mu;` and use it as if it were a constant hard-coded into your moment conditions. This flexibiliity is nice as it let you write one set of moment conditions and apply it to a variety of problems.

```
function [m,e] = lingmmm(b,infoz,stat,y,x,z)
% PURPOSE:  Provide moment conditions and error term for
%             linear GMM estimation
%---------------------------------------------------------
% USAGE:  [m,e] = lingmmm(b,infoz,stat,y,x,z,w)
%  b       model parameters
%  infoz    MINZ infoz structure
%  stat    MINZ status structure
%  y,x,z   Data: dependent, independent, and instruments
%  w       GMM weighting matrix
%---------------------------------------------------------
% RETURNS:
%  m       vector of moment conditions
%  e       Model errors  (Nobs x Neq)
%---------------------------------------------------------
% VERSION: 1.1.2

% written by:
% Mike Cliff,  Purdue Finance  mcliff@mgmt.purdue.edu
% Created: 12/10/98
% Modified 9/26/00 (1.1.1 Does system of Eqs)
%          11//13/00 (1.1.2 No W as input argument)

k = rows(b);
nx = cols(x);
neq = k/nx;
e = [];
if mod(k,nx) ~= 0
  error('Problem determining number of equations')
end

for i = 1:neq
  ei = y(:,i) - x*b((i-1)*nx+1:i*nx);
  e = [e ei];
end

m = vec(z'*e/rows(e));
```

2. Jacobian (Optional)

   As discussed above, it is more convenient to work with the derivative of the moment conditions than the derivative of the objective function. I refer to the former as the Jacobian and the latter as the gradient. This function takes the same arguments as the moment condition file and returns the $L \times K$ matrix of derivatives of each moment condition with respect to the parameters. The file with the Jacobian is referenced in a manner similar to moment conditions, although the field is now `jake`. Although analytic derivatives should always be used whenever possible, numerical derivatives are available by leaving the field `jake` empty or by setting it to `numz`.

3. User's Weighting Matrix (Optional)

The user can elect to use a sub-optimal weighting matrix. This weighting matrix can either be fixed or calculated as a function of the data and parameters. For a fixed initial matrix, the user passes the matrix as the last argument to `gmm()` and sets `gmmopt.W0='Win'`. The matrix must be positive definite with dimensions corresponding to the number of moment conditions. For a calculated weighting matrix, pass the name of the m-file that calculates your **W** to `gmmopt.W0` and/or `gmmopt.W`. As with other file references such as `.momt`, you specify the file name without its extension and the file must be in your path.

As a simple example, you might want to use a weighting matrix which just has diagonal elements. If this function is in the m-file `userw.m` you would put `gmmopt.W='userw'` to invoke this weighting matrix after the first GMM iteration. To also use it as the initial weighting matrix, set `gmmopt.W0='userw'` as well. The code would may be something like the following, but must have input and output arguments as show here.

```
function W = userw(b,gmmopt,Y,X,Z);
%
%function W = userw(b,infoz,Y,X,Z);
%
% Function to calculate user-defined weighting matrix.  Example uses
% inverse of diagonal of cov (e.*Z)

momt = fcnchk(gmmopt.infoz.momt);
[m,e] = feval(momt,b,gmmopt.infoz,[],Y,X,Z);
u = repmat(e,1,cols(Z)).*Z;
S = diag(diag(cov(u)));
W = S\eye(cols(S));
```

If you have your own spectral density matrix m-file called `mys.m` and want to use its inverse as your weighting matrix, then make a m-file that does this

```
function W = mysinv(b,gmmopt,Y,X,Z)
W = inv(mys(b,gmmopt,Y,X,Z));
```

You would need to set `gmmopt.S='mys'` and `gmmopt.W='mysinv'`.

4 Optimization Library

Users solely interested in GMM may not need to be concerned with the details of the optimizer. However, an understanding of the process is useful in problem-solving and enhancing performance. Additionally, the user may wish to modify or enhance the optimization routines employed.

The MINZ optimization library provides a flexible environment for general optimization problems. Although many of the algorithms employed are "good" [typically drawn from Gill, Murray, and Wright (1981) or Press, Teukolsky, Vetterling, and Flannery (1992)], they may

not always be "best" for the situation at hand. The overall flexibility of the software makes it relatively easy for the user to substitute their own algorithms.

The focus is on applications where parameter estimation and hypothesis testing are of primary interest. Econometrics is of course one such field. The library consists of a "control" program `minz` and a number of helper functions which are called by `minz`. The user issues a command such as `[b,infoz,stat] = minz(b0,infoz,Y,X)`. `b0` represents the parameter starting values, `infoz` is the structure variable containing information about the specific procedures used, convergence criteria, etc., and `Y` and `X` represent data. The number of arguments given to `minz` is variable depending on the needs of the objective function of interest, so additional data/variables can be passed. The important thing is that the input arguments must be consistent with the function specified in `infoz.func`. The routine returns the parameter estimates `b`, the `infoz` structure (with any updates) and `stat` which contains a number of fields with information on the status of the solution (e.g., the inverse Hessian). We will examine the components of the `infoz` and `stat` structures then discuss each of the supporting functions.

The `infoz` structure contains the following fields

```
% infoz structure
%   infoz.call    Calling program: 'gmm', 'ls', 'mle', 'other'
%   infoz.func    What to min: 'lsfunc' for LS/GMM
%   infoz.momt    Orthog. conditions m of m'Wm for GMM
%   infoz.jake    Jacobian of momt
%   infoz.grad    Gradient: 'gradfile' for analytic, else        ['numz']
%   infoz.delta   Increment in numerical derivs                 [.000001]
%   infoz.hess    Hessian: ['dfp'], 'bfgs', 'gn', 'marq', 'sd'
%   infoz.H1      Initial Hessian in DFP/BFGS.  [1] = eye, else evaluate
%   infoz.maxit   Maximium iterations                             [100]
%   infoz.step    step size routine                            ['step2']
%   infoz.lambda  Minimum eigenvalue of Hessian for Marquardt     [.001]
%   infoz.cond    Tolerance level for condition of Hessian        [1000]
%   infoz.btol    Tolerance for convergence of parm vector        [1e-4]
%   infoz.ftol    Tolerance for convergence of objective function [1e-7]
%   infoz.gtol    Tolerance for convergence of gradient           [1e-7]
%   infoz.prt     Printing: 0 = None, 1 = Screen, higher = file   [1]
```

The user *must* provide `infoz.func` or, in the case of GMM/LS, `infoz.momt`. All other fields are optional and use the default values (in brackets) unless specified otherwise. If the user would like analytic derivatives, these are specified in `infoz.jake` for GMM/LS, and `infoz.grad` for other problems. If these are left blank, or if they are set to `numz`, then numerical derivatives are used. Next are a number of controls for the (approximate) Hessian, including `infoz.hess`, `infoz.H1`, `infoz.lambda` and `infoz.cond`. These are discussed in detail in the Hessian step below. The user may also specify a file for the step size in `infoz.step` and its option `infoz.stepred`. See the step size description below for more details. Next are a series of settings for convergence criteria. Iterations will stop when any one of the four criteria are met: maximum iterations (`infoz.maxit`), change in parameter values (`infoz.btol`), change in the objective function (`infoz.ftol`), or change in the gra-

dient (`infoz.gtol`). The procedure will tell you which criteria caused the program to stop. `infoz.prt` controls printing during the optimization procedure and `infoz.call` tells the optimization program the kind of problem it is solving: GMM/LS, or other (the typical user does not need to worry about this but it is the mechanism that lets `minz` know when to use the `momt` and `jake` fields).

