Empirical exercise – The MM estimator and the two-step GMM estimator

1. Aim of the exercise

The aim of this exercise is to understand using the method of moments (MM) estimator and the two-step generalised method of moments (GMM) estimator to estimate a model that is non-linear in the parameters. The empirical context is as follows. We are interested in analysing how age, education, and gender affect household income. We suspect that these explanatory variables have a non-linear effect on the dependent variable. Therefore, we decide to consider a regression model where the systematic component is an exponential function of the independent variables and their coefficients. There are different estimators that can be used to estimate a model that is non-linear in the parameters. The non-linear least squares estimator or the method of moments estimator are among them.

We will first study how we can use the information provided by the three explanatory variables (age, education, and gender) of the model to estimate the coefficients of these variables using the method of moments estimator. We will then study how we can make use of extra information provided by additional variables (health satisfaction and marital status) to estimate the same coefficients using the two-step generalised method of moments estimator.

The data is from *Regina T. Riphahn, Achim Wambach and Andreas Million, 2003. Incentive effects in the demand for health care: a bivariate panel count data estimation, Journal of Applied Econometrics, 18 (4), 387-405.* It can be accessed through the link `http://qed.econ.queensu.ca/jae/2003-v18.4/riphahn-wambach-million/`. It consists of an unbalanced panel of 7,293 households surveyed in the German Socio-economic Panel (GSOEP), and includes information on household income, age, gender, and education, among numerous other socio-demographic variables. For this exercise, we use data from the last wave of the panel collected in 1988 that contains 4,483 observations. Two of the survey respondents in this cross-section of the panel data reported zero income. Deleting these two observations leaves a sample of 4,481 observations.

This exercise is based on the method of moments example considered on page 512 in *Greene, William H. Econometric Analysis. Pearson, 2012.* We will replicate the estimation results presented in this example. See this book for other details of the example.

2. Load data

Load the enclosed data in mat format into MATLAB.

```
clear;
load 'M:\exercisegmm.mat';
clearvars -except hhninc age educ female married hsat;
```

3. Produce descriptive statistics

Examine the correlation coefficients between the variables of interest using the `corrcoef` function. The function returns as output `R1` and `P1` arguments, which contain the correlation

coefficient, and the p value corresponding to the test of the null hypothesis of no correlation. Inspect all the correlations.

```
A = [hhninc/10000 age educ female];
[R1,P1] = corrcoef(A);
```

4. Create the systematic component of the regression and define the number of parameters and moments

Consider the code presented at the end of the section. `X` is the vector of regressors. `Z` is the vector of instruments that will be used to construct moment conditions. If `X = Z`, the estimation method is the MM. Otherwise it is the GMM. This will become clear later in the exercise. The remaining lines of the code define the number of coefficients to be estimated, and the number of moment conditions.

```
y = hhninc/10000;
N_obs = length(y);
x_0 = ones(N_obs,1);
X = [x_0 age educ female];
Z = [x_0 age educ female];
% Z = [x_0 age educ female married hsat];
N_par = size(X,2);
N_mom = size(Z,2);
```

5. Step one of GMM

In this and the next section, we construct the two-step GMM estimator, and consider the MM estimator as a special case.

In the first step of GMM, we start by creating an initial weight matrix. Using this weight matrix we obtain parameter estimates. In the second step of GMM, we create a new weight matrix using the estimates from the first step. Using this new weight matrix we obtain the parameter estimates of interest.

The regression model at hand is non-linear, and it leads to a GMM objective function that is non-linear. We need to minimise this function in the first as well as the second step of the two-step GMM estimation. This objective function is stored in the enclosed MATLAB function file. Now give a pause reading this handout, and take a moment to read the handout that explains the content of the enclosed MATLAB function file.

The enclosed MATLAB function file presented a non-linear optimisation problem that must be solved iteratively using a numerical method. The paragraphs below explain how we can make use of a numerical method built in MATLAB.

See the code presented at the end the section. In particular consider the built-in function `fminunc`. `fminunc` stands for 'find minimum of unconstrained multivariable function'. It is an iterative algorithm that attempts to find the local minimum of an objective function in a given interval. It is a derivative-based search algorithm, and does not guarantee a global minimum. It turns out that any derivative-based optimisation algorithm does not guarantee a

global minimum. There are other algorithms for non-smooth or constrained objective functions but we do not consider them here.

The input arguments of the `fminunc` function are the following. The first input argument is `@(B_true)exercisegmmfunction(.)`. It specifies that the objective function is evaluated at `B_true`. The second input argument is `B_ig`. It specifies a set of initial guesses for the values of the parameters of interest. That is, the minimiser is `B_hat`, but to find the `B_hat` the built-in iterative algorithm starts the minimisation algorithm with the initial guess `B_ig`. Obviously, `B_ig` must be of the same size as `B_hat`. The third input argument is `options`. It is a structure array containing all the options about the way in which the algorithm is searching for the minimum.

The input arguments of the `options` function are the following. `GradObj` specifies that the function file `exercisegmmfunction.m` contains a user-defined gradient of the objective function. In the options, we supply `fminunc` with the gradient of the objective function because `fminunc` is a derivative-based search algorithm. When we define the gradient in the objective function and set `GradObj` to `on` in the options of the solver `fminunc`, `fminunc` will use the so-called 'trust-region algorithm' to search for the minimum. If we set `GradObj` to `off`, `fminunc` will disregard our gradient and estimate the gradient with a built-in algorithm. `MaxFunEvals` indicates the maximum number of function evaluations allowed. It takes a positive integer, and we set it to `27100` iterations. `MaxIter` is the maximum number of iterations allowed. It takes a positive integer and we set it to `27100`. These numbers can be changed to make them large enough to find the minimum of the objective function. `Display` shows the steps, or the final outcome of the minimization. For instance, `iter` displays output at each iteration, `final` displays just the final output, or `notify` displays output only if the function does not converge. You may type `doc optimset` in the command window to see the full list of options.

