Solving Dynamic Oligopoly Game

Problem Set 2

Empirical Industrial Organization 2025 Spring

Zixuan

March 14, 2025

1 Tasks

1.1 State space

Decode See Algorithm 1. Let's look at an example of decoding code 8 (index-0 based) to state ntuple $(2, 2, 1)$.

$$\begin{matrix} 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & \dots \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 2 & 2 & 2 & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 2 & \dots \end{matrix}$$

The trick is to make use of the following binomial coefficient matrix.

$$\begin{matrix} 0 & 1 & & & & & \\ 0 & 1 & 1 & & & & \\ 0 & 1 & 2 & 1 & & & \\ 0 & 1 & 3 & 3 & 1 & & \\ 0 & 1 & 4 & 6 & 4 & 1 & \\ 0 & 1 & 5 & 10 & 10 & 5 & 1 \end{matrix}$$

We have 3 firms. For the 1st firm, we start by the value at the 3rd row. Compare the code 8 to this value, deduct the value from code, move to the bottom right by one step. Stop when the code is smaller than the value in the matrix. Then we find the state for the first position by the number of times we have moved. For the 2nd firm, we start by the alue at the 2nd row, etc.

Encode See Algorithm 2. Similarly, encoding $(2, 2, 1)$ to code 8 is to move along the binom matrix. Summing up 1,3,1,2,1 to get 8.

Algorithm 1 Decode an integer state code into a weakly descending n-tuple

```
function DECODE(code, nfirms, binom)
    ntuple ← zeros(nfirms)                    ▷ Initialize output n-tuple
    for i = 0 to nfirms − 1 do
        row ← nfirms - i - 1
        col ← 1
        while code ≥ binom[row, col] do
            code ← code - binom[row, col]
            row ← row + 1
            col ← col + 1
        end while
        ntuple[i] ← col - 1
    end for
    return ntuple
end function
```

Algorithm 2 Encode a weakly descending n-tuple into an integer state code

```
function ENCODE(ntuple, nfirms, binom)
    code ← 0                                  ▷ Initialize state code
    for i = 0 to nfirms − 1 do
        for j = 0 to ntuple[i] − 1 do
            code ← code + binom[nfirms − i − 1 + j, 1 + j]
        end for
    end for
    return code
end function
```

1.2 Equilibrium computation

Continuation value Given a state ntuple ω, and the position j we calculate the `val_up` when the firm j receives a positive $\tau_j = 1$ and `val_up` when the firm j receives $\tau_j = 0$.

Function operator Given a set of value function $V(\omega) = \{V_1(\omega), \ldots, V_N(\omega)\}$ and policy function $x(\omega) = \{x_1(\omega), \ldots, x_N(\omega)\}$, return the new value function and policy function. This is essentially the operator which satisfies the contraction mapping theorem.

Remark. we need to take into account two different state ntuple ω (no entry) and ω_e (with entry). Therefore, we get

1. `val_up`, `val_stay`
2. `val_up_e`, `val_stay_e`

Then the actual `val_up_both` for both cases (entry or not) is taking expectation over entry. Similaryly for `val_stay_both`.

Remark. The value of staying is the following, which is compared to scrap value ϕ to determine whether to exit.

$$\text{Current profit} - \text{Investment} + \beta \left[\Pr(\tau_j = 1)\text{val\_up\_both} + (1 - \Pr(\tau_j = 1))\text{val\_stay\_both}\right]$$

1.3 Simulation

Code The detail of the simulation is described in 3. The simulation is run for 10000 periods.The initial state is $(6, 0, 0)$

Results The statistics of the simulation results are shown in Table 1. I also plot the trajectory of **firms count** and **average investment per period**. Since the number of time period is too large, for visualisation purpose I restrict myself to the first and last 100 periods. (It seems that investment is converging to zero...)

	Firms Count	Average Investment
Baseline	2.2799	0.1904
Low Entry Cost	2.6627	0.1692

Table 1: Simulation Results

Algorithm 3 Industry Evolution Simulation

```
for t = 1 to T do                                              ▷ Encode the current state
    stateCode ← qencode(currentState, etable, multfac)
    stateHistory[t, :] ← currentState
    firmsCountHistory[t] ← ∑(currentState > 0)
                              ▷ Let potential entrant make decision (but does not enter yet!)
    for i = 1 to N do
        if currentState[i] = 0 then
            entryProb ← isentry[stateCode]
            entry ∼ Binomial(1, entryProb)
            if entry then
                entryIndex ← i        ▷ Get the position of the vacancy where one entrant enters.
            end if
            break                                              ▷ Only one entrant can enter.
        end if
    end for
                                                   ▷ Let firms make exit decision and exit.
                                                   ▷ The currentState is not in descending order
            ▷ This is necessary when we retrieve value and policy from newvalue and newx
    sortedIndex ← argsort(currentState)[:: −1]
    for j = 1 to N do
        if newvalue[stateCode, :][sortedIndex][j] < φ then
            currentState[j] ← 0
            currentState[currentState ≤ currentState[j]] ← 0
        end if
    end for
                                                                     ▷ Investment decision
    investmentPolicyolicy ← (currentState > 0) ∗ (newx[stateCode, :][sortedIndex])
    investmentHistory[t, :] ← investmentPolicyolicy                    ▷ Let entrant enter!
    if entry then
        currentState[entryIndex] ← entryLevel
    end if
                                                                ▷ Simulate individual shocks
    Pr(τ_j = 1) ← a·investmentPolicyolicy / (1+a·investmentPolicyolicy)
    τ_j ∼ Binomial(1, Pr(τ_j = 1))
                                                                   ▷ Update industry state
    currentState ← max(min(currentState + individual_shocks − ν[t], kmax), 0)
end for
```