Steps in Optimization Library

1. Evaluate Criterion Function given in `infoz.func`

   The `func` field specifies the file with the objective function. This file can require any number of inputs but must have the form `scalar = myfunc(b,infoz,stat,varargin)`. The `infoz` and `stat` structures need not be used by the function but they must be included as arguments. In the case of least squares and GMM problems, the code sets `infoz.func = 'lsfunc'` to use a standard file. The user then provides a moment conditions file of the form `[m,e] = myfunc(b,infoz,stat,varargin)`.

2. Evaluate Gradient in `infoz.grad`

   In all cases, gradient refers to the first derivative of the objective function with respect to the parameter vector. For GMM/LS, it is more convenient to work with the Jacobian: the first derivative of the moment conditions or model error.

   Handling of the gradient works much like the objective function. For GMM/LS problems, a default gradient file `lsgrad` is used but the user can specify the Jacobian in `infoz.jake`. For all other problems, the user can set `infoz.grad` for analytic derivatives. If the user wants numerical derivatives, he can leave these fields blank or set the appropriate field to `numz`. Again, for GMM/LS the field to set is `jake` whereas in other problems the relevant field is `grad`.

3. Calculate/update Hessian in `infoz.hess` (e.g., `hessz`)

   To choose the search direction method, set `infoz.hess` to `'dfp'` for Davidon-Fletcher-Powell, `'bfgs'` for Broyden-Fletcher-Goldfarb-Shanno, `'gn'` for Gauss-Newton, `'marq'` for Levenberg-Marquardt, or `'sd'` for Steepest Descent. Any one of these choices will result in a call to the file `hessz`. A user can substitute a different method be setting `infoz.hess`. The procedure for adding a new method will be discussed shortly. Note that `hessz` actually returns the inverse Hessian, since this is the object of interest in both the optimization routine and in hypothesis testing. For the DFP/BFGS methods, the inverse is calculated directly. The other methods calculate the normal Hessian then invert it (actually use Gaussian elimiation using the MATLAB \ operator). There are a number of options that can be passed to `hessz`. To specify the initial Hessian to use in the DFP/BFGS algorithms, set `infoz.H0 = 1` for the identity matrix. In the Levenberg-Marquardt alogorithm the user can specify the value added to the diagonal of the Hessian when it is ill-conditioned. This is done by setting `infoz.lambda = value`. Finally, the user can set the criteria for determining ill-conditioning with `infoz.cond`.

4. Determine Step Direction

   The step direction is calculated within `minz` as the product of the inverse Hessian and the gradient.

5. Determine Step Size specified in `infoz.step` (e.g,. `step2`)

   Once a step direction is calculated, the program must determine how far to move in this direction. The default is to use the file `step2` for the line minimization. This file uses a simple algorithm which reduces the step size by a fixed fraction (`infoz.stepred`, 90% by default) until the objective function decreases. A different file is referenced by setting `infoz.step` to the appropriate filename. An alternative routine, based on a the LNSRCH algorithm in Press, Teukolsky, Vetterling, and Flannery (1992, page 378), is provided in `stepz.m` but I have had difficulty with this algorithm in some cases.

6. Program Control `minz`

   After performing the preceeding steps, `minz` then recalculates the objective function and checks the convergence criteria. If none of the convergence criteria are met, repeat the steps. At each iteration some summary information is printed, if desired. As shown in Figure 2, the output contains the iteration number, the condition number of the Hessian in cond(H), an asterisk if the Hessian is poorly conditioned (`infoz.cond`, 1000 by default), and the value of the objective function. The output contains a message about the reason the iterations stopped (e.g., `CONVERGENCE CRITERIA MET: Change in Objective Function`). For GMM estimation, the parameter estimates at intermediate GMM iterations (not optimization iterations) is also displayed to track the progression of the estimation process.

Each of the steps above is essentially self-contained. Although one function may make calls to another, the structure is such that it is easy to add a new function. In general, these helper functions take the form of `helper(b,infoz,stat,varargin)`. The first argument is always the parameter vector, the second the `infoz` structure, the third is the `stat` structure, and the final arguments are the inputs needed by the user's objective function. So long as this framework is preserved, adding a user's helper function should be simple.

The following is a brief description of how to incorporate a user's function rather than a provided one. First, the user needs to write the m-file for the helper function. Give it a name that will not conflict with existing m-files (e.g., don't call a new Hessian algorithm `hess.m`). Make sure the function returns the same thing as the helper function it replaces. A Hessian algorithm should return the `stat` structure variable with the new inverse Hessian, a step size algorithm returns a scalar step size, etc. Then just change the value in the appropriate `infoz` field to point to your function. For example, a new Hessian algorithm called `greathess.m` requires setting `infoz.hess = 'greathess'`. If a user has an existing function which has different arguments, I suggest writing a short conversion function. This new function would have the input and output argument structure required here, make the necessary conversions, and call the user's existing function.

