Non stochastic VFI (infinite time) in Matlab

Application to the NGM

Fabrizio Leone
fabrizioleone93@gmail.com

February 3, 2018

Introduction

The code solves the basic non-stochastic neoclassical growth model in Matlab. The problem looks like:

$$\begin{aligned} V(k) &= \max_{k' \in \Gamma(k)} \{u(c) + \beta V(k')\} \\ c &= f(k) + (1-\delta)k - k' \\ k_0 &> 0 \text{ given} \end{aligned}$$

The solution method consists in simple value function iteration. The existence and the uniqueness of the solution are ensured by the *contraction mapping theorem.*

The Code

```
% NGM model with value function iteration
% Code for infinite time DP
% we set a initial guess for our value and we iterate
    forward

% Fabrizio Leone - 26-01-2018

%% 1. Housekeeping
clear all
clc

%% 2. Set parameters
sigma=1;
beta=0.96;
alpha=0.33;
delta=0.04;
kstar=(((1/(alpha*beta))-((1-delta)/alpha)))^(1/(alpha
    -1));

%create a grid for capital
kmin=kstar*.9;
kmax=kstar*1.1;
n=200;
step=(kmax-kmin)/(n-1);
k=(kmin:step:kmax);

%% 3. Set variables

y=k.^alpha; %production function
ytot=repmat(y+(1-delta)*k,n,1);
kprime=repmat(k,n,1);
c=ytot-kprime'; %check how consumption is constructed

%define utility function

if sigma==1
u=log(c);
else
```

```
u=(c.^(1-sigma)-1)/(1-sigma);
end
u(c<0) = -Inf;

% Initialize VFI
V0=zeros(n,1); %initial value function guess
diff=10;
toler=10^(-5);
it=1;
maxit=10^10;


while diff>=toler & it<=maxit %it<=maxit is not really
    necessary

     vp=repmat(V0,1,n);
     v=u+beta*vp;
     [V1 p]=max(v,[],1); %compute value and policy
         function (p): search for the max along each
         column
     value(it,:)=V1; %store the value of each iteration
     policy(it,:)=p;%store the policy of each iteration

     %metric distance
     diff=max(abs(V1'-V0));

     %update value function
     V0=V1';
     it=it+1;
     disp(it)

     % when the loop breaks, V1 will be the value
         function which solves the
     % problem: V_t=V_{t+1}. it is the fixed point.

end

%% 4. Plotting results

figure(1)
plot(k,V1)
```

```
title('Value Function against capital stock')
xlabel('capital stock')
ylabel('value function')
%notice: the value function is increasing in the
    current stock of capital

figure(2)
plot(k,p);
title('Policy function against capital stock')
xlabel('capital stock')
ylabel('value function')
%
% figure(3)
% surf(value)
% title('Value Function surface')
% xlabel('capital today')
% ylabel('capital tomorrow')
% zlabel('value function')

%% 5. Simulation of transition dynamics

    P=100; % arbitrary length for transition path
    capital_index=ones(1,P);
    capital_transition=ones(1,P);
    capital_index(1)=(3); % arbitrary starting point in
        terms of the index
    %capital_index(1)=(150); % set an initial value
        above the ss.
    capital_transition(1)= k(capital_index(1));

    for t=2:P
    capital_index(t)=(p(capital_index(t-1)));%
        evolution in index space
    capital_transition(t)=k(capital_index(t)); %
        evoluation in capital space
    end

figure(4)
plot(capital_transition)
title('Transitional dynamics for capital')
xlabel('time period')
```

```
110 ylabel('capital stock')
```