The output arguments of the `fminunc` function are the following. `B_hat` is where the function attains a minimum. Hence, `B_hat` is the solution of the objective function. `fval` is the value of the objective function at the solution `B_hat`. `exitflag` indicates the reason `fminunc` stopped. The possible reasons are coded with an integer. For instance, `exitflag` returns the integer `0` which means that the number of iterations exceeded `MaxIter` or number of function evaluations exceeded `MaxFunEvals`. You may type `doc fminunc` in the command window and go to section 'Output Arguments' to see the other integers that `exitflag` could return and their respective descriptions. `output` is a structure array that contains, for example, the number of iterations and the number of function evaluations considered. Type `doc fminunc` and go to section 'Output Arguments' to learn about all the elements of the structure. `GradObj` is the value of the gradient of the objective function at the solution `B_hat`. Finally, `hesssian` is the value of the Hessian of the objective function at the solution.

We have learned about the objective function we need to minimise in the first or the second step of the two-step GMM estimation. If we consider an exactly-identified model, then the first step of GMM estimation is in fact the MM estimation. Consider the case that we have access to only the independent variables age, education, and gender. Including the constant, there are four variables and hence four coefficients to estimate. Set the vector `Z` equal to `X`. Create the initial weighting matrix. Obtain `B_hat_so`. These are the MM estimates.

From now on we continue with an over-identified model. Let the vector `Z` include the additional variables marital status and health satisfaction. Using the initial weighting matrix, obtain the first-step GMM estimates. Proceed to the next section to obtain the second-step GMM estimates.

```
% Create the initial weighting matrix
W_hat = inv(eye(N_mom));
% Minimise the objective function using the initial weighting matrix
B_ig = [1 1 1 1]';
options = optimset('GradObj','on','MaxFunEvals',27100,'MaxIter',27100,'Display',...
    'iter','TolFun',1e-10,'TolX',1e-10,'Algorithm','trust-region');
[B_hat_so,Obj_so,exitflag_so,output_so,GradObj_so,hessian_so] = ...
    fminunc(@(B_true)exercisegmmfunction(y,X,Z,B_true,W_hat,N_obs,...
    N_par),B_ig,options);
```

6. Step two of GMM

In the first step of GMM above, we have obtained the initial estimates of the coefficients of the model. In this second step of GMM we use these estimates to construct a new weight matrix. This new weight matrix takes a specific form. This form is used because it minimises the asymptotic variance of the GMM estimator. The GMM estimator that uses this 'optimal' weight matrix is then the optimal GMM estimator. Consider the code presented at the end of the section. We first construct this optimal weight matrix. We then use it to obtain the optimal GMM estimates using the minimisation routine in MATLAB.

Note that at the start of the minimisation routine, we set the initial guess vector, `B_hat_ig`, to the initial estimates from the first step of GMM, `B_hat_so`. This is to help the minimisation algorithm to search for the coefficient estimates that minimise the objective function in the second step of GMM, which should somewhat be close to the initial coefficient estimates from the first step of GMM.

```
% Create the optimal weighting matrix
e = y-exp(X*B_hat_so);
W_hat = inv((1/N_obs)*(e.*Z)'*(e.*Z));
% Minimise the objective function using the optimal weighting matrix
B_ig = B_hat_so;
options = optimset('GradObj','on','MaxFunEvals',1000,'MaxIter',1000,'Display',...
    'iter','TolFun',1e-10,'TolX',1e-10,'Algorithm','trust-region');
[B_hat_st,Obj_st,exitflag_st,output_st,GradObj_st,hessian_st] = ...
    fminunc(@(B_true)exercisegmmfunction(y,X,Z,B_true,W_hat,N_obs,...
    N_par),B_ig,options);
```

7. Estimate the variance-covariance matrix of the GMM estimates

The variance-covariance matrix of the GMM estimates can be calculated using the code presented at the end of the section.

```
G_bar = -(1/N_obs)*Z'*(X.*repmat(exp(X*B_hat_st),[1 N_par]));
B_hat_st_VCE = (1/N_obs)*inv(G_bar'*W_hat*G_bar);
B_hat_st_SEE = sqrt(diag(B_hat_st_VCE));
```

8. Hansen-Sargan test of overidentifying restrictions

Carry out the Hansen-Sargan test of overidentifying restrictions. This is also called the J test.

```
e = y-exp(X*B_hat_st);
m_bar = (1/N_obs)*Z'*e;
J = N_obs*m_bar'*W_hat*m_bar;
J_df = N_mom-N_par;
J_c_95 = chi2inv(0.95,J_df);
J_p = chi2cdf(J,J_df,'upper');
```

9. Notes on the solution of the generalised method of moments estimation

The `B_hat_st` estimates could have resulted in values slightly different from what they are. This means that the minimisation is not necessarily perfect. Numerical optimisation problems might cause this. For instance, there might be multiple local minimum points, or the function might be very flat around the minimum. We could adjust the function so that convergence is faster or more reliable. We could also study the scatter plot of `Y` against `X` which might signal the location of the minimum. Initial values specified by `B_ig` might matter too. At the end of the iteration, MATLAB displays 'Solver stopped prematurely'. This link leads to some information that might provide clues. We could also experiment with the number of iterations. That is, increasing the number of iterations could increase the chance of finding a better minimum.