As `minz` is iterating the above steps it passes information on the status of the procedure to the different functions it calls. This information is conveniently stored in `infoz.stat`. Upon completion of the procedure, this structure variable is returned to the program calling `minz`. Of particular interest is the inverse Hessian, since it is useful for hypothesis testing. The fields are fairly self-explanatory.

```
% stat structure
%   stat.G       Gradient
%   stat.f       Objective function value
%   stat.Hi      Inverse Hessian
%   stat.dG      Change in Gradient
%   stat.db      Change in Parm vector
%   stat.df      Change in Objective Function
%   stat.Hcond   Condition number for current Hessian
%   stat.hist    History of b at each iteration
```

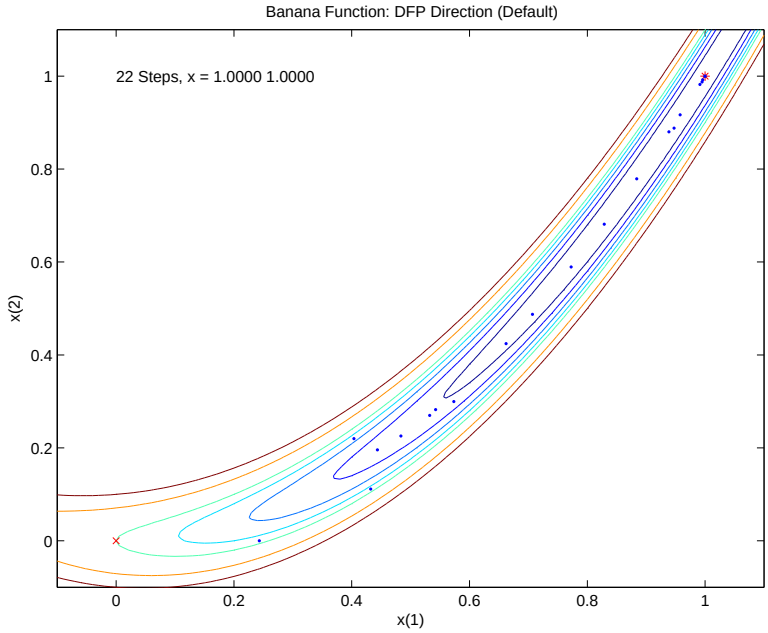
5 Demo Programs

The libraries come with several demonstration programs to illustrate use of the software. I recommend running each of these and looking at the demo program code to see how to formulate and solve problems. I start with a couple of examples using only the MINZ code then provide seven examples using GMM.

1. `rosen_d` Minimize the "banana" function. MINZ only.
2. `nslq_d` Do a nonlinear least squares estimation. MINZ only.
3. `sumstats_d` Calculate summary statistics with GMM.
4. `lingmm_d` Estimate a linear regression model with GMM.
5. `gmm_d` Do a nonlinear GMM estimation; power utility asset pricing model.
6. `ckls_d` Estimate several term structure models. Also shows how to write very flexible moment condition so that you can estimate a model with some of the parameters fixed.
7. `gmmldv_d` Limited dependent variable models (Logit and Probit) by GMM.
8. `hyptest_d` Hypothesis testing examples with GMM.
9. `userw_d` Use a user's weighting matrix in GMM estimation.

To run a demo just type the demo name at the command prompt. The demos provide prompts to guide you through. It will probably be most useful if you also look at the code for each demo to see exactly how things are done.

Figure 3: Iterations of Minimizing Banana Function



5.1 Minimize the Banana Function

To run this demo type `rosen_d` at the MATLAB prompt. The program will draw a couple of pictures of the function then do several different minimizations. Each minimization illustrates the use of a different approximate Hessian. A contour plot of the function is marked with the location of the parameter vector at each step, giving a sense as to the performance of the various algorithms. Figure 3 shows an example. Most of the algorithms perform well, but the steepest descent method fails to find the minimum. Keep in mind that the "best" algorithm depends on the problem at hand.

Look at the files `rosen.m`, `roseng.m`, and `rosenh.m` to get a feel for how the objective function, gradient, and analytic Hessian are specified. Notice in the demo how the Hessian algorithms are changed (e.g., `infoz.hess='bfgs';`).

5.2 Nonlinear Least Squares

Running `nlsq_d` executes a demo of nonlinear least squares. The model is

$$y_t = \beta_1 x_t + \beta_2 x_t^2 + \beta_3 x_t^3 + \beta_4 \cos(\beta_5 x_t) + \varepsilon_t.$$

Note that this model is linear except for the last term. In order to get starting values, the model is estimated by OLS ignoring the final term. A plot then shows the errors for a linear model with a reference cos function. By adjusting the amplitude β_4 and frequency β_5, the augmented OLS predictions are close to the function. These values are then the starting

values in the minimization. The user is then prompted for the choice of approximate Hessian. The final estimates and standard errors are printed and the predictions from the nonlinear and augmented linear models are plotted with the actual values. The demo uses the provided objective function `lsfunc` and gradient `lsgrad` along with the application-specific model errors `nlsqe.m` and Jacobian of errors `nlsqj.m`.

5.3 Estimate Summary Statistics

Now we move from demonstrations of the MINZ package alone to ones also featuring the GMM library. The first example is very simple, estimate the means, variances, and covariances for a matrix of data. The demo is provided as the file `sumstats_d` and is quite short. The basic code is shown below

```
T = 1000;
k = 3;
Y = randn(T,k);
gmmopt.infoz.momt='msdm';
gmmopt.infoz.jake='msdj'
bin = zeros(k + k*(k+1)/2,1);
out = gmm(bin,gmmopt,Y,[ ],ones(T,1));
```

The first three lines simply generate some data. The next two lines tell `gmm` what moment conditions and Jacobian to use. If the Jacobian were not specified the code would automatically use numerical derivatives. We then set up parameter starting values as a vector of zeros and call `gmm`. Not much to it!

The output shows that the parameter estimates calculated by GMM are nearly identical to the conventional estimates. There are two reasons for differences. First, we are doing two-stage GMM, so if the first stage spectral density matrix has important off-diagonal terms, we may end up paying attention to the cross-moments and this may affect the estimates. That turns out not to be the case here. The second reason is simply that the `msdm` function does not do a degrees of freedom correction. Modifying it to do so is straightforward.

5.4 Estimate a Linear Model by GMM

The linear model demo `lingmm_d` is geared to giving a flavor of GMM implementation and also facilitates comparison to more tradional estimation methods. The details appearing on the screen show the steps necessary to run `gmm` in this example. The demo makes use of the moment conditions (`lingmmm`), Jacobian (`lingmmj`), and analytic Hessian (`lingmmh`) already provided with the GMM package. Thus, the user does not need to write their own m-files for a basic linear model.

