Model Setup

Suppose that in each time period, a set of firms producing homogeneous products play a simultaneous move Nash quantity-setting game. The market-level inverse demand curve in market in time t is given by $p_t = 10 - Q_t + x_t$ where Q_t is the total quantity produced by all the firms, and x_t is a demand shifter. Assume that all firms have 1) a marginal cost of 0, and 2) a fixed cost of operation of 5 per-period. Next, consider the dynamic entry and exit process. First, assume that the demand shifter x_t takes on 3 possible values $\{-5, 0, 5\}$, and that this evolves over time according to the following first order Markov transition matrix (e.g. given $x_t = 5$, the probability that $x_{t+1} = -5$ is 0.2):

	x_{t+1}		
	-5	0	5
$x_t = -5$	0.6	0.2	0.2
$x_t = 0$	0.2	0.6	0.2
$x_t = 5$	0.2	0.2	0.6

For the exit process, assume that in each period, each active firm gets a stochastic sell off value $\mu_{it} \sim N(\mu, \sigma_\mu^2)$. This value μ_{it} is assumed to be private information for the firm, and it is assumed to be i.i.d. across firms and time. If a firm decides to exit at period t, it receives current payoff of just μ_{it} (note that this is slightly different than the model in class) and is then out of the market forever. This implies that the value function for an active firm is

$$V(N_t, x_t, \mu_{it}) = \max_{d_{it} \in \{0,1\}} \{\mu_{it}, \pi(N_t, x_t) + 0.9\mathbb{E}[V(N_{t+1}, x_{t+1}, \mu_{it+1})|N_t, x_t, d_{it} = 1]\}$$

where $d_{it} = 0$ if the firm exits and $d_{it} = 1$ if the firm remains in the market. For the entry process, assume that in each period, there is one "potential entrant" into the market. This potential entrant is endowed with a stochastic entry cost $\gamma_{it} \sim N(\gamma, \sigma_\gamma^2)$ that they must pay to enter the market (γ_{it} is also private information to the entrant and assumed iid across time). If the firm decides to enter at t, their single period payoffs at t are $-\gamma_{it}$ - i.e. they only begin to produce and make profits in the next period. If the firm decides not to enter at t, they disappear from the market forever. This implies that the value function for a potential entrant is:

$$V^E(N_t, x_t, \gamma_{it}) = \max_{e_{it} \in \{0,1\}} \{0, -\gamma_{it} + 0.9\mathbb{E}[V(N_{t+1}, x_{t+1}, \mu_{it+1})|N_t, x_t, e_{it} = 1]\}$$

where $e_{it} = 1$ if the firm enters and $e_{it} = 0$ if the firm does not enter the market. We will focus attention on symmetric Markov-Perfect Nash Equilibria of this game that take the following "cutoff" form:

$$\text{For an incumbent firm:} \quad d_{it} = \begin{cases} 0 & \text{if } \mu_{it} > \bar{\mu}(N_t, x_t) \\ 1 & \text{if } \mu_{it} \leq \bar{\mu}(N_t, x_t) \end{cases}$$

$$\text{For a potential entrant:} \quad e_{it} = \begin{cases} 0 & \text{if } \gamma_{it} > \bar{\gamma}(N_t, x_t) \\ 1 & \text{if } \gamma_{it} \leq \bar{\gamma}(N_t, x_t) \end{cases}$$

Assume that the maximum number of firms in the market is 5 (i.e. when $N_t = 5$, assume that the entry cost γ_{it} is ∞ with probability 1). This means that there are 18 possible values of the state i.e. $N_t \in \{0, 1, 2, ..., 5\}$, $x_t \in \{-5, 0, 5\}$. Therefore, equilibrium can be summarized by 36 cutoff numbers, $\bar{\mu}(N_t, x_t)$ at each of the 18 possible states, and $\bar{\gamma}(N_t, x_t)$ at each of the 18 possible states. Actually, note that only 30 of these numbers are relevant, since in states where $N_t = 0$ there are no incumbents (so $\bar{\mu}(0, x_t)$ is irrelevant), and in states where $N_t = 5$ there can be no entry (so $\bar{\gamma}(5, x_t)$ is irrelevant). Defining $\bar{V}(N_t, x_t) = \int V(N_t, x_t, \mu_{it}) p(d\mu_{it})$, we can write the value functions as:

$$V(N_t, x_t, \mu_{it}) = \max_{d_{it} \in \{0,1\}} \left\{ \mu_{it}, \pi(N_t, x_t) + 0.9 \sum_{N_{t+1}} \sum_{x_{t+1}} \bar{V}(N_{t+1}, x_{t+1}) \Pr(x_{t+1}|x_t) \Pr(N_{t+1}|N_t, x_t, d_{it} = 1) \right\}$$

$$V^E(N_t, x_t, \gamma_{it}) = \max_{e_{it} \in \{0,1\}} \left\{ 0, -\gamma_{it} + 0.9 \sum_{N_{t+1}} \sum_{x_{t+1}} \bar{V}(N_{t+1}, x_{t+1}) \Pr(x_{t+1}|x_t) \Pr(N_{t+1}|N_t, x_t, e_{it} = 1) \right\}$$

Note that $\bar{V}(N_t, x_t)$ is not defined when $N_t = 0$ (this follows because $V(N_t, x_t, \mu_{it})$ is not defined at $N_t = 0$, i.e. there is no value function for an incumbent at $N_t = 0$ because there is no incumbent with $N_t = 0$).

Problem 1. *Given these cost and demand primitives, derive a equation expressing the Nash-equilibrium single period profits per firm, as depending on the total number of firms N_t and the demand shifter x_t. In other words, derive the formula for $\pi(N_t, x_t)$. This formula will be useful for the computations later.*

Solution. In the Cournot game with N_t symmetric firms, each firm chooses quantity q_i to maximize $q_i(10 - q_i - Q_{-i} + x_t) - 5$, where Q_{-i} denotes the total quantity produced by other firms. The first-order condition is $10 - 2q_i - Q_{-i} + x_t = 0$. In symmetric Nash equilibrium, all firms produce the same quantity q^*, so $Q_{-i} = (N_t - 1)q^*$. Substituting yields $10 - 2q^* - (N_t - 1)q^* + x_t = 0$, which implies $q^* = (10 + x_t)/(N_t + 1)$. The equilibrium price equals $p^* = 10 - N_t q^* + x_t = (10 + x_t)/(N_t + 1)$, which equals q^* given zero marginal cost. Per-firm profits are therefore $\pi(N_t, x_t) = p^* \cdot q^* - 5 = [(10 + x_t)/(N_t + 1)]^2 - 5$. □

Problem 2. *Why does a cutoff equilibrium make intuitive sense in this setup?*

Solution. A cutoff equilibrium is intuitive because the continuation value from staying,

$$\pi(N_t, x_t) + \beta \, \mathbb{E}\big[V(N_{t+1}, x_{t+1}) \mid N_t, x_t, d = 1\big],$$

is independent of the idiosyncratic draw μ_{it}, while the exit payoff equals μ_{it} itself. Because one function (the value of staying) is flat in μ_{it} and the other (the value of exiting) is strictly increasing in μ_{it}, their payoff schedules can cross at most once. This guarantees the unique indifference cutoff

$$\bar{\mu}(N_t, x_t) \;=\; \pi(N_t, x_t) + \beta \, \mathbb{E}\big[V(N_{t+1}, x_{t+1}) \mid N_t, x_t, d = 1\big],$$

which represents the opportunity cost of remaining in the market. Similarly, $\bar{\gamma}(N_t, x_t)$ denotes the maximum entry cost a prospective entrant will pay to access future profits.

