Empirical exercise – The maximum likelihood estimator and the linear regression model

1. Aim of the exercise

The aim of this exercise it to understand how to carry out maximum likelihood estimation. Suppose that the model of interest is the standard linear regression model with multiple regressors that takes the form $y = X\beta + u$. We assume that u follows the normal distribution. We consider simulated data for the variables of the model. We will study how to estimate the parameters of this model using the maximum likelihood estimator. The exercise assumes knowledge of the theoretical derivation of the maximum likelihood estimation of the linear model.

2. Set seed

Set seed for the random number generators we will make use of to generate simulated data.

```
clear;
rng(1)
```

3. Define the number of observations, number of all parameters in the model, number of coefficients

Define the number of observations, number of all parameters in the model, number of coefficients. Let the number of observations to be 50. Let the model contain five parameters: four coefficients and the standard deviation of the regression error. Define numbers for the total number of parameters and for the coefficients using the code presented at the end of the section.

```
N_obs = 50;
N_par = 5;
N_par_coefficients = 4;
```

4. Simulate data

Denote the systematic component of the regression equation with `X`. We let it contain a constant, and three regressors, where each regressor follows the standard normal distribution. Hence, `X` is `N_obs` by `N_par`. `e` is the error term, and we let it follow a normal distribution with mean 0 and standard deviation 0.5. It is a `N_obs` by `1` vector. `B_true = [1 2 3 4]'` is the true coefficient vector. It is a `N_par` by `1` vector. Under these assumptions, the assumed linear model generates data for `y`.

```
x_0 = ones(N_obs,1);
x_1 = random('normal',0,1,N_obs,1);
x_2 = random('normal',0,1,N_obs,1);
```

```
x_3 = random('normal',0,1,N_obs,1);
X = [x_0 x_1 x_2 x_3];
mu = 0;
sigma = 0.5;
e = random('normal',0,sigma,N_obs,1);
B_true = [1 2 3 4]';
y = X*B_true+e;
```

5. Obtain the ML estimates by maximising the maximum likelihood criterion function using the gradient computed analytically

Consider the code presented at the end of the section. In the first line, `T_ig` specifies an initial guess for the parameter vector `T_true`. `fminunc` starts to search for the minimum using this initial guess vector. `T_true` contains all the parameters of the log-likelihood function that we need to estimate. In particular, it contains five elements; the first four elements are for the four elements of the parameter vector `B_true`, and the fifth element is for the scalar parameter `sigma`.

Now pause reading this file and study the content of the enclosed function file named as `exercisemllrmfun` function. In particular, study the how the maximum likelihood objective function and the gradient of it are constructed based on the maximum likelihood theory for the linear regression model.

Consider the remaining part of the code presented at the end of the section. The `fminunc` function uses a derivate-based search algorithm to find the minimum of the objective function. If we are able to calculate the gradient analytically, we can supply `fminunc` with this gradient. In the current exercise, the gradient of the standard linear model has a simple closed-form solution, as considered in the enclosed function. To supply the `fminunc` function with this gradient, in the `options` of `fminunc`, we set `GradObj` to `on` which instructs `fminunc` to consider the gradient contained in `exercisemllrmfun` when searching for the minimum of the objective function.

```
T_ig = [1 1 1 1 1]';
options = optimset('GradObj','on','MaxFunEvals',1000,'Display','on', ...
    'DerivativeCheck','off','Algorithm','trust-region');
T_hat_a = fminunc(@(T_true)exercisemllrmfun(y,X,T_true,N_obs,N_par),T_ig, ...
    options);
```

6. Obtain the ML estimates by maximising the maximum likelihood criterion function using the gradient computed numerically

The minimisation of an objective function is most precise if we supply `fminunc` with a gradient. However, it is often difficult, if not impossible, to obtain a closed-form solution for the gradient. In such a case we have to rely on numerical approximation of the gradient. In the code presented at the end of the section, in the `options` of `fminunc`, we set `GradObj` to `off` so that `fminunc` considers a built-in algorithm to compute the gradient numerically. Using this gradient, `fminunc` minimises the objective function specified in the function `exercisemllrmfun`.