The GMM results are compared with OLS using White standard errors. The small difference in the standard errors arises because the White version does not include the degrees of freedom correction. You can change the covariance matrix calculations in GMM by altering `gmmopt.S` to see how the results change.

```
  ------------------  GMM PARAMETER ESTIMATES  -----------------
  Parameter          Coeff    Std Err    Null    t-stat     p-val
  parameter 1     0.019747   0.032344    0.00      0.61    0.5415
  parameter 2     1.052202   0.032646    0.00     32.23    0.0000
  parameter 3    -0.995067   0.031469    0.00    -31.62    0.0000
  ============================================================
White Heteroscedastic Consistent Estimates
------------------------------------------------------------
  Variable      Coefficient   Std Error    t-stat     p-val
  variable 1       0.019747    0.032296      0.61    0.5410
  variable 2       1.052202    0.032597     32.28    0.0000
  variable 3      -0.995067    0.031422    -31.67    0.0000
```

5.5 Estimate a Nonlinear Model by GMM

Run the demo `gmm_d`. It is largely self-explanatory. The demo estimates the power utility asset pricing model. The model is

$$E_t[\beta(C_{t+1}/C_t)^{-\gamma}\mathbf{R}_{t+1}] = \mathbf{1}$$

where C_t is per capita real consumption at time t, $\mathbf{R}_{t+1}$ is the return on a vector of N assets from time t to $t+1$. There are two parameters, β and γ. The expectation is conditional on time t infomation. We use the instruments $\mathbf{z}_t$ to capture this information. In the demo we use lagged returns and consumption growth, along with a constant, in forming $\mathbf{z}$. The moment conditions are therefore defined as

$$\mathbf{m}(\boldsymbol{\theta}) = \sum_{t=1}^{T}[\beta(C_{t+1}/C_t)^{-\gamma}\mathbf{R}_{t+1}] - \mathbf{1}] \otimes \mathbf{z}_t$$

The demo uses two assets and four instruments, so there are eight moment conditions and two parameters.

Sample output is shown in Figure 2. Figure 4 shows the objective function surface created with the `objplot` function. The flat valley in the γ direction confirms the relatively large standard error on that parameter. The minimized value appears as an asterisk. Figure 5 shows the weighting matrix used in the estimation. You can see that most weight is placed on a few moments. Figure 6 shows the weights used in the calculation of $\mathbf{S}$. This is useful to know if you have in mind a particular correlation structure for the moments. If you would like to allow for an annual seasonal (12 lags with montly data), you can see that the current estimation places relatively little weight on this lag.

This is the first example where we have an over-identified model (more moments than parameters). Now the output shows the moments (since they are not all zero), the corresponding t-stats, and the J test of overidentification.

```
  ------------------  GMM PARAMETER ESTIMATES  -----------------
  Parameter        Coeff    Std Err    Null   t-stat     p-val
```

```
beta          1.001973    0.001539     1.00      1.28     0.1998
gamma         1.245958    0.511788     0.00      2.43     0.0149

------------------  GMM MOMENT CONDITIONS  ------------------
                    Moment     Std Err    t-stat      p-val
   Moment 1       0.003338    0.002118      1.58     0.1151
   Moment 2       0.003356    0.002122      1.58     0.1137
   Moment 3       0.003465    0.002118      1.64     0.1019
   Moment 4       0.003356    0.002120      1.58     0.1134
   Moment 5      -0.000334    0.000375     -0.89     0.3723
   Moment 6      -0.000330    0.000377     -0.88     0.3814
   Moment 7      -0.000339    0.000369     -0.92     0.3577
   Moment 8      -0.000331    0.000376     -0.88     0.3785

    J-stat = 9.3071    Prob[Chi-sq.(6) > J] = 0.1570
================================================================
```

Figure 4: GMM Objective Function

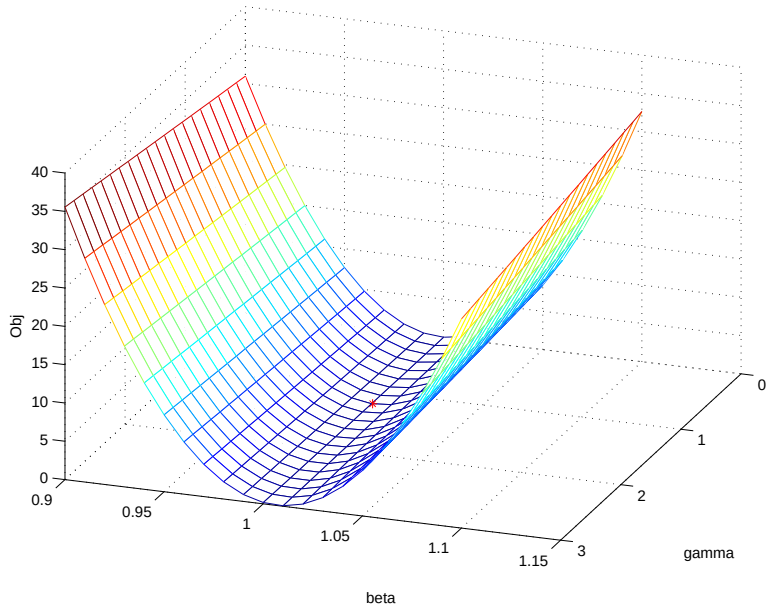


Figure 5: Example of Weighting Matrix

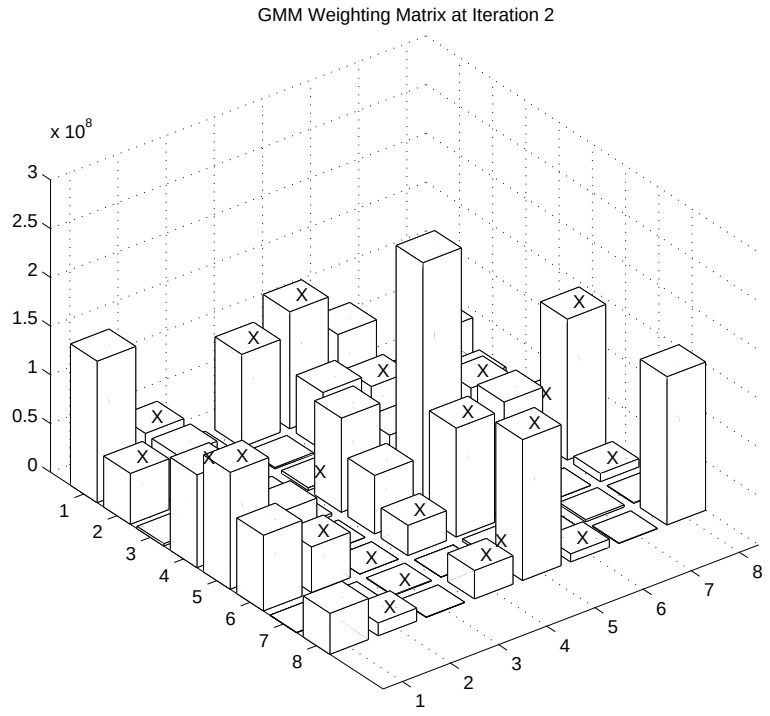
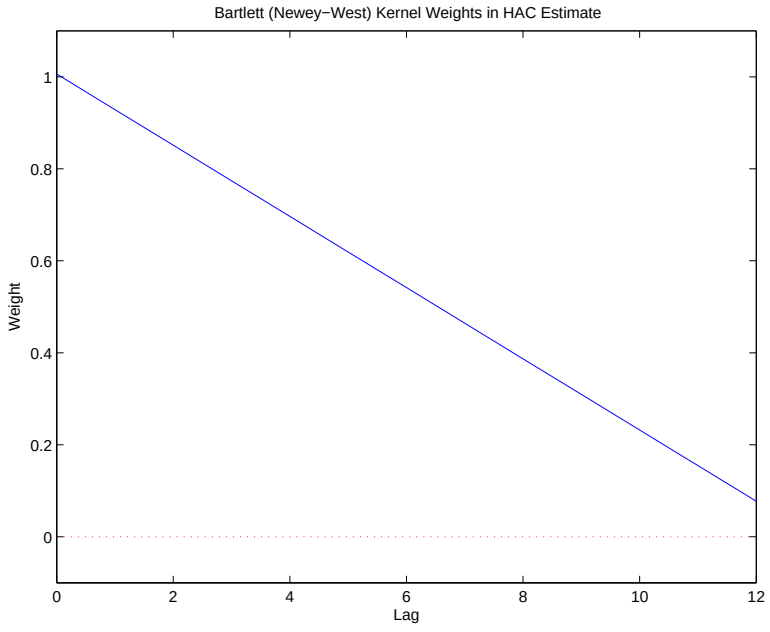