Since μ_{it} and γ_{it} are normally distributed with full support, the equilibrium involves smooth adjustment of entry and exit probabilities,

$$p(d=1 \mid N_t, x_t) = \Phi\left(\frac{\bar{\mu}(N_t, x_t) - \mu}{\sigma_\mu}\right), \qquad p(e=1 \mid N_t, x_t) = \Phi\left(\frac{\bar{\gamma}(N_t, x_t) - \gamma}{\sigma_\gamma}\right).$$

Computationally, cutoff equilibria reduce the infinite-dimensional strategy space to one cutoff per state, making the problem tractable while still allowing heterogeneous firm decisions. □

Problem 3. *Describe (at an intuitive level) what $\bar{V}(N_t, x_t)$ measures. How does this relate to the alternative specific value functions of Rust (1987)?*

Solution. The quantity $\bar{V}(N_t, x_t) = \int V(N_t, x_t, \mu_{it})\, p(d\mu_{it})$ is the incumbent's expected continuation value before observing its idiosyncratic exit shock. Because the stay payoff $\pi(N_t, x_t) + \beta\, \mathbb{E}[V_{t+1}]$ is constant in μ_{it} while the exit payoff equals μ_{it}, we obtain

$$\bar{V}(N_t, x_t) = \Phi(z)\,[\pi(N_t, x_t) + \beta\, \mathbb{E}[V_{t+1}]] + [1 - \Phi(z)]\, \mathbb{E}[\mu_{it} \mid \mu_{it} > \bar{\mu}(N_t, x_t)], \quad z = \frac{\bar{\mu}(N_t, x_t) - \mu}{\sigma_\mu}.$$

This captures the option value of staying: the firm retains the right to exit when μ_{it} is high.

In Rust (1987), the choice-specific value is $v_a(s) = u(s, a) + \beta\, \mathbb{E}[V(s') \mid s, a]$ and the ex-ante value is $V(s) = \mathbb{E}_\varepsilon[\max_a \{v_a(s) + \varepsilon_a\}]$. With Type-I extreme value errors, $V(s) = 0.5772 + \log\left(\sum_a e^{v_a(s)}\right)$ and $p(a \mid s) = e^{v_a(s)} / \sum_{a'} e^{v_{a'}(s)}$.

In the exit model with normally distributed shocks,

$$\bar{V}(N_t, x_t) = \mathbb{E}[\max\{\mu_{it},\, \pi(N_t, x_t) + \beta\, \mathbb{E}[V_{t+1}]\}], \qquad p(d=1 \mid N_t, x_t) = \Phi(z).$$

In both settings, the integrated value function (either $\bar{V}$ or V) summarizes continuation values after integrating out private shocks. □

Problem 4. *Assume that the parameters of the model take the values: $\gamma = 5$, $\sigma_\gamma^2 = 5$, $\mu = 5$, and $\sigma_\mu^2 = 5$. Use the following iterative process to solve for the equilibrium (let's assume at the moment that it is unique). Note that this process is related to, but slightly different than the procedure I outlined in class:*

1. *Guess $\bar{\mu}(N_t, x_t)$, $\bar{\gamma}(N_t, x_t)$, and $\bar{V}(N_t, x_t)$ (note: this is just 3 vectors of 15 numbers, since $\bar{\mu}(N_t, x_t)$ and $\bar{V}(N_t, x_t)$ are not defined when $N_t = 0$, and $\bar{\gamma}(N_t, x_t)$ is not defined when $N_t = 5$)*

2. *Using $\bar{\mu}(N_t, x_t)$ and $\bar{\gamma}(N_t, x_t)$, compute the transition matrices $\Pr(N_{t+1} | N_t, x_t, d_{it} = 1)$ and $\Pr(N_{t+1} | N_t, x_t, e_{it} = 1)$. There should be six of these matrices (for $d_{it} = 1$, there should be 1 for each of the 3 x_t's, and for $e_{it} = 1$ there should be one for each of the 3 x_t's). Why are the two matrices different (for a given x_t)?*

3. *Compute*

$$\Psi_1(N_t, x_t) = 0.9 \sum_{N_{t+1}} \sum_{x_{t+1}} \bar{V}(N_{t+1}, x_{t+1}) \Pr(x_{t+1} | x_t) \Pr(N_{t+1} | N_t, x_t, d_{it} = 1)$$

$$\Psi_2(N_t, x_t) = 0.9 \sum_{N_{t+1}} \sum_{x_{t+1}} \bar{V}(N_{t+1}, x_{t+1}) \Pr(x_{t+1} | x_t) \Pr(N_{t+1} | N_t, x_t, e_{it} = 1)$$

4. *Solve for "new" optimal cutoffs at each state using:*

$$\mu'(N_t, x_t) = \pi(N_t, x_t) + \Psi_1(N_t, x_t)$$
$$\gamma'(N_t, x_t) = \Psi_2(N_t, x_t)$$

Why do these equations tell us the new optimal cutoffs (given the computed values of $\Pr(N_{t+1}|N_t, x_t, d_{it} = 1)$ *and* $\Pr(N_{t+1}|N_t, x_t, e_{it} = 1)$*)?*

5. *Solve for "new"* $\bar{V}'(N_t, x_t)$ *at each state using:*

$$\bar{V}'(N_t, x_t) = \int V(N_t, x_t, \mu_{it}) p(d\mu_{it}) =$$

$$\int \max_{d_{it} \in \{0,1\}} \left\{ \mu_{it}, \pi(N_t, x_t) + 0.9 \sum_{N_{t+1}} \sum_{x_{t+1}} \bar{V}(N_{t+1}, x_{t+1}) \Pr(x_{t+1}|x_t) \Pr(N_{t+1}|N_t, x_t, d_{it} = 1) \right\} p(d\mu_{it})$$

$$= \int \max_{d_{it} \in \{0,1\}} \{\mu_{it}, \pi(N_t, x_t) + \Psi_1(N_t, x_t)\} p(d\mu_{it})$$

$$= \left[1 - \Phi\left(\frac{\pi(N_t, x_t) + \Psi_1(N_t, x_t) - \mu}{\sigma_\mu}\right)\right] \left[\mu + \sigma_\mu \cdot \frac{\phi\left(\frac{\pi(N_t, x_t) + \Psi_1(N_t, x_t) - \mu}{\sigma_\mu}\right)}{1 - \Phi\left(\frac{\pi(N_t, x_t) + \Psi_1(N_t, x_t) - \mu}{\sigma_\mu}\right)}\right]$$
$$+ \Phi\left(\frac{\pi(N_t, x_t) + \Psi_1(N_t, x_t) - \mu}{\sigma_\mu}\right) [\pi(N_t, x_t) + \Psi_1(N_t, x_t)]$$

where ϕ *and* Φ *are the standard normal pdf and cdf. Where is the last equality coming from?*

