

# Value Function Iteration, and Cross-Sectional Simulation using C++/Armadillo: a Simple Example

Gill Segal\*

September 2014

## 1 Model

Consider the following simple economy. There is a mass of firms  $i \in [0, 1]$ . Each firm, indexed by  $i$ , has a capital stock given by  $k_{i,t}$ . The firm's output is subject to two shocks: (1) an aggregate TFP shock  $x_t$ ; (2) an idiosyncratic TFP shock  $z_{i,t}$ . The log of these two shocks follow an AR(1) with Gaussian innovations. The production technology is given by  $y_{i,t} = z_{i,t}x_t k_{i,t}^\alpha$  where  $\alpha < 1$  is the returns to scale. Capital evolves according to:  $k_{i,t+1} = (1 - \delta)k_{i,t} + I_{i,t}$ , where  $I_{i,t}$  is the investment of the firm at time  $t$ , and  $\delta$  is the depreciation rate.

$$\begin{aligned} V(k_{i,t}, z_{i,t}, x_t) &= \max_{k_{t+1}} z_{i,t}x_t k_{i,t}^\alpha - I_{i,t} + \beta E_t V(k_{i,t+1}, z_{i,t+1}, x_{t+1}) \\ k_{i,t+1} &= (1 - \delta)k_{i,t} + I_{i,t} \\ \log(z_{i,t+1}) &= \rho_z \log(z_{i,t}) + \sigma_z \varepsilon_{i,z,t+1} \\ \log(x_{t+1}) &= \rho_x \log(x_t) + \sigma_x \varepsilon_{x,t+1} \end{aligned}$$

Aggregate output, capital and investment, are given by  $Y_t = \int_i y_{i,t}$ ,  $K_t = \int_i k_{i,t}$ ,  $I_t = \int_i I_{i,t}$ .

---

\*The Wharton School, University of Pennsylvania, segalg@wharton.upenn.edu.

## 2 Solution

I solve the above model using value function iteration (VFI). The grids for the shocks  $x$  and  $z$  each have 11 states, and are constructed as in Tauchen (1986). The grid for the endogenous state,  $k$ , has 150 states. In the cross-sectional simulation, I use a cross-section of  $CS = 10,000$  firms, and time-series of  $TS = 10,000$  periods, and I truncate the first 1,000 periods to avoid dependence on initial values. I code two identical versions (in terms of the algorithm used): one in Matlab, and one in C++.

### 2.1 Coding VFI in C++: the Easy Way

Many economists who wish to make the running time of their code faster resolve to C++. The problem is twofold: (1) Many Matlab users do not have sufficient background in C++; (2) Those who code VFI in C++ mostly use their own code, implemented from scratch - this code may be prone to bugs, or may not be implemented efficiently.

One potential solution, is to write your VFI in C++, but rely on the external, and highly-recommended library called ‘Armadillo’<sup>1</sup>. As described at the Armadillo project website, “Armadillo is a high quality C++ linear algebra library, aiming towards a good balance between speed and ease of use; the syntax (API) is deliberately similar to Matlab”. You can find the documentation of Armadillo here: <http://arma.sourceforge.net/docs.html>. It is VERY intuitive for Matlab users. New releases are available periodically. If you are intimidated about downloading Armadillo, and coding your VFI from scratch using Armadillo, the attached code files should be useful for you.

### 2.2 What’s in the Code folder

The code folder contains already an old version of Armadillo 3.6.0. (no need to download or install Armadillo yourself – unless you want the newest version of Armadillo, or want to speed up the code further using Lapack library, which Armadillo uses for some Matrix Algebra operations). It contains also the source files:

- The Matlab code of the VFI is in file: `main_vfi_matlab.m`, and the simulation is in file: `simulate_matlab.m`.
- The C++ code of the VFI is in file: `main.cpp`, and the simulation is in file: `simulateEconomy.cpp`. There are some other utility source files attached, and you can find their functionality in their in-line documentation.

Each line of code in Matlab is Paralleled by an equivalent line of code in C++. To make the equivalence transparent, I divide the code into segments numbered (I, II, III,...). The segment number appears right before the relevant code, both in the C++ files, and in the Matlab files, so the comparison is rather intuitive.

### 2.3 Running the Code

- To run the Matlab code, simply open Matlab and run the file `main_vfi_matlab.m` (found in the “code” folder).
- To run the C++ code, on a Linux/Unix machine, follow the next 3 steps.
  1. Optional: Install Armadillo and Lapack libraries on your machine. The instructions on how to do so, are defined here: <http://arma.sourceforge.net/download.html>.

---

<sup>1</sup>Credits for the Armadillo library can be found here: <http://arma.sourceforge.net/contact.html>

2. Compile the source files:

- (a) If you did not follow step 1 (and rely only on the files attached in the .zip file), go to the directory “code” (attached), and then simply type in the shell (command line):
- ```
g++ -m64 ./armadillo-3.6.0/include -DARMA_NO_DEBUG -O2 -o main.out main.cpp calibration.h
initializeGrids.h initializeGrids.cpp maxIndexByRow.h maxIndexByRow.cpp simulateEconomy.h
simulateEconomy.cpp
```
- (b) If you followed step 1, type:
- ```
g++ -m64 -L ./lapack-3.4.2 -I ./armadillo-3.6.0/include -DARMA_NO_DEBUG -llapack
-lblas -O2 -o main.out main.cpp calibration.h initializeGrids.h initializeGrids.cpp
maxIndexByRow.h maxIndexByRow.cpp simulateEconomy.h simulateEconomy.cpp
```
- (here I assume the Lapack library 3.4.2 is installed under the same folder “code”).

This compiles the files, and produces an executable file named `main.out`.

3. In the shell (command line), type: `main.out`. This runs the code for the VFI and simulation.

- To run the C++ code on a Windows machine, you need to install Armadillo yourself, via the instructions here: <http://arma.sourceforge.net/download.html>. Then, simply compile the files using your favorite C++ compiler.

The output of the Matlab and C++ versions of the code are identical. I simply print the mean and standard deviation moments of aggregate output, capital and investment. These are shown below:

```
Aggregate output. Mean: 1.57, Stdev: 0.078.
Aggregate capital. Mean: 4.28, Stdev: 0.033.
Aggregate investment. Mean: 0.42, Stdev: 0.049.
```

The difference, of course is in the running time. The difference is absolutely staggering. Here are the running-times, on my own ‘not-so-powerful’ machine (Windows 7 64Bit, Processor: Intel i5, RAM: 8GB):

| Section    | Running Time with Matlab | Running Time with C++/Armadillo |
|------------|--------------------------|---------------------------------|
| VFI        | 84.39 sec                | 46.92 sec                       |
| Simulation | 16 min, 37.88 sec        | 14.29 sec                       |
| Overall    | 18 min, 2.26 sec         | 1 min, 1.22 sec                 |

The running time varies between different machines, and operating systems. Yet, the conclusion is always the same: you save a significant amount of time using C++/Armadillo.

You can modify the given C++ code to accommodate your own model, assuming it is also solved using a value function iteration and simulations. Enjoy!