Figure 6: Example of Kernel Weights



Note that this demo is intended to provide an illustration of how to use the code, not be an example of the "best" way to estimate this particular model. Among other things, the choice of assets and instruments are quite important. You can easily adjust the instruments by changing `nz` to add more lags to the instrument set or by excluding consumption from the instrument set (the first column of the rawdata matrix).

5.6 Term Structure Model, CKLS

This is an example of estimating the parameters of the several nested term structure models, as in Chan, Karolyi, Longstaff, and Sanders (1992). The continuous time model is

$$dr_t = \kappa(\theta - r_t)dt + \sigma r_t^{\gamma} dW_t$$

where r_t is the interest rate at time t and W_t is a Brownian motion. This model implies that

$$E[dr_t] = E\left[\kappa(\theta - r_t)\right] dt$$

and

$$\mathrm{Var}[dr_t] = \sigma^2 r_t^{2\gamma} dt$$

A special case when $\gamma = .5$ is Cox, Ingersoll, and Ross (1985)

We have to discretize the model for estimation, so we will use $y_t = dr_t = r_t - r_{t-1}$, $X_t = r_t$, and $dt = 1/12$ since we use monthly observations. We choose instruments $\mathbf{z}_t = [1 \;\; r_t]$, giving moments

$$m_t = \begin{bmatrix} y_t - (\alpha + \beta X_t)/12 \\ (y_t - (\alpha + \beta X_t)/12)^2 - \sigma^2 X_t^{2\gamma}/12 \end{bmatrix} \otimes [1 \;\; X_t]$$

where σ^2 is taken as a parameter (rather than σ). The file `ckls_dm.m` provides the following bit of code (some of the documentation is purged) to define the moment conditions

```
function [m,e]=ckls_dm(b,infoz,stat,y,X,Z)

% --- Grab parameters out of b, Find parameters that are fixed -------
parms = repmat(NaN,4,1);
parms(isnan(infoz.parms)) = b;
parms(~isnan(infoz.parms)) = infoz.parms(~isnan(infoz.parms));
alpha = parms(1);
beta = parms(2);
sigsq = parms(3);
gamma = parms(4);

% --- Form model residuals ------------------------------------------
T = rows(X);
e1 = y - (alpha + beta*X)/12;
e2 = e1.^2 - (sigsq*X.^(2*gamma))/12;
e = [e1 e2];
```

```
% --- Moments are inner product with instruments --------------------
m = reshape(Z'*e/T,4,1);
```

Ignoring the first block of code, the rest just implements the equation for m_t in vector form.

So what is the first block of code for? Well, suppose you want to estimate a restricted version of the model, say where $\gamma = 0.5$ as in Cox, Ingersoll, and Ross (1985). Since that model has only three parameters, we need a way to pass a vectot of starting values with only three rows, and we need to the moment conditions to plug in the value 0.5 for γ. One brute force way to do this is copy our moment conditions file for the full model and make the necessary edits. Well, now suppose we want to have another restricted model that sets $\beta = 0$ and $\gamma = 0$. We now need another file. This is not very flexible, and it will be quite cumbersome to maintain all these files. So how about if we can use only the one file which contains moment conditions for the full model? What we need is a way to tell the file which parameters should be fixed (not estimated) and what values to plug in for them. That is what the mysterious first block of code does.

When we set up the `gmmopt` structure, we will add a field `.infoz.parms`. This field is not recognized by the standard code, but we can have our file `ckls_dm` look for it. The field contains four elements, correpsonding to $[\alpha\beta\sigma^2\gamma]'$. For parameters that are to be estimated, set the element to `NaN`, and for those that are to be fixed, set the element to the desired value. The code will then take the `.infoz.parms` structure and the parmeter vector `b` to determine α, β, σ^2, and γ. At the end of the demo we will estimate nine variations of the model, as in Table 3 of Chan, Karolyi, Longstaff, and Sanders (1992).

Runing the demo by typing `ckls_d` gives the following results for the full model.

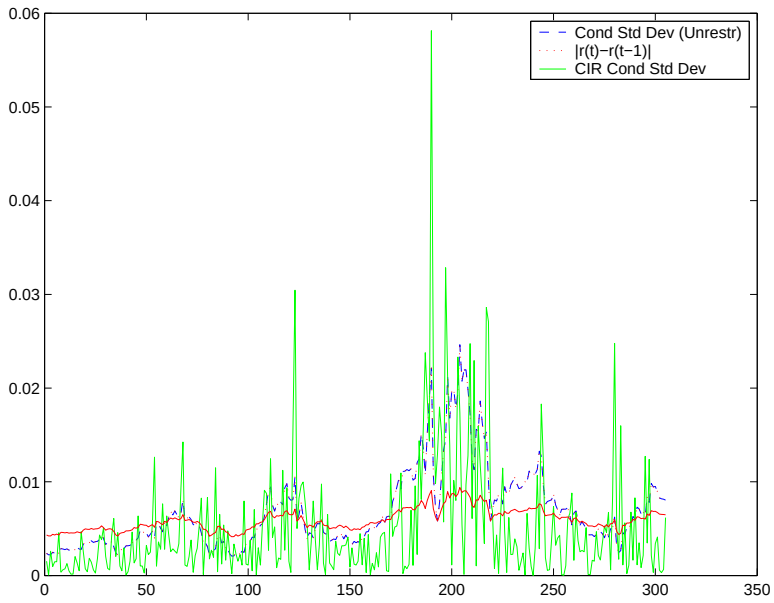
```
------------------  GMM PARAMETER ESTIMATES  -----------------
Parameter         Coeff     Std Err     Null    t-stat      p-val
alpha          0.041902    0.015939     0.00      2.63     0.0086
beta          -0.607658    0.269772     0.00     -2.25     0.0243
sigma^2        1.778786    2.905567     0.00      0.61     0.5404
gamma          1.508122    0.313462     0.50      3.22     0.0013

 ============================================================
```

