Dave Heath, USC Marshall School of Business

March 2013

This note and the accompanying files illustrate one way to do GMM estimation in Matlab. I certainly don't claim that it will always work or that it's anywhere close to optimal.

We will do the simplest model there is – OLS – but it's a template that works with most problems and introduces a few useful commands in Matlab. We want to estimate

$$y_t = \alpha + \beta x_t + \epsilon_t$$

Make sure your working directory is in the Matlab path.

Make a blank m-file and save it as work_file.m . I usually keep it open and then you can run the whole file by typing "work_file" in the command window, or else run individual commands and pieces of the file by highlighting in the editor and hitting F9.

Initializing,

```
clear all; close all; clc;
```

Import your data. In this case we'll just simulate some.

```
T = 100;
x = randn(T,1);
eps_raw = trnd(10,T-1,1);
y = 2*x + [eps_raw;0] + exp(x).*[0;eps_raw];
figure; plot(x,y, '.'); title('scatterplot Y vs X')
OLS_results = regstats(y,x);
```

$\alpha = 0$, $\beta = 2$, and our errors are fat tailed, heteroskedastic and autocorrelated.

Now we need to do the estimate. We will need another m-file, a function that takes in 1) parameters 2) data and 3) a weighting matrix and gives us a J-stat.

Open another blank m-file and save it as OLS_J.m .

```
function [Jstat, g_t, g_T] = OLS_J(param_vec, data, W)
```

OLS_J takes in the three arguments and gives us the J-stat, and g_t and g_T optionally.

param_vec is the vectorized parameters $\theta = [\alpha\, \beta]'$. Data is $[y\, x]$.

```
alpha = param_vec(1); beta = param_vec(2);
```

```
y = data(:,1); x = data(:,2);
T = size(y,1);
```

Given those, we want to compute the moments

$$u_t = y_t - \alpha - \beta x_t$$

$$g_t = u_t \otimes [1\, x_t]$$

```
u_t = nan(T,1);
g_t = nan(T,2);
for t = 1:T
    u_t(t) = y(t) - alpha - beta*x(t);
    g_t(t,:) = kron(u_t(t),[1 x(t)]);
end

g_T = mean(g_t);

Jstat = g_T*W*g_T';
```

That's a bit clunky but just to illustrate the general way with the kron() function.

Save OLS_J.m and let's make sure it works. Go back to work_file.m :

```
testparams = [0;1];
Jstat_test = OLS_J(testparams, [y,x], eye(2));
```

Eye(2) is the 2x2 identity matrix. Evaluating gives us a Jstat for $\theta = [0\, 1]'$. Okay, now we need to search for the best fit.

To do this we need to specify a function call that takes *only* a parameter vector, and gives the J stat, for the specified data and weighting matrix:

```
OLS_J1 = @(param_vec) OLS_J(param_vec, [y,x], eye(2));
Jstat_test2 = OLS_J1([0;1]);
```

Okay, now to search. I like to do a grid search or shotgun search to find a good starting point.

```
bestgridJ = 1e10; bestgridtheta = [NaN; NaN];
for gridalpha = -10:1:10
    for gridbeta = -10:1:10
        Jstat = OLS_J1([gridalpha;gridbeta]);
        if Jstat < bestgridJ
            bestgridJ = Jstat;
            bestgridtheta = [gridalpha;gridbeta];
```

```
        end
    end %i
end %j
```

That steps over the parameter space from [-10,-10] to [+10,+10] in increments of one and picks the best point. We use that as our starting point for the real search.

```
theta_hat = fminunc(OLS_J1, bestgridtheta);
disp('OLS coeffs 1st stage coeffs')
disp([OLS_results.beta, theta_hat]);
```

We get exactly the same estimate as OLS.

Okay, now on to the second stage. First we have to evaluate the model at the first stage to get the $g_t(\theta)$ for our optimal weighting matrix.

```
[Jstat, g_t, g_T] = OLS_J1(theta_hat);

figure; plot(g_t); title('Time series of the moments')
figure; plot(g_t(1:end-1,1),g_t(2:end,1),'+'); title('g1_t vs g1_{t-1}')
figure; plot(g_t(1:end-1,2),g_t(2:end,2),'o'); title('g2_t vs g2_{t-1}')
```

Always a good idea to look at the pricing errors / moments. They potentially look autocorrelated from the last two plots (plus we set it up that way), so let's do Newey West:

```
Acovg = g_t.'*g_t/T;
num_lags = 1;
for n = 1:num_lags
     NWweight = 1 - n/(num_lags+1);
     lag_cov = g_t(1+n:end,:).'*g_t(1:end-n,:)/T;
     Acovg = Acovg + NWweight*(lag_cov+lag_cov');
end

W2 = inv(Acovg);
```

Now let's define our function call that evaluates the J stat using the optimal weighting matrix:

```
OLS_J2 = @(param_vec) OLS_J(param_vec, [y,x], W2);
```

The first stage $\hat{\theta}$ is a good guess for a starting point. (Of course since this system is exactly identified, it's a perfect guess.)

```
theta_hat2 = fminunc(OLS_J2, theta_hat);
disp('OLS coeffs 2nd stage coeffs')
```

```
disp([OLS_results.beta, theta_hat2]);
```

And finally, standard errors. With the optimal weighting matrix the formula is simple. We just need $\frac{\partial g_T}{\partial \theta}$, the gradient of the moment estimates. Again, we could easily get them analytically but as an illustration of the more general approach, let's do finite difference:

```
stepsize = 1e-10;
[ans, ans, g_T] = OLS_J2(theta_hat2);
for i = 1:2
    theta_hat2_fd = theta_hat2;
    theta_hat2_fd(i) = theta_hat2(i)+stepsize;
    [ans, ans, g_T_fd] = OLS_J2(theta_hat2_fd);
    dgT(:,i) = (g_T_fd - g_T)'/stepsize;
end
```

gives the gradient matrix via finite difference, and

```
thetahat2_SE = sqrt(diag(inv(dgT'*W2*dgT))/T);
```

gives the standard errors. In the end,

- We exactly recapitulate the OLS estimate (because we are exactly identified and can always set the Jstat to zero)
- The standard error for α is about the same size in OLS vs GMM
- The standard error for β is larger in GMM reflecting the autocorrelation in the errors