Forcing `fminunc` to use the numerical approximation of the gradient comes with a price. The price is that the algorithm can get stuck in some regions of the parameter space, and hence fail to converge.

```
T_ig = [1 1 1 1 1]';
options = optimset('GradObj','off','MaxFunEvals',1000,'Display','on', ...
    'DerivativeCheck','off','Algorithm','quasi-newton');
T_hat_n = fminunc(@(T_true)exercisemllrmfun(y,X,T_true), ...
    T_ig,options);
```

7. Inference

To carry out inference on the population coefficients of the linear model, we need the variance estimates of the coefficient estimates. There are different estimators of the variance-covariance matrix of the maximum likelihood estimates. In the code presented at the end of the section, in the first five lines we prepare input for the calculation of the alternative estimators of the variance-covariance matrix of the maximum likelihood estimates. See the definitions of `G`, `Hessian_matrix_B_true`, and `Information_matrix_B_true` in the lab notes or, respectively, on pages 405, 401, and 407 in Davidson and MacKinnon (1999).

In the remaining part of the code, three estimators are considered. The first is the "Empirical Hessian" estimator. The second is the "Outer-product-of-the-gradient" estimator. The third estimator is the "Sandwich" estimator.

```
% Input for alternative estimators
B_hat = T_hat_a(1:end-1);
sigma_hat = T_hat_a(end);
G = -(1/sigma_hat^2)*(y-X*B_hat).*-X;
Hessian_matrix_B_true = -(1/sigma_hat^2)*(X'*X);
Information_matrix_B_true = -Hessian_matrix_B_true;
% Alternative estimators
B_hat_VCE_EH = inv(Information_matrix_B_true);
B_hat_VCE_OPG = inv(G'*G);
B_hat_VCE_S = inv(Hessian_matrix_B_true)*(G'*G)*inv(Hessian_matrix_B_true);
```

8. Inference

Given the variance estimates of the coefficient estimates, we can now carry out hypothesis testing on the population coefficients of the model. Two tests are considered in the code presented at the end of the section. Using the t test we check whether a given coefficient is statistically different from zero. The $Wald$ test is a test of multiple linear restrictions on a set of coefficients. Here we test if the coefficients of three regressors are jointly equal to 0.

```
% t test
B_hat_SEE_S = sqrt(diag(B_hat_VCE_S));
t = B_hat./B_hat_SEE_S;
```

```
% Wald test
R = [0 1 0 0;0 0 1 0;0 0 0 1];
q = [0;0;0];
Wald = (R*B_hat-q)'*inv(R*B_hat_VCE_S*R')*(R*B_hat-q);
p = 1-chi2cdf(Wald,size(R,1));
```

9. Obtain the OLS estimates and compare them with the ML estimates

Use the supplied function `exercisefunctionlssrobust` to obtain the OLS estimates of coefficients of the model, the square root of the mean squared error of the regression, and the variance-covariance matrix of the coefficient estimates.

`T_hat_a(1:end-1)` contains the ML estimates of the coefficients of the model. `LSS.B_hat` contains the OLS estimates. Compare the two types of estimates. They are equal to each other. This is because the OLS estimator is the same as the ML estimator if we assume that the errors of the model are normal and homoskedastic.

The last element of `T_hat_a` contains the ML estimate of `sigma`. Compare this with the OLS estimate `sigma_hat`. Note that the former is smaller than the latter. This is because the sum of squared residuals is divided by N in the former case, while it is divided by $N - K$ in the latter case.

`B_hat_VCE_S` is the sandwich estimator of the variance-covariance matrix of the ML estimates. Compare this with the OLS estimator of the variance-covariance matrix that is robust to heteroskedasticity. They are equal to each other if the `LSS.N_obs/(LSS.N_obs-LSS.N_par)` adjustment is not made when calculating `LSS.B_hat_VCE_robust`.

```
LSS = exercisefunctionlssrobust(y,X);
% Compare the coefficient estimates
T_hat_a(1:end-1)
LSS.B_hat
% Compare the square root of the mean squared error estimates
T_hat_a(end)
LSS.sigma_hat
% Compare the estimated variance-covariance matrix of the ML and the OLS estimates
B_hat_VCE_S
LSS.B_hat_VCE_robust
```

10. Bonus: search for global minimum

The `fminunc` solver uses an algorithm that searches for the local minimizer of the objective function starting with an initial guess of the minimizer. We could check if different initial guesses lead to the same minimizer to infer that the local minimizer is also the global minimizer. This is roughly what the `GlobalSearch` solver does. It starts the local solver `fmincon` from multiple initial guesses, and rejects those guesses that are unlikely to improve the best local minimum found across iterations.

```
T_ig = [1 1 1 1 1]';
```

```
options = optimset('GradObj','on','MaxFunEvals',1000,'Display','on', ...
    'DerivativeCheck','off');
objective = @(T_true)exercisemllrmfun(y,X,T_true);
problem = createOptimProblem('fmincon','objective',objective,'x0',T_ig, ...
    'options',options);
gs = GlobalSearch('Display','iter');
T_hat_gs = run(gs,problem);
```