For those that are familiar with this economic model, we can then convert the estimated parameters (α, β, σ^2, and γ) into the economically interesting θ, κ, and σ.

```
Long-run mean,    theta = 6.8956%
Speed of adj,     kappa = 0.6077
Volatility parm,  sigma = 1.3337
Cond. Vol. parm,  gamma = 1.5081
Average Cond Volatility = 0.6840%
R^2 (yld change)   = 0.0266
R^2 (sqrd yld chg) = 0.1576
```

Figure 7: CLKS Conditional Volatility Estimates



The demo then estimates the CIR version of the model by setting `gmmopt.infoz.parms = [NaN NaN NaN 0.5]'`. Results are

```
  ------------------  GMM PARAMETER ESTIMATES  -----------------
  Parameter         Coeff     Std Err     Null    t-stat      p-val
  alpha          0.028733    0.013869     0.00      2.07     0.0383
  beta          -0.420795    0.242914     0.00     -1.73     0.0832
  sigma^2        0.006557    0.001677     0.00      3.91     0.0001

  -------------------  GMM MOMENT CONDITIONS  ------------------
                     Moment     Std Err    t-stat      p-val
     Moment 1      0.000061    0.000033      1.83     0.0679
     Moment 2     -0.000007    0.000004     -1.83     0.0680
     Moment 3      0.000029    0.000016      1.83     0.0680
     Moment 4      0.000004    0.000002      1.83     0.0680

      J-stat = 4.0487     Prob[Chi-sq.(1) > J] = 0.0442
 ================================================================
Constraints:     gamma= 0.5000
Long-run mean,     theta = 6.8282%
Speed of adj,      kappa = 0.4208
Volatility parm,   sigma = 0.0810
Cond. Vol. parm,   gamma = 0.5000
```

```
Average Cond Volatility = 0.5926%
R^2 (yld change)   = 0.0128
R^2 (sqrd yld chg) = 0.0038
```

In comparing the CIR results to the more general model, we can see that that $\gamma = 0.5$ restriction is rejected. One way of looking at this is to see the p-value of 0.0442 on the overidentification test. Another way is to do a t-test of the null $\gamma = 0.5$ in the first estimation. I used the code `gmmopt.null = [0; 0; 0; 0.5]` to specify my null for the coefficients, and the output shows the p-value is only 0.0013. Figure 7 shows the estimated conditional standard deviation from both models, along with the absolute value of the return changes in the data. The general model is much more able to generate variation in conditional volatility.

We close out the demo by estimating nine models, as in Chan, Karolyi, Longstaff, and Sanders (1992). The table below shows the models and the setting of `gmmopt.infoz.parms`.

	`gmmopt.infoz.parms`			
Model	α	β	σ^2	γ
Full	NaN	NaN	NaN	NaN
Merton	NaN	0	NaN	0
Vasicek	NaN	NaN	NaN	0
CIR SR	NaN	NaN	NaN	0.5
Dothan	0	0	NaN	1
GBM	0	NaN	NaN	1
Brennan-Schwartz	NaN	NaN	NaN	1
CIR VR	0	0	NaN	1.5
CEV	0	NaN	NaN	NaN

The demo produces some tables with the parameter estimates, standard errors, and various measures of model fit.

5.7 Example With User's W

The demo `userw_d` shows how to use alternative weighting matrices. One estimation uses a fixed matrix of numbers as the initial weighting matrices, while other estimates use a user-defined function `userw` to calculate the weighting matrix initially and/or at subsequent iterations. The example uses a simple linear model.

5.8 Estimating Limited Dependent Variable Models

A special form of econometric model involves dependent variables which take on discrete values. Standard linear regression models are inappropriate in this context, but there are several popular alternatives. This example illustrates how to do this in GMM.

We focus on three types of models: linear probability, Logit, and Probit. Each one is based on writing the likelihood function for the data. Parameter estimates are the values that maximize the log-likelihood. Consider an observed variable y_i which is one if an event

is successful and zero otherwise. [I use i subsripts rather than t because these models are often used with a cross-section of observations rather than a time series.] We seek to explain when the event occurs or not with explanatory variables $\mathbf{x}_i$. The likelihood function is

$$\mathcal{L} = \prod_{i=1}^{N} [F(\boldsymbol{\beta}'\mathbf{x}_i)]^{y_i} [1 - F(\boldsymbol{\beta}'\mathbf{x}_i)]^{1-y_i}$$

where $F(\boldsymbol{\beta}'\mathbf{x}_i)$ is the probability the event was a success ($y_i = 1$), given $\mathbf{x}_i$. The three models we consider differ in specification of the F function. The linear probability model simply uses $F() = \boldsymbol{\beta}'\mathbf{x}_i$. The logit model is based on the logistic function, $F() = e^{\boldsymbol{\beta}'\mathbf{x}_i}/(1 + e^{\boldsymbol{\beta}'\mathbf{x}_i})$. The probit model uses the normal CDF, $F() = \Phi(\boldsymbol{\beta}'\mathbf{x}_i)$.

The maximum likelihood estimate sets the score of the log-likelihood function to zero,

$$\begin{aligned} \frac{\partial \ln \mathcal{L}}{\partial \boldsymbol{\beta}} &= \frac{\partial \left[\sum_{i=1}^{N} y_i \ln F(\boldsymbol{\beta}'\mathbf{x}_i) + (1 - y_i) \ln (1 - F(\boldsymbol{\beta}'\mathbf{x}_i))\right]}{\partial \boldsymbol{\beta}} \\ &= \sum_{i=1}^{N} \left[y_i \frac{F_1(\boldsymbol{\beta}'\mathbf{x}_i)}{F(\boldsymbol{\beta}'\mathbf{x}_i)} + (1 - y_i) \frac{-F_1(\boldsymbol{\beta}'\mathbf{x}_i)}{1 - F(\boldsymbol{\beta}'\mathbf{x}_i)} \right] = \mathbf{0} \end{aligned}$$

where F_1 means the first derivative of $F()$.

