GameSimulation3

December 1, 2019

Estimating Firm-Entry under Discrete Game Framework with Python

1. Introduction

Marketing concerns understanding, predicting, and influencing various agents' choices, for example for firms, they need to figure out what to sell, how much to charge, when to introduce new products, etc.

Discrete game is a generalization of a standard discrete model where utility depends not only on the player himself but also on the actions of other players. The decisions involve strategic interactions and they are NOT made in a vacuum.

To find the payoff function or utility function from the firms entry decisions, the structure of the game depends on what's being modelled, for example: - Decision set may be discrete or continuous, thus the game model could be either discrete or continuous as well. - Every information may be observed by the players, or these players make decisions with uncertainty, thus the game model could be either under complete or incomplete information framework. - This game may be a one-shot game, or it may continue for many periods, therefore we need to decide whether to build a model of repeted game and include the discount rate.

As for this simulation, I focus on a static, discrete game with complete information.

2. Simulation

2.1 The Model

To illustrate the static, discrete firm-entry game with complete information, I first make several assumptions as follows:

- Suppose rm 1, 2 and 3 compete in many local markets. Like the models of Ellicken and Misra (2011), Zhaokun Zhang and Zhesheng Zheng (2019).
- Each rm chooses either "enter" or "not enter".
- Each rm can only open at most one store in one market.

With python, I made a simulate and generate m = 5000 small and isolated local markets. In this process, I futher assume that:

- X_m is a measure of market characteristics and it's the same for all 3 rms.
- Z_{im} represents rm i's characteristics that are unique to this firm. Moreover, one firm's character doesn't influence other firms' profit.

- y_{im} is the action of its rivals, $y_{im} = 1$ if one of the rival firms decide to enter; $y_{im} = 2$ if both rival firms decide to enter; $y_{im} = 0$ if none of the rival firms decide to enter.
- δ captures the degree of compatitiviness within one market.
- ϵ_i is a component of profits the firm i sees but we don't. (In this complete information model, we assume not only firm i knows ϵ_i, the other two firms also observe ϵ_i, i.e., the researcher cannot get the information of $\epsilon_{1,2,3}$, yet all 3 firms WILL consider ϵ when they make choices about whether to enter.)

Therefore, the profit function of firm i in market m is:

$$\pi_{im} = \alpha_i' X_m + \beta_i' Z_{im} + \delta y_{-im} + \varepsilon_{im} \tag{1}$$

In this model, I assume the true value of parameters $(alpha1,\ alpha2,\ alpha3,\ beta1,\ beta2,\ beta3,\ delta,\ Scale)$ are $(2, 1, 0, 2, 2, 0.7, -2, 1)$ correspondingly. Thus, expected prot is a function of the market characteristics, the rm's characteristics, and its rivals' decisions.

2.2 Equilibrium

In equilibrium, firms maximize profits, taking rivals' actions as given. Therefore, a Nash equilibrium is characterized by:

$$\begin{aligned} y_{1m} &= 1\left[\alpha_1' X_m + \beta_1' Z_{1m} + \delta y_{-1m} + \varepsilon_{1m} \geq 0\right] \\ y_{2m} &= 1\left[\alpha_2' X_m + \beta_2' Z_{2m} + \delta y_{-2m} + \varepsilon_{2m} \geq 0\right] \\ y_{3m} &= 1\left[\alpha_3' X_m + \beta_3' Z_{3m} + \delta y_{-3m} + \varepsilon_{3m} \geq 0\right] \end{aligned} \tag{2}$$

The likelihood of observing n_m firms in a given market m can be computed in closed form:

$$\begin{aligned} &\Pr(n_m = 3) = \textstyle\prod_i \Pr(\alpha_i' X_m + \beta_i' Z_{im} - \delta y_{-im} + \varepsilon_{im} \geq 0) \\ &\Pr(n_m = 0) = \textstyle\prod_i \Pr(\alpha_i' X_m + \beta_i' Z_{im} - \delta y_{-im} + \varepsilon_{im} < 0) \\ &\Pr(n_m = 2) = \textstyle\sum_i \left[\left(\prod_{j\neq i} \Pr\left(\alpha_j' X_m + \beta_j' Z_{jm} - \delta y_{-jm} + \varepsilon_{jm} \geq 0\right)\right) \cdot \Pr(\alpha_i' X_m + \beta_i' Z_{im} - \delta_{-im} + \varepsilon_{im} < 0)\right] \\ &\Pr(n_m = 1) = 1 - \Pr(n_m = 3) - \Pr(n_m = 0) - \Pr(n_m = 1) \end{aligned} \tag{3}$$