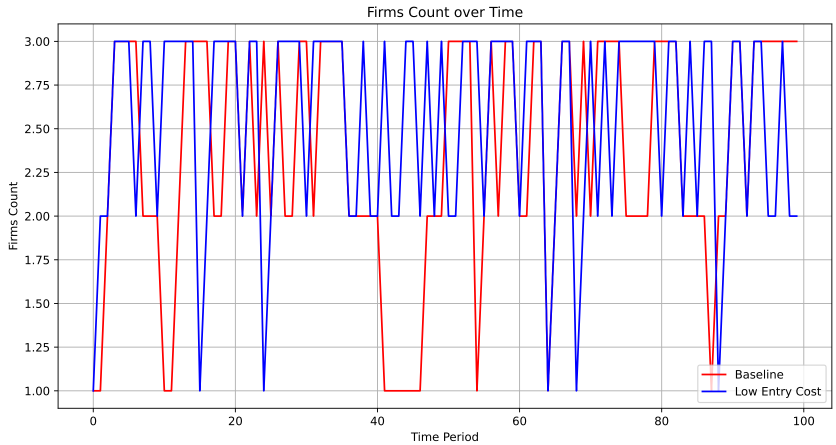

Figure 1: Firms Count Trajectory (First 100 periods)

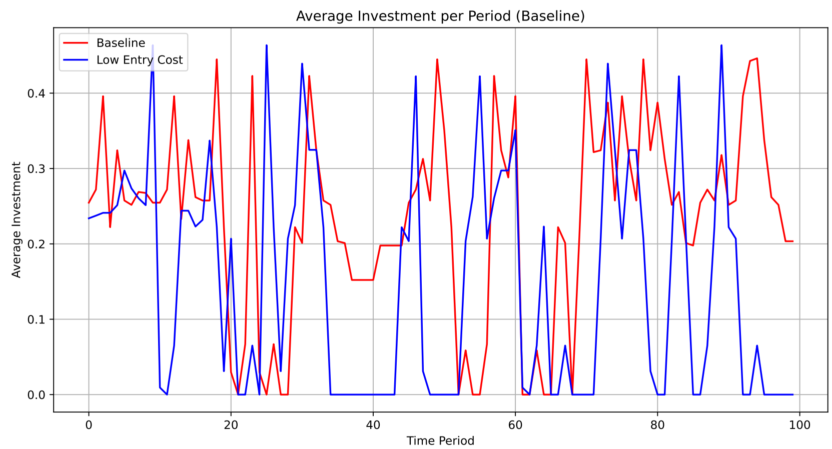

Figure 2: Average Investment Trajectory (First 100 periods)

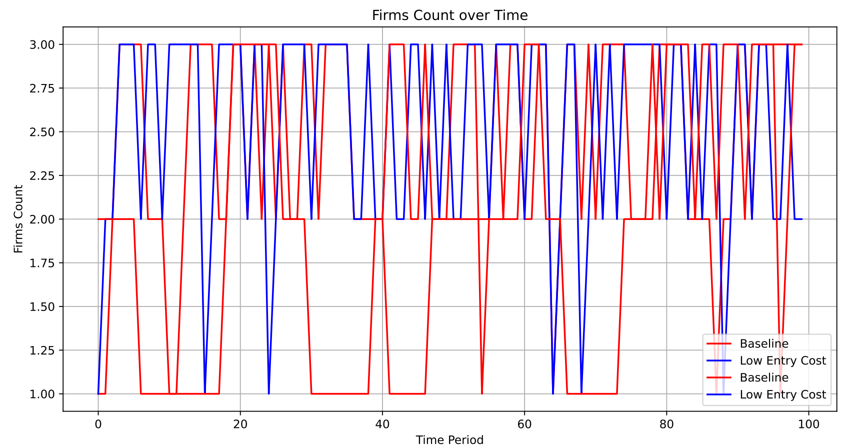

Figure 3: Firms Count Trajectory (Last 100 periods)

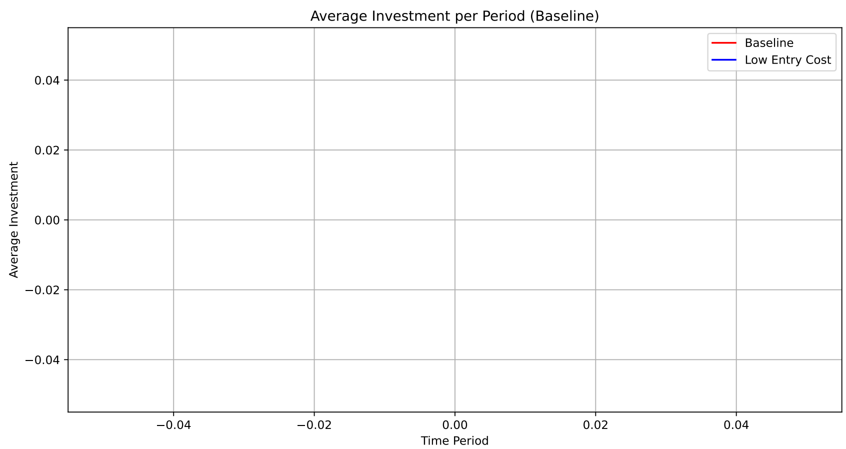

Figure 4: Average Investment Trajectory (Last 100 periods)