For the linear probility model, we have

$$\begin{aligned} \sum_{i=1}^{N} y_i(1 - \boldsymbol{\beta}'\mathbf{x}_i)\mathbf{x}_i + (1 - y_i)(\boldsymbol{\beta}'\mathbf{x}_i)\mathbf{x}_i &= \sum_{i=1}^{N} \left[y_i(1 - \boldsymbol{\beta}'\mathbf{x}_i) + (1 - y_i)(0 - \boldsymbol{\beta}'\mathbf{x}_i) \right] \mathbf{x}_i \\ &= \sum_{i \in y_i = 1} (y_i - \boldsymbol{\beta}'\mathbf{x}_i) + \sum_{i \in y_i = 0} (y_i - \boldsymbol{\beta}'\mathbf{x}_i) \\ &= \sum_{i=1}^{N} e_i \mathbf{x}_i = \mathbf{0} \end{aligned}$$

We can interpret the part in square brackets as the residual because the parts $(1 - \boldsymbol{\beta}'\mathbf{x}_i)$ or $(0 - \boldsymbol{\beta}'\mathbf{x}_i)$ are turned on or off depending on the value of y_i. Thus, MLE tells us to choose $\boldsymbol{\beta}$ to set the errors orthogonal to the $\mathbf{x}$'s, just like OLS. Consequently, I use the moment conditions and Jacobian from a linear model (`lingmmm` and `lingmmj`).

For symmetric distributions like the logistic or normal $F(-x) = 1 - F(x)$, so we can let $q_i = 2y_i - 1$ and use

$$\mathbf{m}_i = \sum_{i=1}^{N} \left[\frac{F_1(q_i \boldsymbol{\beta}'\mathbf{x}_i)}{F(q_i \boldsymbol{\beta}'\mathbf{x}_i)} \right] = \mathbf{0}$$

as moment conditions. The files `logitm.m` and `probitm.m` specify these, respectively.

Since analytic derivatives are not too cumbersome, we specifiy them as well. Note that the first derivative of the log-likelihood function is the vector of moment conditions. Therefore, the second derivative of $\ln \mathcal{L}$ is the derivative of the moment conditions (Jacobian). The files `logitj.m` and `probitj.m` give the Jacobians.

The final consideration is construction of the standard errors. In the maximum likelihood context, these are just the inverse of the negative of the Hessian of the log-likelihood function, evaluated at the final parameter estimates. We have already computed this analytically as the Jacobian, so we can make use of that work. What is needed is to tell the GMM code to use our own spectral density matrix, rather than one of the options provided by `gmmS.m`. To do so, we simply provide the name of our m-file in `gmmopt.S`. The relevant code from the file for the spectral density matrix of the logit model, `logitS.m`, is below:

```
function S = logitS(b,infoz,y,x,z)
S = -logitj(b,infoz,[],y,x,z)*rows(y);
```

The function is very simple. It is just a wrapper function that calls the Jacobian and changes the sign. The input and output arguments for your spectral density function must match those of `gmmS.m`. Note that we do not invert S, since that is handled in `gmm.m` by the code:

```
term = (M'*W*M)\eye(k);
bcov = term*(M'*W*S*W*M)*term/nobs;      % Cov(b)
```

Since $\mathbf{W}$ is an Indentity matrix, this reduces to $(\mathbf{M}'\mathbf{M})^{-1}(\mathbf{M}'\mathbf{S}\mathbf{M})(\mathbf{M}'\mathbf{M})^{-1}$. $\mathbf{M}$ is symmetric and positive definite (it is the Hessian of $\ln \mathcal{L}$), so we get $\mathbf{M}^{-1}\mathbf{M}'^{-1}\mathbf{M}'\mathbf{S}\mathbf{M}\mathbf{M}^{-1}\mathbf{M}'^{-1}$ which becomes $\mathbf{M}^{-1}\mathbf{S}\mathbf{M}'^{-1}$. Now it is clear that we want $\mathbf{S} = -\mathbf{M}$ to get $\text{var}(\mathbf{b}) = -\mathbf{M}^{-1}$. The multiplication by sample size in `logitS` is undoing the division in `gmm.m`.

To run the model, we need to set up some of the GMM options. For the logit model, the relevant code is

```
opt.gmmit = 1;
opt.W0 = 'I';
opt.infoz.momt = 'logitm';
opt.infoz.jake = 'logitj';
opt.S = 'logitS';
out2 = gmm(b0,opt,grade,X,X);
```

We just want a one-stage estimate, so we set `opt.gmmit = 1`. Also, we want to use an Identiy matrix as the initial weighting matrix rather than the cross-product of the instruments, so we specify that in the `W0` line. The next two lines just indicate the names of the files containing our moment conditions and Jacobian. Then we need to provide the name of the spectral density matrix file. Finally, we call the GMM routine. The demo does the estimation by GMM for the linear probability model, Logit, and Probit. For comparison, it also runs standard MLE Logit and Probit (from the Econmetrics Toolbox). Some of the output is shown below.

```
Coefficient Estimates
          Linear      Logit    Probit   Logit (ML)   Probit (ML)
Const    -1.4980   -13.0213   -7.4523     -13.0213       -7.4523
GPA       0.4639     2.8261    1.6258       2.8261        1.6258
TUCE      0.0105     0.0952    0.0517       0.0952        0.0517
PSI       0.3786     2.3787    1.4263       2.3787        1.4263

Standard Errors
          Linear    Logit   Probit   Logit (ML)   Probit (ML)
Const     0.5239   4.9313   2.5425       4.9313        2.5425
GPA       0.1620   1.2629   0.6939       1.2629        0.6939
TUCE      0.0195   0.1416   0.0839       0.1416        0.0839
PSI       0.1392   1.0646   0.5950       1.0646        0.5950

t-stat
          Linear     Logit    Probit   Logit (ML)   Probit (ML)
Const    -2.8594   -2.6405   -2.9311     -2.6405       -2.9311
GPA       2.8640    2.2377    2.3431      2.2377        2.3431
TUCE      0.5387    0.6722    0.6166      0.6722        0.6166
PSI       2.7200    2.2344    2.3970      2.2344        2.3970

Marginal Effects
          Linear    Logit   Probit
Const
GPA       0.4639   0.5339   0.5333
TUCE      0.0105   0.0180   0.0170
PSI       0.3786   0.4493   0.4679
```

5.9 Implementation of Classic Test Statistics

Here we see how to implement the three classic test statistics. I use the example where the full model is

$$y_t = \alpha + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \varepsilon_t = \mathbf{X}_t\boldsymbol{\beta} + \varepsilon_t.$$

The null we wish to test is $\beta_2 = 0$.