The sample log-likelihood is:

$$\ln \mathcal{L} = \sum_{m=1}^{M} \sum_{l=0}^{3} \mathcal{I}(n_m = l) \ln \Pr(n_m = l) \tag{4}$$

2.3 Simulation

```
In [2]: import numpy as np
        from scipy.stats import norm
        import scipy as np
        from scipy.optimize import minimize
```

```
In [3]: m = 5000
```

In [68]:
```
np.random.seed(678456)
X_m = np.random.normal(loc=1.0, scale=1, size=(1,m))
X_m = X_m.astype(np.float32)
np.random.seed(123456)
Z_1m = np.random.normal(loc=3.0, scale=1, size=(1,m))
Z_1m = Z_1m.astype(np.float32)
np.random.seed(234567)
Z_2m = np.random.normal(loc=3.0, scale=1, size=(1,m))
Z_2m = Z_2m.astype(np.float32)
np.random.seed(345678)
Z_3m = np.random.normal(loc=3.0, scale=1, size=(1,m))
Z_3m = Z_3m.astype(np.float32)

Scale = 1.0
np.random.seed(67891011)
e_1 = np.random.normal(loc=0.0, scale=Scale, size=(1,m))
e_1 = e_1.astype(np.float32)
np.random.seed(34567891)
e_2 = np.random.normal(loc=0.0, scale=Scale, size=(1,m))
e_2 = e_2.astype(np.float32)
np.random.seed(12345678)
e_3 = np.random.normal(loc=0.0, scale=Scale, size=(1,m))
e_3 = e_3.astype(np.float32)
```

In [74]:
```
alpha1, alpha2, alpha3, beta1, beta2, beta3, delta, Scale = 2,1,0, 2,2,1, -2, 1
```

