Solving Dynamic Oligopoly Game

Problem Set 2
Empirical Industrial Organization 2025 Spring

Zixuan

March 14, 2025

1 Tasks

1.1 State space

decode Let's see 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 frim, 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 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.

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 $val_u p$ when the firm j receives $\tau_j = 0$.

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
```

Algorithm 3 Compute continuation value for moving up and staying at the same efficiency level

```
function CalcVal(place, w, x, k, oldvalue, etable, multfac, two_n, kmax, nfirms,
        mask, delta, a)
    z1 ← zeros(nfirms)                                        ▷ Lower bound (0)
    z2 ← full(nfirms, kmax)                                   ▷ Upper bound (kmax)
                                          ▷ Adjust "mask" based on firm's position
    if nfirms > 1 then
        zeros_row ← zeros(1, two_n)
        if place = 1 then
            locmask ← stack([zeros_row, mask])
        else if place = nfirms then
            locmask ← stack([mask, zeros_row])
        else
            locmask ← stack([mask[:place - 1], zeros_row, mask[place - 1:]])
        end if
    else
        locmask ← zeros(1,1)
    end if
                                                  ▷ Modify investment and state
    x[place − 1] ← 0                                ▷ Own investment is set to zero
    w[place − 1] ← k                               ▷ Own efficiency level is updated
    justone ← zeros(nfirms)
    justone[place − 1] ← 1                               ▷ Mark this firm's position
    p_up ← (a · x)/(1 + a · x)                           ▷ Probability of moving up
    val_up, val_stay ← 0
    for i = 0 to two_n − 1 do
        probmask ← ∏(locmask[:, i] · p_up + (1 − locmask[:, i]) · (1 − p_up))
        d ← w + locmask[:, i]                                     ▷ Private shock
        temp ← stack([d, justone])
        temp ← sort_descending(temp)
        d ← temp[:, 0]
        e ← d − 1                                               ▷ Aggregate shock
        e ← max(e, z1)
        d ← min(d, z2)
        pl1 ← argmax(temp[:, 1])                      ▷ Find "place" in new state
                            ▷ Update expected value for staying at efficiency level
        val_stay+ = ((1 − delta) · oldvalue[qencode(d, etable, multfac), pl1]
          +delta · oldvalue[qencode(e, etable, multfac), pl1]) · probmask
                                                  ▷ Compute value when moving up
        new_d ← d
        new_d[pl1] ← new_d[pl1] + 1
        new_e ← new_d − 1
        new_e ← max(new_e, z1)
        new_d ← min(new_d, z2)
        val_up+ = ((1 − delta) · oldvalue[qencode(new_d, etable, multfac), pl1]
          +delta · oldvalue[qencode(new_e, etable, multfac), pl1]) · probmask
    end for
    return (val_up, val_stay)
end function
```

1.3 Simulation

References