6. *With the new* $\mu'(N_t, x_t)$*,* $\gamma'(N_t, x_t)$*, and* $\bar{V}'(N_t, x_t)$*, go back to step 1.*

7. *Iterate procedure until convergence.*

Solution. Step 2 computes transition matrices that differ because they condition on distinct events. For incumbents who stay, N_{t+1} depends on how many of the other $N_t - 1$ incumbents stay (a binomial with probability $\Phi((\bar{\mu}(N_t, x_t) - \mu)/\sigma_\mu)$), plus the focal firm, and potential entry (a Bernoulli with probability $\Phi((\bar{\gamma}(N_t, x_t) - \gamma)/\sigma_\gamma)$). For entrants, N_{t+1} equals the number of staying incumbents plus one. Step 4 identifies optimal cutoffs via indifference: $\bar{\mu}(N_t, x_t) = \pi(N_t, x_t) + \Psi_1(N_t, x_t)$ and $\bar{\gamma}(N_t, x_t) = \Psi_2(N_t, x_t)$. Step 5 exploits the truncated-normal formula: $\bar{V}(N_t, x_t) = \Phi(z)\, \bar{\mu}(N_t, x_t) + [1 - \Phi(z)] \left[\mu + \sigma_\mu \frac{\phi(z)}{1 - \Phi(z)}\right]$, where $z = (\bar{\mu}(N_t, x_t) - \mu)/\sigma_\mu$, and where $\phi(z)/(1 - \Phi(z))$ denotes the inverse Mills ratio. □

Problem 5. *Try resolving for the equilibrium (question 4) starting with 5 different guesses of the initial values of* $\mu(N_t, x_t)$*,* $\gamma(N_t, x_t)$*, and* $\bar{V}(N_t, x_t)$*. Do you find any evidence of multiple equilibria?*

Solution. Solving from five distinct initial conditions, all trajectories converge to the same equilibrium, providing no evidence of multiple equilibria. □

Problem 6. *At the equilibrium (or one of the equilibria - if there are multiple equilibria, you can pick whichever one you like), tell me the values* $\bar{\mu}(3,0)$, $\bar{\gamma}(3,0)$, $\bar{V}(3,0)$, *and* $V(3,0,-2)$.

Solution. At the equilibrium state $(N_t, x_t) = (3,0)$, the computed values are $\bar{\mu}(3,0) = 8.632$, $\bar{\gamma}(3,0) = 7.024$, and $\bar{V}(3,0) = 8.681$. For $V(3,0,-2)$, we have

$$V(3,0,-2) = \max\{-2, \pi(3,0) + \Psi_1(3,0)\} = \max\{-2, 8.632\} = 8.632.$$

□

Problem 7. *Consider a market starting out with 0 firms and* $x_t = 0$. *Simulate this structure of this market 10000 periods into the future. You can do this using only the equilibrium* $\bar{\mu}(N_t, x_t)$ *and* $\bar{\gamma}(N_t, x_t)$ *functions you have computed (along with* $\Pr(x_{t+1}|x_t)$*). Note that you will need to take draws from the* μ_{it} *and* γ_{it} *distributions to do this. What is the average number of firms in the market across these 10000 periods?*

Solution. Simulating 10,000 periods from $(N_0, x_0) = (0,0)$ using the equilibrium cutoff rules yields an average market size of 3.474 firms. □

Problem 8. *Suppose the government decided to implement a 5 unit entry tax. What happens to the average number of firms in equilibrium? (note: you will need to resolve for a new equilibrium to do this). Can you answer this question if there are multiple equilibria? Why or why not?*

Solution. Introducing a 5-unit entry tax reduces average market size from 3.474 to 3.3625 firms (a 3.21% decline). At $(N, x) = (3,0)$, the entry cutoff falls from $\bar{\gamma}(3,0) = 7.024$ to 3.009, lowering the entry probability from 82% to 19%.

If multiple equilibria were present, the policy effect would not be well-defined: the tax could shift the economy across equilibria, making comparative statics ambiguous. □

Problem 9. *Using the simulated data from Question 7), use a BBL-like estimator to estimate* γ, σ_γ^2, μ, *and* σ_μ^2. *In other words assume you know everything about the model (demand, fixed and marginal costs, discount factor,* $p(x_{t+1}|x_t)$ *distribution) except for these 4 parameters, and estimate them. More specifically, estimation should proceed as follows:*

A) Estimate the reduced form expected policy functions, $\hat{d}(N_t, x_t)$ *and* $\hat{e}(N_t, x_t)$, *directly from the data. At each state, these policy functions tell us, respectively, 1) the probability that any specific incumbent stays in the market, and 2) the probability that the potential entrant enters. You can estimate these with a frequency simulator, e.g.* $\hat{d}(N_t, x_t)$ *is the proportion of all incumbents in the data at state* (N_t, x_t) *who decided to stay in the market,* $\hat{e}(N_t, x_t)$ *is the proportion of times in the data where at state* (N_t, x_t), *the entrant decided to enter. Note that you do not need to estimate* $\hat{d}(N_t, x_t)$ *when* $N_t = 0$, *nor do you need to estimate* $\hat{e}(N_t, x_t)$ *when* $N_t = 5$ *(if you want, you can just set* $\hat{d}(0, x_t) = \hat{e}(5, x_t) = 0$*). If there are other states* (N_t, x_t) *that are never reached in the dataset, use the values of* $\hat{d}(N_t, x_t)$ *and* $\hat{e}(N_t, x_t)$ *from a nearby state that was observed in the data. Alternatively, you can estimate* $\hat{d}(N_t, x_t)$ *and* $\hat{e}(N_t, x_t)$ *using a probit or logit model with a high order polynomial in* N_t *and* x_t.

B) Guess the parameters γ, σ_γ^2, μ, and σ_μ^2. At each possible value of the state, use $\hat{d}(N_t, x_t)$, $\hat{e}(N_t, x_t)$, $\pi(N_t, x_t)$ and $\Pr(x_{t+1}|x_t)$ to "forward" simulate the PDV of one of the incumbents (call this firm "Firm 1") conditional on that firm deciding not to exit in the initial period (clearly, you can't do this at states where $N_t = 0$). This involves simulating 1) decisions of potential entrants in the current period and the future, 2) decisions of firms other than Firm 1 in the current period and the future, and 3) decisions of Firm 1 in the future (since we are already conditioning on Firm 1 not exiting in the current period, we don't need to simulate Firm 1's current decision). To simulate all these decisions, you will need to take draws of μ_{it} and γ_{it} (these draws will obviously depend on the parameters. Create them by taking draws from $N(0,1)$'s, multiplying by σ_μ (or σ_γ) and adding μ (or γ). Importantly (and like we did in the BLP problem set, hold the underlying $N(0,1)$ draws constant as the parameters change (also hold the draws on the x_t process the same across simulations). Given such draws, an incumbent stays in the market if $\Phi\left(\frac{\mu_{it}-\mu}{\sigma_\mu}\right) \leq \hat{d}(N_t, x_t)$ and an entrant enters if $\Phi\left(\frac{\gamma_{it}-\gamma}{\sigma_\gamma}\right) \leq \hat{e}(N_t, x_t)$. Note that these equations are implied by the cutoff equilibrium, because if, e.g. $\hat{d}(N_t, x_t) = 0.25$, then firms with μ_{it}'s in the lower quartile of the distribution must be the ones that are staying in the market. Simulate forward in time until either Firm 1 exits the market (or until t gets very large, e.g. 100) and add up Firm 1's total discounted profits. Note that if Firm 1 eventually exits, you will need to add her simulated μ_{it} in that period (appropriately discounted) into total profits to compute the simulated total stream of profits.

