Learning MATLAB using OLS

Suppose we believe that there is a linear relationship between a dependent variable y and an explanatory variable x, i.e., $y = \alpha + \beta x$. Consider the simple ordinary least squares problem:

$$\min_{\hat{\alpha},\hat{\beta}} \sum_{i=1}^{N} (y_i - \hat{\alpha} - \hat{\beta} x_i)^2 \tag{1}$$

The normal equations (first-order conditions) for $\hat{\alpha}$ and $\hat{\beta}$ are

$$\sum_{i=1}^{N} (y_i - \hat{\alpha} - \hat{\beta} x_i) = 0, \tag{2}$$
$$\sum_{i=1}^{N} (y_i - \hat{\alpha} - \hat{\beta} x_i) x_i = 0. \tag{3}$$

We know from basic econometrics that we can solve analytically for $\hat{\alpha}$ and $\hat{\beta}$:

$$\hat{\beta} = \frac{\sum_{i=1}^{N} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{N} (x_i - \bar{x})^2}, \tag{4}$$
$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}. \tag{5}$$

Equivalently, we can write the OLS problem using matrix and vector notation. Define

$$y = \begin{bmatrix} y_1 & \cdots & y_N \end{bmatrix}',$$

$$X = \begin{bmatrix} 1 & \cdots & 1 \\ x_1 & \cdots & x_N \end{bmatrix}',$$

and let

$$b = \begin{bmatrix} \hat{\alpha} & \hat{\beta} \end{bmatrix}'.$$

Then the problem can be written as

$$\min_{b} (y - Xb)'(y - Xb) \tag{6}$$

and

$$b = (X'X)^{-1}(X'y). \tag{7}$$

We will solve this problem in four different ways: computing the estimators directly using both the matrix notation and the summations, finding the zero of the system of normal equations using a nonlinear equation solver, and using a non-gradient-based optimization algorithm to find the minimum of the objective function. But first we must generate some data.

```
-------------------GenerateData.m----------------------

%GenerateData
%
%GenerateData(alpha,beta,N)
%
%alpha: y-intercept
%beta: slope
%N: number of points
function GenerateData(alpha,beta,N)
```

```
x=[1:1:N]';

y=alpha + beta * x + randn(N,1);
%y=alpha + beta * x ;

save data x y

figure(1);
plot(x,y,'b',x,alpha+x*beta,':r')
```

The following code will solve for the estimators directly using the matrix notation outlined above:

```
--------------------OLS1.m-------------------------

function [alphaHat, betaHat] = OLS1

load data;

N = length(x)

X = [ones(N,1) x];

b = X\y;
%b = (X'*X)\(X'*y);
%b = inv(X'*X)*(X'*y);

alphaHat = b(1);
betaHat = b(2);

return;
```

This code solves for the estimators directly using the summations and means:

```
---------------------OLS2.m-----------------------------

function [alphaHat, betaHat] = OLS2

load data;

N = length(x);
meanX = sum(x)/N;
meanY = sum(y)/N;

betaHat = sum((x-meanX).*(y-meanY))./sum((x-meanX).^2);

alphaHat = mean(y) - betaHat * mean(x);

return;
```

In order to find the estimators by solving the system of normal equations we will use MATLAB's nonlinear equation solver, `fsolve`. `fsolve` is part of MATLAB's optimization toolbox. It is a robust nonlinear equation solver which adjusts the algorithm it employs based on characteristics of the problem. `fsolve` takes as its first argument a function. The function should take as it's arguments the variables being solved for and return the value of the nonlinear equations as functions of those variables. This function is setup below for the OLS problem.

```
---------------------focs.m------------------------------------

function out = focs(in)
```

```
global x y

alphaHat = in(1);
betaHat = in(2);

out(1) = sum(y-alphaHat-betaHat.*x);
out(2) = sum((y-alphaHat-betaHat.*x).*x);
```

Following is the code which calls fsolve on the normal equations.

```
------------------OLS3.m--------------------------------------

function [alphaHat, betaHat] = OLS3

global x y

load data;

[sol, fval] = fsolve('focs',[2 1], optimset('TolFun',1.e-10));

error = sum(fval.^2);

if (error > sqrt(1.e-10))
    disp('sol is not a zero!');
end

alphaHat = sol(1);
betaHat = sol(2);
```

Finally we will use `fminsearch` to solve for the estimators by minimizing the objective function. `fminsearch` is also part of MATLAB's optimization toolbox. It uses a simplex method to converge to the minimizers. The simplex method is a non-gradient based method which means that the algorithm does not attempt to compute the objective function's gradient. Thus `fminsearch` is a good choice when the objective function is discontinuous or non-smooth. Note that for the problem at hand, `fminsearch` is an extremely bad choice. First of all because there are algorithms which are optimized for minimizing objective functions of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})^2 \tag{8}$$

and these algorithms are implemented in `lsqnonlin`. Second, while non-gradient based methods are more robust, gradient-based methods are, in general, more efficient. Hence, whenever possible, to save on computation time, one should use a gradient-based method. Clearly, are objective function is smooth and continuous so preferably we could use `fminunc` which is a function that implements a robust Newton algorithm. Thus my choice to use `fminsearch` here is clearly for pedagogical purposes.

`fminsearch` takes as it's first argument the function to be minimized. This function should take as its arguments the variables over which is will be minimized and return the value of the function given those variables. The code for the objective function is given below:

```
---------------SSE.m----------------------------------------------

function out = SSE(in)

global x y

alphaHat = in(1);
betaHat = in(2);

out = sum( ( y- alphaHat - betaHat .* x ).^2 );
```

The following code demonstrates how to call `fminsearch` to minimized the sum of squared errors:

```
--------------------OLS4.m----------------------------------------
function [alphaHat, betaHat] = OLS4

global x y

load data;

[sol, fval] = fminsearch('SSE',[1 .5], optimset('TolFun',1.e-15,'display','final'));

alphaHat = sol(1);
betaHat = sol(2);
```