Explanation

- **Lines 1 to 20:** set parameters values, the steady state capital value of the model and define the capital grid. Make sure to include the ss of capital inside the grid, as done at lines 16 to 20.
- **Lines 24 to 36**: define consumption and available resources. The matrices look like, *assuming there are only* 3 *grid points for* k:

$$k = \begin{pmatrix} k_1 & k_2 & k_3 \end{pmatrix}$$

$$y = \begin{pmatrix} k_1^\alpha & k_2^\alpha & k_3^\alpha \end{pmatrix}$$

$$ytot = \begin{matrix} k_1^\alpha-(1-\delta)k_1 & k_2^\alpha-(1-\delta)k_2 & k_3^\alpha-(1-\delta)k_3 \\ k_1^\alpha-(1-\delta)k_1 & k_2^\alpha-(1-\delta)k_2 & k_3^\alpha-(1-\delta)k_3 \\ k_1^\alpha-(1-\delta)k_1 & k_2^\alpha-(1-\delta)k_2 & k_3^\alpha-(1-\delta)k_3 \end{matrix}$$

$$c = \begin{matrix} k_1^\alpha-(1-\delta)k_1-k_1' & k_2^\alpha-(1-\delta)k_2-k_1' & k_3^\alpha-(1-\delta)k_3-k_1' \\ k_1^\alpha-(1-\delta)k_1-k_2' & k_2^\alpha-(1-\delta)k_2-k_2' & k_3^\alpha-(1-\delta)k_3-k_2' \\ k_1^\alpha-(1-\delta)k_1-k_3' & k_2^\alpha-(1-\delta)k_2-k_3' & k_3^\alpha-(1-\delta)k_3-k_3' \end{matrix}$$

$$u(c) = \begin{matrix} u(k_1^\alpha-(1-\delta)k_1-k_1') & u(k_2^\alpha-(1-\delta)k_2-k_1') & u(k_3^\alpha-(1-\delta)k_3-k_1') \\ u(k_1^\alpha-(1-\delta)k_1-k_2') & u(k_2^\alpha-(1-\delta)k_2-k_2') & u(k_3^\alpha-(1-\delta)k_3-k_2') \\ u(k_1^\alpha-(1-\delta)k_1-k_3') & u(k_2^\alpha-(1-\delta)k_2-k_3') & u(k_3^\alpha-(1-\delta)k_3-k_3') \end{matrix}$$

- **Lines 38 to 44**: Set the initial guess for the value function, the counter, the tolerance and the maximum number of iterations.
- **Lines 46 to 59**: Main loop.
  1. We first define a value function v as a function of the return function u and our initial guess V_0.
  2. We then look for the **maximum in each column**. $V1$ is the value of the maximum in each column, while p tells us the position of the maximum (i.e. which value of k' in the grid maximizes our current utility). The first one is our implied value function, the latter the policy function of each iteration. We then create two

matrices, *value* and *policy* where we store, in each row, the $V1$ and the p of each iteration. When the loop breaks, it means that $V1 = V0$, so we found the fixed point of the iteration. It's worthy of note that the last $V1$ we find is the actual solution, while *value* stores all the value functions we generate by recursive solution.

Notice that this method of searching the maximum on the whole capital grid is inefficient, since many point of the matrix will never arise as a maximum. Looking at the policy function p, we see that the optimal capital choice lies indeed along a diagonal starting at the 9^{th} point of the matrix v.

3. We then compute the distance between V_0 and V_1 according to our preferred metric and prepare for the next iteration, updating the counter and the value function. The distance computed within each iteration serves to tell matlab when to break the loop.

- **Lines 63 to 80**: Plotting results
- **Lines 81 to the end**: Simulate transitional dynamics for capital

1. Set a number of simulation, say P,
2. Set two row matrices of ones, *capital_index* and *capital_transition*, with the same length,
3. Set a random value in the first position of *capital_index* and evaluate *capital_transition* at the value of k corresponding to the random value we chose. ex. if we put a 3 as first value of *capital_index*, then the first value of *capital_transition* must be the third value of the grid k. We can either choose a value below or above the steady state of capital.
4. Within the loop, plug into each position *capital_index* the $P-1$th value of the policy. Then evaluate *capital_transition* at the value of k corresponding to that position. In this way *capital_transition* is a row matrix telling us what is the optimal capital level to choose, given our starting point,
5. Plot *capital_transition* to see the transitional path.