C) Do the above forward simulation process 50 times for one of the incumbents in each of the 15 states (i.e. all the states except those with $N_t = 0$), and average across these 50 runs to get an expectation of this PDV. Call these values $\Lambda(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$.

D) Note that according to the model, a firm will stay in the market if $\mu_{it} < \Lambda(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$, which, given the distribution of μ_{it}, should happen with probability

$$\Pr(\textit{incumbent "stays in" at } N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) = \Phi\left(\frac{\Lambda(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) - \mu}{\sigma_\mu}\right)$$

Therefore, if the simulations of $\Lambda(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$ were done at the true parameters γ, σ_γ^2, μ, σ_μ^2, one would expect

$$\Phi\left(\frac{\Lambda(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) - \mu}{\sigma_\mu}\right) \approx \hat{d}(N_t, x_t) \textit{ at every } N_t, x_t \tag{1}$$

E) Do a similar forward simulation process to compute the expected PDV of future profits for a potential entrant in each of the 15 states (ignore states where $N_t = 5$). More specifically, what you want to simulate here is the expected PDV of a potential entrant entering, net of the entry cost γ_{it} (by "net of the entry cost γ_{it}" I mean that you should not add the $-\gamma_{it}$ for the entering period into the total PDV). Denote these simulated values for each of the 15 states by $\Lambda^E(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$. Similar to Step 4), according

to the model, a potential entrant will enter if $\Lambda^E(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) > \gamma_{it}$*, which, given the distribution of* γ_{it}*, should happen with probability*

$$\Pr(\textit{entrant enters at } N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) = \Phi\left(\frac{\Lambda^E(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) - \gamma}{\sigma_\gamma}\right)$$

Therefore, if the simulations of $\Lambda^E(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$ *were done at the true parameters* γ*,* σ_γ^2*,* μ*,* σ_μ^2*, one would expect*

$$\Phi\left(\frac{\Lambda^E(N_t, x_t|\gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) - \gamma}{\sigma_\gamma}\right) \approx \hat{e}(N_t, x_t) \textit{ at every } N_t, x_t \tag{2}$$

F) Given that equations (1) and (2) should hold at the true parameters, we will base estimation on them. More specifically, you should use a search procedure to find the γ*,* σ_γ^2*,* μ*, and* σ_μ^2 *that minimizes the following minimum distance criterion:*

$$\begin{aligned}\min \sum_{N_t=1}^{5}\sum_{x_t}\left[\Phi\left(\frac{\Lambda(N_t, x_t \mid \gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) - \mu}{\sigma_\mu}\right) - \hat{d}(N_t, x_t)\right]^2 \\ + \sum_{N_t=0}^{4}\sum_{x_t}\left[\Phi\left(\frac{\Lambda^E(N_t, x_t \mid \gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2) - \gamma}{\sigma_\gamma}\right) - \hat{e}(N_t, x_t)\right]^2\end{aligned} \tag{1}$$

Solution. The BBL-like estimator follows the two-step CCP/forward-simulation approach of Hotz et al. (1994). Continuation values $\Lambda(N_t, x_t \mid \theta)$ and $\Lambda^E(N_t, x_t \mid \theta)$ are computed from 50 simulated paths over a 1,000-period horizon per initial state. The estimates are $\hat{\mu} = 6.059$, $\hat{\sigma}_\mu = 2.799$, $\hat{\gamma} = 5.040$, and $\hat{\sigma}_\gamma = 3.440$, with an objective value of 0.051855. □

Problem 10. *You do not need to compute standard errors for your estimate (note that standard minimum distance variance formulas are inappropriate here, as they do not account for the estimation error in* $\Lambda(N_t, x_t \mid \gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$ *and* $\Lambda^E(N_t, x_t \mid \gamma, \sigma_\gamma^2, \mu, \sigma_\mu^2)$*, both of which depend on* $\hat{d}(N_t, x_t)$ *and* $\hat{e}(N_t, x_t)$*). Does your estimation procedure recover the true parameters accurately?*

Solution. With 50 simulations, the point estimates exhibit positive bias. Increasing the number of simulations to 1,000 improves the objective value to 0.002690, yielding $\hat{\mu} = 4.928$, $\hat{\sigma}_\mu = 2.359$, $\hat{\gamma} = 5.055$, $\hat{\sigma}_\gamma = 2.185$. Percentile 95% confidence intervals from 100 bootstrap repetitions highlight that the latter procedure recovers the true parameters more accurately.

100 rep percentile CI	50 sims	1,000 sims
$[\hat{\mu}_{\text{boot},2.5\%},\ \hat{\mu}_{\text{boot},97.5\%}]$	[2.653, 8.010]	[4.360, 5.850]
$[\hat{\sigma}_{\mu,\text{boot},2.5\%},\ \hat{\sigma}_{\mu,\text{boot},97.5\%}]$	[2.075, 4.987]	[1.973, 2.706]
$[\hat{\gamma}_{\text{boot},2.5\%},\ \hat{\gamma}_{\text{boot},97.5\%}]$	[1.800, 7.580]	[4.069, 5.947]
$[\hat{\sigma}_{\gamma,\text{boot},2.5\%},\ \hat{\sigma}_{\gamma,\text{boot},97.5\%}]$	[1.723, 4.849]	[1.709, 3.018]

