

Macro III

Homework 1- Finite Dynamic Programming

Consider the problem of an agent who lives J periods and decides how much of a cake to eat in each period over the lifetime. The cake can be stored over time without decay.

$$V_1^*(x_1) \equiv \max \sum_{j=1}^J \beta^{j-1} u(c_j) \\ \text{subject to (1) } c_j + x_{j+1} = x_j \text{ and (2) } c_j, x_j \geq 0$$

It is not difficult to write down the solution to this problem with some convenient choice for $u(c)$. The solution is as follows (when $u(c) = \ln(c)$):

$$V_1^*(x) = \sum_{j=1}^J \beta^{j-1} \ln(c_j^*) = C_1 + C_2 \ln(x)$$

$$c_j^* = \beta^{j-1} x / \sum_{j=1}^J \beta^{j-1}$$

$$C_1 = \sum_j \beta^{j-1} [\ln(\beta^{j-1} / \sum_{j=1}^J \beta^{j-1})]$$

$$C_2 = \sum_{j=1}^J \beta^{j-1}$$

1. Write a program to solve the finite dynamic programming version of this problem indicated below.

$$V_j(x) = \max_{y \in \Gamma(x)} u(x - y) + \beta V_{j+1}(y)$$

$$\Gamma(x) = \{y \in X : y \leq x\}$$

$$X = \{x_1, \dots, x_N\}$$

2. Compute solutions when you set $J = 10$ and $\beta = 0.9$. Set the state space $X = \{x_1, \dots, x_N\}$ so that $N = 101$ and $x_i = 10(i - 1)/(N - 1)$. Thus, the state space has 101 evenly spaced points on the interval $[0, 10]$.
3. Graph the function $V_1(x)$ and $V_1^*(x)$ on the same graph.

4. Graph the computed optimal decision rule and the actual optimal decision rule $y_j(x)$ on the same graph when $j = 1$.
5. Graph the function $V_1(x)$ and $V_1^*(x)$ when $N = 201$ and the gridpoints x_i are evenly spaced according to the rule used above.

SUGGESTIONS:

1. Create arrays $U(NX, NY)$, $V(NX, NJ)$, $Y(NX, NJ)$ where NX is the number of gridpoints on the state, NY is the number of gridpoints on the control and NJ is the number of periods. The U array is the return function, the V array is the value function and the Y array is the decision rule.

[Note: (i) $NX = NY$, (ii) $NX = 101$, (iii) $NJ = J + 1$ - see why below]

2. Initialize the value function in the period $J + 1$ to zero. Thus, the last column in the V matrix are a bunch of zeros.

3. The U array need only be computed once at the start of the program. When the state and controls would imply that consumption is zero or negative, then a standard trick is to set the return function to a sufficiently small (negative) value. If you set this number correctly, then you need not impose restrictions on feasible controls in the maximization step. This makes programming simpler -see below.

4. After computing the U array and initializing the value function array V as indicated above, the remaining program consists essentially of two DO LOOPS. There is an outside DO LOOP over the time index and an inside DO LOOP over the state index. Between these two DO LOOPS there is a maximization operation. The maximization operation can be done with a single built-in command on an array of numbers in most computer languages. So for each time period you may want to apply a vector addition command that computes the value of all possible decisions in a period and then maximize over the elements in this vector. Of course, you can ignore this suggestion and program this max operation yourself.