Econ 260D: Numerical Dynamic Optimization

Professor Christopher Costello

April 17, 2023

1 Numerical Solutions by Value Function Iteration

While simple dynamic problems may be solved analytically, many realistic problems are too cumbersome to solve analytically. In such cases, one approach is to employ numerical optimization techniques. Even when you seek analytical solutions to a problem, numerical approaches can be helpful in building intuition and checking analytical results. Here we introduce the simplest method for numerically solving dynamic programming problems, called "Value Function Iteration" or VFI. Note that it can be used for deterministic or stochastic problems and that states and/or controls could be vectors. If you have a continuous state or control space, you must modify VFI slightly.

The intuition behind VFI is as follows: Start at the end of time (or if the problem is infinite-horizon, start a long distance into the future). Assume there is no future. For many possible values of the period-T state, numerically optimize and thus solve for the value function in period T. Step back a period, invoke the period-T value function, and solve for the period $T-1$ value function, etc. Repeat this process until you reach the present period. Often, the main interest of the researcher is the *policy function*, not the *value function*. The policy function often converges to a single function within 10 steps. The value function will only converge at the rate of discount (so you might have to step back for 20-30 periods, depending on the discount rate – fewer periods for high discount rates).

If your problem is an infinite horizon problem, then instead of "stepping back in time", think of this process as "iterating on the value function". If your problem is truly a finite horizon problem with T periods, then in principle you need to step back T times, but if the policy function converges in fewer than T periods, this is practically not necessary. For example, suppose $T = 100$ and the policy function converges by stepping back from 100, 99, ..., 95. Then you know that the policy

function for the first 95 years is the same, and then it diverges to the functions derived in the backward induction.

The recipe for Value Function Iteration is as follows:

1. Discretize the state and control space. Or, you can descretize the state space and allow the control to be continuous. I will follow this latter approach.

2. "Guess" or insert (if known) the correspondence: $V_{T+1}(x_{T+1})$. If you assume no future, this would just be set equal to 0.

3. Step back 1 period (to period T), and calculate:

$$V_T(x_T) = \max_{u_T} \left[\pi(x_T, u_T) + \delta V_{T+1}(G(x_T, u_T)\right] \tag{1}$$

   for every possible value, x_T. This gives the *policy function*, $u_T^*(x_T)$ and the value function $V_T(x_T)$.

4. Step back 1 period (to period $T-1$), obtain the *policy function*, $u_{T-1}^*(x_{T-1})$ and the value function $V_{T-1}(x_{T-1})$. Repeat until you get to time 0.

This procedure will give a set of $T+1$ value functions and a set of $T+1$ policy functions. For infinite horizon problems, we are typically interested in the properties of the policy function.

2 A Renewable Resource Example

A renewable resource stock x_t is harvested to maximize profit. Harvest in period t is h_t and the current period profit from harvest is: $\Pi(h) = \alpha h - \beta h^2$, which corresponds to a downward sloping linear demand curve. The discount factor is δ and the equation of motion (i.e. the growth of the resource stock) is:

$$x_{t+1} = x_t + r x_t \left(1 - \frac{x_t}{K}\right) - h_t \tag{2}$$

for scalar parameters r and K. The dynamic programming equation is:

$$V_t(x_t) = \max_{h_t} \Pi(h_t) + \delta V_{t+1}(x_t + r x_t \left(1 - \frac{x_t}{K}\right) - h_t) \tag{3}$$

Again, the problem is that the value function is not known.

The pseudo-code for solving this problem is as follows:

1. Set all parameters (including T).

2. Discretize the state space into N equally spaced values on $[0, K]$, you can make this grid finer by increasing N.

3. Set the $T+1$ value function to zero (so V_{T+1} is an $Nx1$ vector of zeros, each one corresponding to a different level of the stock).

4. Loop backwards over time, going from T to 1.

   (a) Loop over the N possible stock values

      i. Write a function that takes the following inputs: (1) harvest, (2) stock, (3) next period's value function and returns the payoff.

      ii. Use an optimization routine to find the optimal harvest for any given stock and next period value function.

   (b) End the loop over stock

5. End the loop over time.

3 R Code

Here is the actual R code for solving this problem:

```
rm(list = ls(all = TRUE))
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

library(ggplot2)   # Data visualization
library(tidyr)   # Tidy data management
library(dplyr)
library(cowplot)

sizex = 100 #size of the state grid
T=10 #time horizon for backward induction

a=20
b=.1
delta=1/1.2
r=.8
K=100
small=K/1000

xgrid = seq(small,K,length.out=sizex)
```

```
f = function(h,x)
{
  xnext = x + r*x*(1-x/K) - h
}

pi = function(h)
{
  profit = a*h-b*h^2
}

Payoff = function(h,x,V)
{
  #xnext = max(small,f(h,x))
  xnext = f(h,x)
  Vnext = spline(x=xgrid,y=V,xout=xnext)
  negout = -(pi(h) + delta*Vnext$y)
  return(negout)
}

DFall = data.frame()
Vnext = vector()
V = seq(0,0,length.out=sizex)

#Try payoff function
z=Payoff(.17992,.1,V)

for(t in T:1)
{
  print(t)
  for(i in 1:sizex)
  {
    x = xgrid[i]
    guess = x/2
    low = 0 #lower bound on harvest
    high = x + r*x*(1-x/K) #upper bound on harvest
    Thing = optim(par=guess,fn=Payoff,lower=low,upper=high,x=x,V=V,method='L-BFGS-
    hstar = Thing$par
    Vstar = -Thing$value
    Vnext[i] = Vstar
    DFnow = data.frame(time=t,x=x,hstar=hstar,Vstar=Vstar)
```

```
    DFall = bind_rows(DFall,DFnow)
  }
  V = Vnext
}

Ph = ggplot(data=DFall) +
  geom_path(aes(x=x,y=hstar,color=factor(time)),size=1.3) +
  xlab("Stock, x") +
  ylab("Harvest, h") +
  scale_color_discrete(name="Year") +
  theme_bw() +
  theme(legend.position = "none")
Ph

PV = ggplot(data=DFall) +
  geom_path(aes(x=x,y=Vstar,color=factor(time)),size=1.3) +
  xlab("Stock, x") +
  ylab("Value Function, V") +
  scale_color_discrete(name="Year") +
  theme_bw()
#PV

#Forward Simulation
DFopt = DFall %>% filter(time==1)
hpol = DFopt$hstar
xpol = DFopt$x
xsim=vector()
hsim=vector()

xsim[1]=K/10
Tsim = seq(1,20)

for(tt in Tsim)
{
  Thing = spline(x=xpol,y=hpol,xout=xsim[tt])
  hsim[tt] = Thing$y
  if(tt<max(Tsim))
  {
     xsim[tt+1] = f(h=hsim[tt],x=xsim[tt])
  }
```

```
}

DFsim = data.frame(time=Tsim,xsim=xsim,hsim=hsim)

Pxsim = ggplot(data=DFsim) +
  geom_line(aes(x=time,y=xsim),color="blue",size=1.5) +
  xlab("Time") +
  ylab("Stock, x") +
  theme_bw()
#Pxsim

Phsim = ggplot(data=DFsim) +
  geom_line(aes(x=time,y=hsim),color="blue",size=1.5) +
  xlab("Time") +
  ylab("Harvest, h") +
  theme_bw()
#Phsim

Pall = plot_grid(Ph,PV,Pxsim,Phsim,ncol=2,nrow=2)
Pall

ggsave(filename="../Fig1.png",plot=Pall,width=6,height=5,unit="in")
```

Which produces the following plot:

Figure 1: Policy function, value function, and forward simulations.