Both sets of results are insensitive to initial guesses of parameter values in the optimization. □

```
import os
import numpy as np
from scipy.stats import norm, binom
from scipy.optimize import minimize
from joblib import Parallel, delayed

class DynamicGame:
    def __init__(self, beta=0.9, mu_mean=5.0, mu_var=5.0, gamma_mean=5.0,
                 gamma_var=5.0, entry_tax=0.0):
        self.beta = beta
        self.mu_mean, self.mu_std = mu_mean, np.sqrt(mu_var)
        self.gamma_mean, self.gamma_std = gamma_mean, np.sqrt(gamma_var)
        self.entry_tax = entry_tax
        self.N_max, self.x_vals = 5, np.array([-5, 0, 5])
        self.x_trans = np.array([[0.6, 0.2, 0.2], [0.2, 0.6, 0.2],
                                 [0.2, 0.2, 0.6]])
        N_vals = np.arange(1, 6)[:, None]
        self.pi = np.vstack([np.zeros(3),
                             ((10 + self.x_vals) / (N_vals + 1))**2 - 5])
        self.mu_cutoff = self.gamma_cutoff = self.V_bar = np.zeros_like(self.pi)

    def solve_equilibrium(self, tol=1e-8, max_iter=5000, damp=0.5):
        self.V_bar = self.mu_cutoff = self.gamma_cutoff = self.pi.copy()
        for _ in range(max_iter):
            V_old, mu_old, ga_old = self.V_bar.copy(), self.mu_cutoff.copy(), \
                self.gamma_cutoff.copy()
            p_stay = np.clip(norm.cdf((self.mu_cutoff - self.mu_mean) /
                                      self.mu_std), 0, 1)
            p_enter = np.clip(norm.cdf((self.gamma_cutoff - self.gamma_mean) /
                                       self.gamma_std), 0, 1)
            p_stay[0], p_enter[5] = 0, 0

            Pr_d1, Pr_e1 = np.zeros((6, 6, 3)), np.zeros((6, 6, 3))
            for N in range(6):
                for j in range(3):
                    if N > 0:
                        pe = p_enter[N,j]
                        for k in range(N):
                            w, Na = binom.pmf(k, N-1, p_stay[N,j]), 1+k
                            Pr_d1[N,Na,j] += w*(1-pe)
                            if Na < 5: Pr_d1[N,Na+1,j] += w*pe
                    if N < 5:
                        for k in range(N+1):
                            Pr_e1[N, min(1+k,5), j] += binom.pmf(k, N,
                                        p_stay[N,j] if N>0 else 0)
```

```
            EV = self.x_trans.T @ self.V_bar.T
            Psi1 = self.beta * np.einsum('ijk,jk->ik', Pr_d1, EV.T)
            Psi2 = self.beta * np.einsum('ijk,jk->ik', Pr_e1, EV.T)
            mu_new, ga_new = self.pi + Psi1, Psi2 - self.entry_tax

            z = (mu_new - self.mu_mean) / self.mu_std
            V_new = np.zeros_like(self.V_bar)
            V_new[1:] = ((1-norm.cdf(z[1:]))*self.mu_mean +
                         self.mu_std*norm.pdf(z[1:]) + norm.cdf(z[1:])*mu_new[1:])

            self.V_bar = damp*V_new + (1-damp)*V_old
            self.mu_cutoff = damp*mu_new + (1-damp)*mu_old
            self.gamma_cutoff = damp*ga_new + (1-damp)*ga_old

            if (max(np.abs(self.V_bar-V_old).max(),
                    np.abs(self.mu_cutoff-mu_old).max(),
                    np.abs(self.gamma_cutoff-ga_old).max()) < tol):
                return True
        return False

    def simulate(self, T=10000, seed=2007):
        rng = np.random.default_rng(seed)
        d_cut = (self.mu_cutoff - self.mu_mean) / self.mu_std
        e_cut = (self.gamma_cutoff - self.gamma_mean) / self.gamma_std
        N, xj = 0, 1
        N_hist = np.zeros(T, dtype=int)
        for t in range(T):
            N_hist[t] = N
            n_stay = (int(np.sum(rng.standard_normal(N) <= d_cut[N,xj]))
                      if N > 0 else 0)
            e = int(rng.standard_normal() <= e_cut[N,xj]) if N < 5 else 0
            N = n_stay + e
            xj = np.searchsorted(self.x_trans[xj].cumsum(), rng.random())
        return N_hist

def estimate_ccps(eq, T=10000, seed=2007):
    rng = np.random.default_rng(seed)
    d_cut = (eq.mu_cutoff - eq.mu_mean) / eq.mu_std
    e_cut = (eq.gamma_cutoff - eq.gamma_mean) / eq.gamma_std
    stay_cnt, tot_cnt, ent_cnt, ent_opp = {}, {}, {}, {}
    N, xj = 0, 1
    for _ in range(T):
        xv = eq.x_vals[xj]
        if N > 0:
```

```
            ns = int(np.sum(rng.standard_normal(N) <= d_cut[N,xj]))
            stay_cnt[(N,xv)] = stay_cnt.get((N,xv), 0) + ns
            tot_cnt[(N,xv)] = tot_cnt.get((N,xv), 0) + N
        else:
            ns = 0
        if N < 5:
            e = int(rng.standard_normal() <= e_cut[N,xj])
            ent_cnt[(N,xv)] = ent_cnt.get((N,xv), 0) + e
            ent_opp[(N,xv)] = ent_opp.get((N,xv), 0) + 1
            N = ns + e
        else:
            N = ns
        xj = np.searchsorted(eq.x_trans[xj].cumsum(), rng.random())
    d_hat = {(N,x): np.clip(stay_cnt.get((N,x),0)/tot_cnt.get((N,x),1),
                            1e-3, 1-1e-3) if (N,x) in tot_cnt else 0.5
             for N in range(1,6) for x in eq.x_vals}
    e_hat = {(N,x): np.clip(ent_cnt.get((N,x),0)/ent_opp.get((N,x),1),
                            1e-3, 1-1e-3) if (N,x) in ent_opp else 0.5
             for N in range(5) for x in eq.x_vals}
    return d_hat, e_hat

def _generate_common_draws(n_states_inc, n_states_ent, n_sim, horizon, seed):
    rng = np.random.default_rng(seed)
    half = max(1, n_sim // 2)

    def antithetic_norm(shape):
        z = rng.standard_normal((shape[0], half) + shape[2:])
        za, out = -z, np.concatenate([z, za], axis=1)
        if out.shape[1] < n_sim:
            out = np.concatenate([out, rng.standard_normal((shape[0], 1) +
                                                           shape[2:])], axis=1)
        return out

    def antithetic_unif(shape):
        u = rng.random((shape[0], half) + shape[2:])
        ua, out = 1.0 - u, np.concatenate([u, ua], axis=1)
        if out.shape[1] < n_sim:
            out = np.concatenate([out, rng.random((shape[0], 1) +
                                                  shape[2:])], axis=1)
        return out

    mu_other = antithetic_norm((n_states_inc, n_sim, horizon, 5))
    mu_firm1 = antithetic_norm((n_states_inc, n_sim, horizon))
    gamma_entry = antithetic_norm((n_states_inc, n_sim, horizon))
    demand_u_inc = antithetic_unif((n_states_inc, n_sim, horizon))
```

```
