Exercise: Basics in Matlab

Stockholm Doctoral Program in Economics
Computational Bootcamp

Kathrin Schlafmann

This exercise is meant to make you familiar with coding in Matlab. How easy these exercises will be depends on your prior experience in programming: If it is your first interaction with coding you will see and learn many new things. If you are already familiar with coding, these exercises will be easier, but you might still discover some features in Matlab you weren't aware of or realize that there are specific topics you might want to read up on. In any case, if you want further practice in Matlab after completing these exercises, check out the resources mentioned at the end of the lecture slides.

Questions

1. Getting familiar with the different windows in Matlab:
   (a) Construct a scalar variable `a` with value 1 in the "Command Window". Do you see any written output in the "Command Window" after executing? If so, how can you suppress it? If not, why don't you?
   (b) Locate the constructed variable in the current "Workspace". How do you see that you indeed constructed a scalar?
   (c) Open the variable `a` in the "Variables" Window - This is another way of seeing what you constructed.
   (d) Use the `size()`-function to check the dimensions of `a` - a third way of making sure that you constructed a scalar.

2. Constructing a vector:
   (a) Construct a row vector with 10 elements which are all equal to 1 and call it `aVec`. Is there more than one way of doing that?
   (b) How can you make sure you constructed the correct object?
   (c) Construct a column vector `bVec` that is the transpose of the row vector `aVec`.
   (d) Construct a column vector `cVec` that is identical to `bVec` except for the 2nd element (which is equal to zero). How can check that you did the correct thing?
   (e) What if you wanted `cVec` to differ from `bVec` not in the 2nd element, but in the last-but-one element? And could you write it in a way that it would even work if you do not know the exact length of `bVec`? This will be a very convenient way of coding when you want your code to be flexible with respect to grid sizes later on.

3. Working with matrices:

   (a) Construct manually the following matrix:

$$aMat = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 20 & 15 & 10 & 5 \end{bmatrix}$$

   Make sure you constructed the correct matrix.

   (b) Construct another, identical, matrix `bMat` the following way:

      i. Construct 3 row vectors which are equal to the 3 rows in matrix `aMat`. Can you do that without typing each element?

      ii. combine the 3 vectors to form the matrix `bMat`.

   (c) How can you be sure that matrix `aMat` and matrix `bMat` are identical? Look up the function `isqual()` in Matlab's help menu to see how you can do it.

   (d) Construct a matrix `cMat` by replacing the last row in matrix `aMat` with a vector of twos.

   (e) If you use `isequal()` to compare `aMat` with `cMat`, what is the result?

4. Working with scripts:

   (a) Open the file `exerciseBasic.m` in the "Editor" window.

   (b) This file contains commands, comments and cells. What is what in this file?

   (c) Look at the cell referring to Question 4: What do you expect the code in this cell to do?

   (d) Highlight the code in this cell and execute it. Were you right about what the code does?

   (e) Can you also execute this cell without highlighting the code first?

   (f) Can you rewrite the code of this cell to construct a vector `approx2` and `trueval2` which is identical to the output vectors but are constructed without using a loop? Test that `approx2 = approx` and `trueval2 = trueval`.

5. Working with for-loops:

   (a) You want to calculate the sum of all numbers from 1 to 100. Look at the code fragment in the cell for Question 5. Can you complete the for-loop to calculate the sum?

   (b) Is it necessary to use a loop here? Could you use a vector instead and sum the elements?

   (c) Do you even need to construct the vector first?

6. Working with if-statements:

(a) Look at the code fragment in the cell for Question 6. You see a variable x with a specific value. Depending on this value you want Matlab to print in the command line whether x is larger than or smaller-or-equal to 100 (Displaying text on screen can be done with the function `disp('text to print')`). Write an if-statement that does that.

(b) Play around with the value of x. Does your code do what you expect it to do?

(c) Now you are interested whether x is larger than 200, >100 and <=200, or <= 100. Alter your if-statement to allow for these 3 possibilities. Play around with x. Does it work?

7. Working with functions:

(a) Open the file `firstFunction.m`. This is a function file. How do you know this?

(b) Look at the function. What do you expect it to do?

(c) Execute the cell that calls the function. Were your expectations right?

(d) When you execute this cell, why does it not generate variables `x`, `y`, `z` and `w`?

8. Working with functions, cont'd:
You want to run OLS regressions where a dependent variable y is a function of exogenous regressors X (a matrix). There are regression models implemented in Matlab, but we ignore this for now and pretend we would have to code it manually. Recall that the OLS estimator is defined as

$$\beta = (X'X)^{-1}X'y$$

(a) The code in the cell for question 8(a) generates the exogenous regressors X and the dependent variable y. Write the line that computes β.

Now you realize that you are in fact interested in several dependent variables, not only the one you have just dealt with. You also realize that repeatedly writing the line to compute β for each dependent variable is very tedious (and it is very dangerous, you are bound to make a typo in some of the lines!!!). You therefore decide to write a function `myRegression()` which computes the OLS coefficient β for a generic dependent variable y and a generic matrix X.

(b) Write the function `myRegression()` and call it from the script for one of the dependent variables. Does it do what you expect?

(c) Write a loop to compute all the coefficients systematically (so you only have to write the call of your function once).

9. Monte Carlo Simulation:
You now want to check the standard errors for the estimate that you get using `myRegression()` if you only have a small sample of observations. You decide to check the performance of your estimator using Monte Carlo simulation. That means that you (i) assume a particular relationship between two variables y and x ($y = a + bx + \epsilon$ (the truth) where ϵ is a noise term). (ii) Then you draw random samples for x and ϵ, compute corresponding y, and estimate the relationship using `myRegression()`. You repeat step (ii) T times and each time record your estimates for the coefficient.

(a) Run a Monte Carlo Simulation Experiment by completing the code fragments provided in question 9.

(b) Play around with the sample size of the random samples `Nobs`, the number of repetitions `T` and the variance of the noise `sigma_eps`. What do you see?

(c) Repeatedly execute the Monte Carlo Simulation for the same parameter values. Do your results differ? How can you change that?