To fix some notation, we will call `uout` the output structure from the unrestricted estimate, and `rout` the restricted output. $\mathbf{y}$ is a $T \times 1$ vector of the dependent variable, $\mathbf{X}$ is a $T \times 3$ matrix (including a constant). The instruments $\mathbf{Z}$ are just the independent variables, so we have OLS. We use White's correction in calculation of the spectral density matrix. To run the demo, type `hyptest_d`.

```
gmmopt.S='W';
bu = [0;0;0];
br = [0;0];
```

```
gmmopt.infoz.momt='lingmmm';
uout = gmm(bu,gmmopt,y,X,X);
```

This is a "bare-bones" specification. You may set other options such as using analytic Jacobian and Hessian, specifying the spectral density matrix, etc.

Likelihood Ratio Test (LRT)

We want to look at the change in the objective function from the unconstrained model to the constrained version. It is important to estimate the restricted model using the same weighting matrix as the unrestricted model. To implement this, estimate the full model and save the weighting matrix. Then use this weighting matrix to estimate the restricted model. We drop β_2 from the parameter vector and $x_{2,t}$ from $\mathbf{X}_t$, but keep all the instruments.

```
W0 = uout.W;
gmmopt.W0 = 'Win';
rout=gmm(br,gmmopt,y,X(:,1:2),X,W0);
LR = rout.nobs*(rout.f - uout.f);
```

Wald Test (W)

The Wald test is based on the restrictions of the form $\mathbf{Rb} = \mathbf{r}$, where $\mathbf{R}$ is $q \times k$ and $\mathbf{r}$ is a q-vector with q denoting the number of restrictions and k denoting the number of parameters. Denote the variance of $\mathbf{b}$ as $\mathbf{\Sigma}$, so

$$\text{var}(\mathbf{Rb} - \mathbf{r}) = \mathbf{R\Sigma R}'$$

The test statistic is

$$W = (\mathbf{Rb} - \mathbf{r})'(\mathbf{R\Sigma R}')^{-1}(\mathbf{Rb} - \mathbf{r})$$

In the above example, $\mathbf{R} = [0\ 0\ 1]$ and $\mathbf{r} = 0$ so the test becomes $\beta_2^2/\text{var}(\beta_2)$, the square of the conventional t-statistic. The actual code need to test $H_0 : \beta_2 = 0$ is

```
R = [0 0 1];
r = 0;
W = (R*uout.b - r)'*inv(R*uout.bcov*R')*(R*uout.b - r);
```

To test if $\beta_1 = 0$ and $\beta_2 = 1$, set $\mathbf{R} = [0\ 1\ 0;\ 0\ 0\ 1]$ and $\mathbf{r} = [0;\ 1]$. To test $\beta_1 + \beta_2 = 0$ set $\mathbf{R} = [0\ 1\ 1]$ and $\mathbf{r} = 0$.

Lagrange Multiplier Test (LM)

Implementation using the GMM package is somewhat more complicated than with the other tests because it is necessary to take the restricted estimates and reevaluate the moment conditions and Jacobian for the full model at the restricted values. The test statistic is $LM = T(\mathbf{m}'\mathbf{WM})(\mathbf{M}'\mathbf{WM})^{-1}(\mathbf{M}'\mathbf{Wm})$.

```
% ---- Estimate Constrained Model (Normal W) ------------------------------
rout = gmm(br,gmmopt,y,X(:,1:2),X);
Wr = rout.W;

% ---- Reevaluate Unconstrained Model at Constrained Estimate -------------
gmmopt.infoz.call = 'gmm';
lmb = [rout.b; 0];                    % Construct restr version of full b
momt = fcnchk(gmmopt.infoz.momt);
jake = fcnchk(gmmopt.infoz.jake);
m = feval(momt,lmb,gmmopt.infoz,[],y,X,X);   % Reeval moment conditions
M = feval(jake,lmb,gmmopt.infoz,[],y,X,X);   % Reeval Jacobian
term1 = M'*Wr*m;
term2 = (M'*Wr*M)\eye(uout.nvar);
LM = rout.nobs*term1'*term2*term1;
```

We can compare each test statistic. The Wald is very close to the the t^2 from OLS (with White standard errors). The small difference is due to a degrees of freedom correction. The other statistics follow the ranking $W \geq LR \geq LM$, which holds in a finite sample for linear models.

```
Comparison of Test Statistics
                    t      t^2     Wald       LR       LM
Test Stat     -3.0255   9.1537   9.1262   8.7749   8.7741
p-value        0.0025   0.0025   0.0025   0.0031   0.0031
```

References

Andrews, Donald W. K., 1991, Heteroskedasticity and autocorrelation consistent covariance matrix estimation, *Econometrica* 49, 817–858.

——— , and J. Christopher Monahan, 1992, An improved heteroskedasticity and autocorrelation consistent covariance matrix estimator, *Econometrica* 60, 953–966.

Chan, K.C., G. Andrew Karolyi, Francis Longstaff, and Anthony Sanders, 1992, An empirical comparison of alternative models of the short-term interest rate, *Journal of Finance* 47, 1209–1227.

Cochrane, John H., 2001, *Asset Pricing* (Princeton University Press: Princeton, NJ).

Cox, John C., Jonathan E. Ingersoll, and Stephen A. Ross, 1985, A theory of the term structure of interest rates, *Econometrica* 53, 385–407.

Davidson, R., and J. MacKinnon, 1993, *Estimation and Inference in Econometrics* (Oxford University Press: New York).

den Haan, Wouter J., and Andrew Levin, 1999, A practitioner's guide to robust covariance matrix estimation, Working Paper.

Gallant, A. Ronald, 1987, *Nonlinear Statistical Models* (John Wiley & Sons: New York).

Gill, Philip E., Walter Murray, and Margaret H. Wright, 1981, *Practical Optimization* (Academic Press: New York).

Greene, William H., 1997, *Econometric Analysis* (Prentice Hall: Upper Saddle River, NJ) 3 edn.

Hamilton, James D., 1994, *Time Series Analysis* (Princeton University Press: Princeton, NJ).

Hansen, Lars Peter, 1982, Large sample properties of generalized method of moments estimators, *Econometrica* 50, 1029–1054.

——— , and Robert J. Hodrick, 1980, Forward exchage rates as optimal predictors of future spot rates: An empirical analysis, *Journal of Political Economy* 88, 829–853.

Newey, Whitney, and Kenneth West, 1987, A simple positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix, *Econometrica* 55, 703–708.

Press, William H., Saul A. Teukolsky, William H. Vetterling, and Brian P. Flannery, 1992, *Numerical Recipes in FORTRAN: The Art of Scientific Computing* (Cambridge University Press: New York) 2 edn.

White, Halbert, 1980, A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity, *Econometrica* 48, 817–838.