    mu_inc_initial = antithetic_norm((n_states_ent, n_sim, 5))
    mu_firmE = antithetic_norm((n_states_ent, n_sim, horizon))
    mu_otherE = antithetic_norm((n_states_ent, n_sim, horizon, 5))
    gamma_entryE = antithetic_norm((n_states_ent, n_sim, horizon))
    demand_u_ent = antithetic_unif((n_states_ent, n_sim, horizon))

    return (mu_other, mu_firm1, gamma_entry, demand_u_inc, mu_inc_initial,
            mu_firmE, mu_otherE, gamma_entryE, demand_u_ent)

def bbl_estimate(d_hat, e_hat, eq, n_sim=50, horizon=100, seed=2007,
                 bootstrap=False, n_bootstrap=100, start_from=None):
    states_inc = [(N, j) for N in range(1, 6) for j in range(3)]
    states_ent = [(N, j) for N in range(0, 5) for j in range(3)]
    d_arr = np.full((6, 3), 0.5)
    e_arr = np.full((6, 3), 0.5)
    for N in range(1, 6):
        for j, xv in enumerate(eq.x_vals):
            if (N, xv) in d_hat: d_arr[N, j] = d_hat[(N, xv)]
    for N in range(0, 5):
        for j, xv in enumerate(eq.x_vals):
            if (N, xv) in e_hat: e_arr[N, j] = e_hat[(N, xv)]
    d_arr, e_arr = np.clip(d_arr, 1e-6, 1 - 1e-6), np.clip(e_arr, 1e-6, 1 - 1e-6)
    d_thr, e_thr = norm.ppf(d_arr), norm.ppf(e_arr)
    x_cum_full = eq.x_trans.cumsum(axis=1)
    VERBOSE = int(os.environ.get("BBL_VERBOSE", "0"))
    N_STARTS = max(1, int(os.environ.get("BBL_N_STARTS", "3")))

def make_simulator(hor, ns, sd):
        beta_pows = eq.beta ** np.arange(hor + 1)
        draws = _generate_common_draws(len(states_inc), len(states_ent),
                                       ns, hor, sd)
        (mu_other, mu_firm1, gamma_entry, demand_u_inc, mu_inc_initial,
         mu_firmE, mu_otherE, gamma_entryE, demand_u_ent) = draws
        ns_eff = mu_other.shape[1]

        def next_x_idx(x_idx_vec, u_vec):
            c = x_cum_full[x_idx_vec]
            return np.sum(u_vec[:, None] > c, axis=1)

        def simulate_Lambda(theta):
            mu_m, sig_mu, ga_m, sig_g = theta
            Lam, LamE = {}, {}

            def sim_inc_state(si, N0, xi0):
                N = np.full(ns_eff, N0, dtype=int)
```

```
x = np.full(ns_eff, xi0, dtype=int)
alive, pdv = np.ones(ns_eff, dtype=bool), np.zeros(ns_eff)
pdv += eq.pi[N0, xi0]
stay_other = np.zeros(ns_eff, dtype=int)
ent0 = np.zeros(ns_eff, dtype=int)
if N0 > 1:
    z_vec0 = mu_other[si, :, 0, :N0 - 1]
    stay_other = np.sum(z_vec0 <= d_thr[N0, xi0],
                        axis=1)
if N0 < 5:
    ent0 = (gamma_entry[si, :, 0] <= e_thr[N0, xi0])\
        .astype(int)
N = np.minimum(1 + stay_other + ent0, 5)
x = next_x_idx(x, demand_u_inc[si, :, 0])
for t in range(1, hor):
    if not alive.any(): break
    idx, zf = np.where(alive)[0], \
        mu_firm1[si, np.where(alive)[0], t]
    thr = d_thr[N[idx], x[idx]]
    stay_f1, exiting = zf <= thr, idx[~stay_f1]
    if exiting.size > 0:
        pdv[exiting] += beta_pows[t] * \
            (mu_m + sig_mu * mu_firm1[si, exiting, t])
        alive[exiting] = False
    surv = idx[stay_f1]
    if surv.size > 0:
        pdv[surv] += beta_pows[t] * eq.pi[N[surv], x[surv]]
        stay_o = np.zeros(surv.size, dtype=int)
        ent = np.zeros(surv.size, dtype=int)
        has_others = N[surv] > 1
        if np.any(has_others):
            sv_idx = np.where(has_others)[0]
            z_o = mu_other[si, surv[sv_idx], t, :]
            thr_o = d_thr[N[surv[sv_idx]], x[surv[sv_idx]]]\
                [:, None]
            mask = np.arange(5) < (N[surv[sv_idx]] - 1)\
                [:, None]
            stay_o_vals = (z_o <= thr_o) & mask
            stay_o[sv_idx] = stay_o_vals.sum(axis=1)
        Nn = stay_o + 1
        can_ent = N[surv] < 5
        if np.any(can_ent):
            ce_idx = np.where(can_ent)[0]
            ent[ce_idx] = (gamma_entry[si, surv[ce_idx], t] <= \
                           e_thr[N[surv[ce_idx]], x[surv[ce_idx]]])\
```

```
                        .astype(int)
                N[surv] = np.minimum(Nn + ent, 5)
                x[surv] = next_x_idx(x[surv], \
                                     demand_u_inc[si, surv, t])
        return (N0, eq.x_vals[xi0]), float(pdv.mean())

    def sim_ent_state(si, N0, xi0):
        stay_inc = np.zeros(ns_eff, dtype=int)
        if N0 > 0:
            stay_inc = np.sum(mu_inc_initial[si, :, :N0] <=
                              d_thr[N0, xi0], axis=1)
        N = np.minimum(stay_inc + 1, 5)
        x = next_x_idx(np.full(ns_eff, xi0, dtype=int),
                       demand_u_ent[si, :, 0])
        alive, pdv = np.ones(ns_eff, dtype=bool), np.zeros(ns_eff)
        for t in range(1, hor):
            if not alive.any(): break
            idx, zE = np.where(alive)[0], \
                mu_firmE[si, np.where(alive)[0], t]
            thr = d_thr[N[idx], x[idx]]
            stay_E, exiting = zE <= thr, idx[~stay_E]
            if exiting.size > 0:
                pdv[exiting] += beta_pows[t] * \
                    (mu_m + sig_mu * mu_firmE[si, exiting, t])
                alive[exiting] = False
            surv = idx[stay_E]
            if surv.size > 0:
                pdv[surv] += beta_pows[t] * eq.pi[N[surv], x[surv]]
                stay_o = np.zeros(surv.size, dtype=int)
                ent = np.zeros(surv.size, dtype=int)
                has_others = N[surv] > 1
                if np.any(has_others):
                    sv_idx = np.where(has_others)[0]
                    zOE = mu_otherE[si, surv[sv_idx], t, :]
                    thr_o = d_thr[N[surv[sv_idx]], x[surv[sv_idx]]]\
                        [:, None]
                    mask = np.arange(5) < (N[surv[sv_idx]] - 1)\
                        [:, None]
                    stay_o_vals = (zOE <= thr_o) & mask
                    stay_o[sv_idx] = stay_o_vals.sum(axis=1)
                Nn = stay_o + 1
                can_ent = N[surv] < 5
                if np.any(can_ent):
                    ce_idx = np.where(can_ent)[0]
                    ent[ce_idx] = (gamma_entryE[si, surv[ce_idx], t] <= \
```

```
                                        e_thr[N[surv[ce_idx]], x[surv[ce_idx]]])\
                            .astype(int)
                    N[surv] = np.minimum(Nn + ent, 5)
                    x[surv] = next_x_idx(x[surv], \
                                         demand_u_ent[si, surv, t])
            return (N0, eq.x_vals[xi0]), float(pdv.mean())
        n_jobs = int(os.environ.get("BBL_NJOBS", "1"))
        if Parallel is not None and n_jobs != 1:
            inc_results = Parallel(n_jobs=n_jobs, prefer="threads")(
                delayed(sim_inc_state)(si, N0, xi0)
                for si, (N0, xi0) in enumerate(states_inc))
            ent_results = Parallel(n_jobs=n_jobs, prefer="threads")(
                delayed(sim_ent_state)(si, N0, xi0)
                for si, (N0, xi0) in enumerate(states_ent))
        else:
            inc_results = [sim_inc_state(si, N0, xi0)
                           for si, (N0, xi0) in enumerate(states_inc)]
            ent_results = [sim_ent_state(si, N0, xi0)
                           for si, (N0, xi0) in enumerate(states_ent)]
        for k, v in inc_results: Lam[k] = v
        for k, v in ent_results: LamE[k] = v
        return Lam, LamE

    def objective(theta):
        Lam, LamE = simulate_Lambda(theta)
        inc_err, ent_err = 0.0, 0.0
        for N, j in states_inc:
            inc_err += (norm.cdf((Lam[(N, eq.x_vals[j])] - theta[0]) /
                                 theta[1]) - d_arr[N, j]) ** 2
        for N, j in states_ent:
            ent_err += (norm.cdf((LamE[(N, eq.x_vals[j])] - theta[2]) /
                                 theta[3]) - e_arr[N, j]) ** 2
        return inc_err + ent_err

    return simulate_Lambda, objective

sim_full, obj_full = make_simulator(horizon, n_sim, seed)

if start_from is not None:
    starts = [np.array(start_from, dtype=float)]
else:
    starts, rng = [], np.random.default_rng(
        int(os.environ.get("BBL_SEED", "12345")))
    for _ in range(N_STARTS):
        starts.append(np.array([rng.uniform(0, 8), rng.uniform(1.2, 3.5),
```

```
                                    rng.uniform(0, 8), rng.uniform(1.2, 3.5)]))

    best_res, all_results = None, []
    for i, x0 in enumerate(starts):
        res = minimize(obj_full, x0=x0, method='BFGS')
        all_results.append((i, x0, res))
        if best_res is None or res.fun < best_res.fun: best_res = res
    best_res.all_starts, best_res.se = all_results, None
    try:
        def cdf_inc(val, N, j): return norm.cdf((val - best_res.x[0]) / best_res.x[1])
        def cdf_ent(val, N, j): return norm.cdf((val - best_res.x[2]) / best_res.x[3])
        simulate_Lambda, _ = make_simulator(horizon, n_sim, seed)
        Lam, LamE = simulate_Lambda(best_res.x)
        inc_sq, ent_sq, inc_abs, ent_abs = [], [], [], []
        for N, j in states_inc:
            pred, err = cdf_inc(Lam[(N, eq.x_vals[j])], N, j) - d_arr[N, j]
            inc_sq.append(err*err), inc_abs.append(abs(err))
        for N, j in states_ent:
            pred, err = cdf_ent(LamE[(N, eq.x_vals[j])], N, j) - e_arr[N, j]
            ent_sq.append(err*err), ent_abs.append(abs(err))
        inc_sum, ent_sum = float(np.sum(inc_sq)), float(np.sum(ent_sq))
        best_res.obj_breakdown = {
            "inc_sum": inc_sum, "ent_sum": ent_sum, "total": inc_sum + ent_sum,
            "inc_mse": inc_sum / len(states_inc), "ent_mse": ent_sum / len(states_ent),
            "inc_max_abs": float(np.max(inc_abs)) if inc_abs else 0.0,
            "ent_max_abs": float(np.max(ent_abs)) if ent_abs else 0.0,
            "n_sim": n_sim, "horizon": horizon,}
    except Exception:
        best_res.obj_breakdown = None
    return best_res

def _bootstrap_rep(b, eq, theta_hat, n_sim, horizon, T_ccp, seed):
    try:
        boot_seed = seed + 10000 * (b + 1)
        d_hat_boot, e_hat_boot = estimate_ccps(eq, T=T_ccp,
                                               seed=boot_seed)
        old_verbose, old_nstarts, old_njobs = (
            os.environ.get("BBL_VERBOSE", "0"),
            os.environ.get("BBL_N_STARTS", "1"),
            os.environ.get("BBL_NJOBS", "1")
        )
        os.environ["BBL_VERBOSE"], os.environ["BBL_N_STARTS"], \
            os.environ["BBL_NJOBS"] = "0", "1", "1"
        res = bbl_estimate(d_hat_boot, e_hat_boot, eq,
                           n_sim=n_sim, horizon=horizon,
```

```
                            seed=boot_seed, bootstrap=False,
                            n_bootstrap=0, start_from=theta_hat)
        os.environ["BBL_VERBOSE"], os.environ["BBL_N_STARTS"], \
            os.environ["BBL_NJOBS"] = old_verbose, old_nstarts, \
            old_njobs
        return res.x if res is not None and hasattr(res, 'x') \
            else np.array([np.nan]*4)
    except Exception:
        return np.array([np.nan]*4)

def bootstrap_estimates_parallel(eq, theta_hat, n_sim=300,
                                 horizon=300, T_ccp=10000,
                                 seed=2007, n_boot=100, n_jobs=-1):
    if Parallel is None:
        estimates = [_bootstrap_rep(b, eq, theta_hat, n_sim,
                                    horizon, T_ccp, seed)
                     for b in range(n_boot)]
    else:
        print(f"Running {n_boot} BBL bootstraps in parallel "
              f"(n_jobs={n_jobs})...")
        batch_estimates = Parallel(n_jobs=n_jobs,
                                   backend="threading", verbose=1)(
            delayed(_bootstrap_rep)(b, eq, theta_hat, n_sim,
                                    horizon, T_ccp, seed)
            for b in range(n_boot))
        estimates = batch_estimates
        print(f"  Completed {n_boot} bootstrap replications")

    estimates = np.array(estimates)
    valid = ~np.isnan(estimates).any(axis=1)
    return estimates[valid] if valid.sum() >= 5 else None

def main():
    print("Dynamic Entry/Exit Game: Equilibrium and Estimation")
    inits = [("profits", None), ("0", np.zeros((6,3))),
             ("1", np.ones((6,3))),
             ("U[0,10]", np.random.RandomState(123)\
              .uniform(0,10,(6,3))),
             ("U[-5,15]", np.random.RandomState(456)\
              .uniform(-5,15,(6,3)))]
    sols = []
    for name, init_val in inits:
        game = DynamicGame()
        if init_val is not None:
            game.V_bar = game.mu_cutoff = \
```

```
            game.gamma_cutoff = init_val.copy()
    game.solve_equilibrium()
    sols.append(game)
    print(f"Init {name}: V_bar(3,0)={game.V_bar[3,1]:.4f}, "
          f"mu_bar(3,0)={game.mu_cutoff[3,1]:.4f}")

eq = sols[0]
print(f"Equilibrium at (N,x)=(3,0): mu_bar={eq.mu_cutoff[3,1]:.6f}, "
      f"gamma_bar={eq.gamma_cutoff[3,1]:.6f}, V_bar={eq.V_bar[3,1]:.6f}")

N_hist = eq.simulate()
print(f"Average firms: {N_hist.mean():.4f}")

tax = DynamicGame(entry_tax=5.0)
tax.solve_equilibrium()
N_tax = tax.simulate()
print(f"With 5-unit tax: {N_tax.mean():.4f} firms "
      f"(Change: {N_tax.mean() - N_hist.mean():.4f})")

T_ccp = int(os.environ.get("BBL_T_CCP", "10000"))
d_hat, e_hat = estimate_ccps(eq, T=T_ccp, seed=2007)
print(f"Estimated {len(d_hat)} CCPs and {len(e_hat)} ECPs "
      f"from {T_ccp} data periods")

n_sim_ps, horizon_ps = 50, 1000
res_ps = bbl_estimate(d_hat, e_hat, eq, n_sim=n_sim_ps,
                      horizon=horizon_ps, seed=2007)
if hasattr(res_ps, 'all_starts'):
    for i, x0, r in res_ps.all_starts:
        print(f"Start {i}: obj={r.fun:.6f}, mu={r.x[0]:.3f}, "
              f"sigma_mu={r.x[1]:.3f}, gamma={r.x[2]:.3f}, "
              f"sigma_gamma={r.x[3]:.3f}")

n_sim_hp, horizon_hp = (int(os.environ.get("BBL_N_SIM", "1000")),
                        int(os.environ.get("BBL_H", "1000")))
res_hp = bbl_estimate(d_hat, e_hat, eq, n_sim=n_sim_hp,
                      horizon=horizon_hp, seed=2007)
if hasattr(res_hp, 'all_starts'):
    for i, x0, r in res_hp.all_starts:
        print(f"Start {i}: obj={r.fun:.6f}, mu={r.x[0]:.3f}, "
              f"sigma_mu={r.x[1]:.3f}, gamma={r.x[2]:.3f}, "
              f"sigma_gamma={r.x[3]:.3f}")

res = res_hp # res = res_ps
do_bootstrap = int(os.environ.get("BBL_BOOTSTRAP", "0")) > 0
```

```
    if do_bootstrap:
        n_boot, n_jobs, T_ccp_boot = \
            (int(os.environ.get("BBL_N_BOOTSTRAP", "100")),
            int(os.environ.get("BBL_BOOT_NJOBS", "-1")),
            int(os.environ.get("BBL_BOOT_T_CCP", "10000")))
        boot = bootstrap_estimates_parallel(eq, res.x, res.n_sim,
                                            horizon=1000,
                                            T_ccp=T_ccp_boot,
                                            seed=2007,
                                            n_boot=n_boot,
                                            n_jobs=n_jobs)
        if boot is not None and boot.shape[0] >= 5:
            se, ci_lo, ci_hi = (boot.std(axis=0, ddof=1),
                                np.percentile(boot, 2.5, axis=0),
                                np.percentile(boot, 97.5, axis=0))
            print(f"Bootstrap SE: mu={se[0]:.3f}, sigma_mu={se[1]:.3f}, "
                  f"gamma={se[2]:.3f}, sigma_gamma={se[3]:.3f}")
            print(f"95% CI: mu=[{ci_lo[0]:.3f}, {ci_hi[0]:.3f}], "
                  f"sigma_mu=[{ci_lo[1]:.3f}, {ci_hi[1]:.3f}], "
                  f"gamma=[{ci_lo[2]:.3f}, {ci_hi[2]:.3f}], "
                  f"sigma_gamma=[{ci_lo[3]:.3f}, {ci_hi[3]:.3f}]")

if __name__ == "__main__":
    main()
[Running] python -u ".../dg.py"
Q4-5: Equilibrium Computation
  Init profits  : V_bar(3,0)=8.6810, mu_bar(3,0)=8.6320
  Init 0        : V_bar(3,0)=8.6810, mu_bar(3,0)=8.6320
  Init 1        : V_bar(3,0)=8.6810, mu_bar(3,0)=8.6320
  Init U[0,10]  : V_bar(3,0)=8.6810, mu_bar(3,0)=8.6320
  Init U[-5,15] : V_bar(3,0)=8.6810, mu_bar(3,0)=8.6320
  Result: UNIQUE

Q6: Equilibrium at (N,x)=(3,0)
  mu_bar(3,0) = 8.631981
  gamma_bar(3,0) = 7.024259
  V_bar(3,0) = 8.681050
  V(3,0,-2) = 8.631981 (firm stays)

Q7: Market Simulation (10,000 periods)
  Average firms: 3.4740

Q8: Entry Tax Counterfactual
  With 5-unit tax: 3.3625 firms
  Change: -0.1115 firms (-3.21%)
```

```
Q9: BBL-like Estimation (forward simulation of Lambda, Lambda^E)
  Step 1: Nonparametric CCP estimation from simulated data
    Simulating 10000 periods from equilibrium...
    Estimated 15 incumbent CCPs and 15 entry CCPs from data
    Sample CCP comparison (estimated vs true):
      d(N=3,x=0): est=0.9497, true=0.9478
      d(N=3,x=-5): est=0.5147, true=0.5157
      d(N=3,x=5): est=0.9990, true=1.0000
      e(N=3,x=0): est=0.8206, true=0.8173
      e(N=3,x=-5): est=0.9990, true=0.9980
      e(N=3,x=5): est=0.9290, true=0.9261
  Step 2: Forward simulation (Lambda, Lambda^E)
          and minimum distance search
  Using 50 simulations, horizon=1000
    Start rng-1   : obj=0.051855, mu= 6.059, sigma_mu=2.799,
                    gamma= 5.040, sigma_gamma=3.440, iters=26
    Start rng-2   : obj=0.051855, mu= 6.059, sigma_mu=2.799,
                    gamma= 5.040, sigma_gamma=3.440, iters=23
    Start rng-3   : obj=0.051855, mu= 6.059, sigma_mu=2.799,
                    gamma= 5.040, sigma_gamma=3.440, iters=30
    Objective breakdown:    total=0.051855, inc_sum=0.041749, ent_sum=0.010106
  Running 100 BBL bootstraps in parallel (n_jobs=-1)...
    Bootstrap SE:           mu= 1.341, sigma_mu=0.703,
                            gamma= 1.522, sigma_gamma=0.824
    Percentile 95% CI:      mu=[ 2.653,  8.010],
                            sigma_mu=[2.075, 4.987],
                            gamma=[ 1.800,  7.580],
                            sigma_gamma=[1.723, 4.849]
  Using 1000 simulations, horizon=1000
    Start rng-1   : obj=0.002690, mu= 4.928, sigma_mu=2.359,
                    gamma= 5.055, sigma_gamma=2.185, iters=28
    Start rng-2   : obj=0.002690, mu= 4.928, sigma_mu=2.359,
                    gamma= 5.055, sigma_gamma=2.185, iters=26
    Start rng-3   : obj=0.002690, mu= 4.928, sigma_mu=2.359,
                    gamma= 5.054, sigma_gamma=2.186, iters=26
    Objective breakdown:    total=0.002690, inc_sum=0.002342, ent_sum=0.000348
  Running 100 BBL bootstraps in parallel (n_jobs=-1)...
    Bootstrap SE:           mu= 0.390, sigma_mu=0.193,
                            gamma= 0.481, sigma_gamma=0.320
    Percentile 95% CI:      mu=[ 4.360, 5.850],
                            sigma_mu=[1.973, 2.706],
                            gamma=[ 4.069, 5.947],
                            sigma_gamma=[1.709, 3.018]
```

References

Hotz, V. J., R. A. Miller, S. Sanders, and J. Smith (1994): "A Simulation Estimator for Dynamic Models of Discrete Choice," *The Review of Economic Studies*, 61, 265–289.

Rust, J. (1987): "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher," *Econometrica*, 55, 999–1033.