In [75]:
```
#Outcome Matrix
Outcome = np.zeros([1,3*m])

Profit_1m_2enter_3enter = alpha1*X_m + beta1*Z_1m + (delta*1 + delta*1) + e_1
Profit_1m_2enter_3enter = np.maximum(Profit_1m_2enter_3enter, 0)
Profit_1m_2enter_3notenter = alpha1*X_m + beta1*Z_1m + delta*1 + e_1
Profit_1m_2enter_3notenter = np.maximum(Profit_1m_2enter_3notenter, 0)
Profit_1m_2notenter_3enter = alpha1*X_m + beta1*Z_1m + delta*1 + e_1
Profit_1m_2notenter_3enter = np.maximum(Profit_1m_2notenter_3enter, 0)
Profit_1m_2notenter_3notenter = alpha1*X_m + beta1*Z_1m + e_1
Profit_1m_2notenter_3notenter = np.maximum(Profit_1m_2notenter_3notenter, 0)

Profit_2m_1enter_3enter = alpha2*X_m + beta2*Z_2m + (delta*1 + delta*1) + e_2
Profit_2m_1enter_3enter = np.maximum(Profit_2m_1enter_3enter, 0)
Profit_2m_1enter_3notenter = alpha2*X_m + beta2*Z_2m + delta*1 + e_2
Profit_2m_1enter_3notenter = np.maximum(Profit_2m_1enter_3notenter, 0)
Profit_2m_1notenter_3enter = alpha2*X_m + beta2*Z_2m + delta*1 + e_2
Profit_2m_1notenter_3enter = np.maximum(Profit_2m_1notenter_3enter, 0)
Profit_2m_1notenter_3notenter = alpha2*X_m + beta2*Z_2m + e_2
Profit_2m_1notenter_3notenter = np.maximum(Profit_2m_1notenter_3notenter, 0)

Profit_3m_1enter_2enter = alpha3*X_m + beta3*Z_3m + (delta*1 + delta*1) + e_3
```

```
Profit_3m_1enter_2enter = np.maximum(Profit_3m_1enter_2enter, 0)
Profit_3m_1enter_2notenter = alpha3*X_m + beta3*Z_3m + delta*1 + e_3
Profit_3m_1enter_2notenter = np.maximum(Profit_3m_1enter_2notenter, 0)
Profit_3m_1notenter_2enter = alpha3*X_m + beta3*Z_3m + delta*1 + e_3
Profit_3m_1notenter_2enter = np.maximum(Profit_3m_1notenter_2enter, 0)
Profit_3m_1notenter_2notenter = alpha3*X_m + beta3*Z_3m + e_3
Profit_3m_1notenter_2notenter = np.maximum(Profit_3m_1notenter_2notenter, 0)

for i in range(m):
    if (Profit_1m_2enter_3enter[0][i] == 0 and Profit_2m_1enter_3enter[0][i] == 0 and
        Outcome[0,3*i] = 0
        Outcome[0,3*i+1] = 0
        Outcome[0,3*i+2] = 0
    elif (Profit_1m_2enter_3enter[0][i] > 0 and Profit_2m_1enter_3enter[0][i] > 0 and
        Outcome[0,3*i] = 1
        Outcome[0,3*i+1] = 1
        Outcome[0,3*i+2] = 0
    elif (Profit_1m_2enter_3enter[0][i] > 0 and Profit_2m_1enter_3enter[0][i] == 0 and
        Outcome[0,3*i] = 1
        Outcome[0,3*i+1] = 0
        Outcome[0,3*i+2] = 1
    elif (Profit_1m_2enter_3enter[0][i] > 0 and Profit_2m_1enter_3enter[0][i] > 0 and
        Outcome[0,3*i] = 1
        Outcome[0,3*i+1] = 1
        Outcome[0,3*i+2] = 1
    elif (Profit_1m_2enter_3enter[0][i] == 0 and Profit_2m_1enter_3enter[0][i] > 0 and
        Outcome[0,3*i] = 0
        Outcome[0,3*i+1] = 1
        Outcome[0,3*i+2] = 0
    elif (Profit_1m_2enter_3enter[0][i] == 0 and Profit_2m_1enter_3enter[0][i] > 0 and
        Outcome[0,3*i] = 0
        Outcome[0,3*i+1] = 1
        Outcome[0,3*i+2] = 1
    elif (Profit_1m_2enter_3enter[0][i] > 0 and Profit_2m_1enter_3enter[0][i] == 0 and
        Outcome[0,3*i] = 1
        Outcome[0,3*i+1] = 0
        Outcome[0,3*i+2] = 0
    elif (Profit_1m_2enter_3enter[0][i] == 0 and Profit_2m_1enter_3enter[0][i] == 0 ar
        Outcome[0,3*i] = 0
        Outcome[0,3*i+1] = 0
        Outcome[0,3*i+2] = 1
    else:
        Outcome[0,3*i] = np.nan
        Outcome[0,3*i+1] = np.nan
        Outcome[0,3*i+2] = np.nan
```

In [76]:
```
Outcome_1 = Outcome.reshape([m,3])
np.savetxt("/Users/Documents/MicroEconometrics/Hw3/Outcome.txt", Outcome_1,f
```

```
In [77]: outcome = np.sum(Outcome_1, axis = 1)
         np.savetxt("/Users/Documents/MicroEconometrics/Hw3/OOutcome.txt", outcome, f
```

2.4 Estimating Payoff Function

```
In [13]: def Probability(alpha1, alpha2, alpha3, beta1, beta2, beta3, delta, Scale):
             probability = np.zeros([1,m])
             #probability = probability.astype(np.float32)
             #dist = tfp.distributions.Normal(loc=0., scale=3.)
             Profit_1m_2enters = alpha1*X_m + beta1*Z_1m + 2*delta + e_1
             Profit_1m_1enter = alpha1*X_m + beta1*Z_1m + delta + e_1
             Profit_1m_notenter = alpha1*X_m + beta1*Z_1m + e_1

             Profit_2m_2enters = alpha2*X_m + beta2*Z_2m + 2*delta + e_2
             Profit_2m_1enter = alpha2*X_m + beta2*Z_2m + delta + e_2
             Profit_2m_notenter = alpha2*X_m + beta2*Z_2m + e_2

             Profit_3m_2enters = alpha3*X_m + beta3*Z_3m + 2*delta + e_3
             Profit_3m_1enter = alpha3*X_m + beta3*Z_3m + delta + e_3
             Profit_3m_notenter = alpha3*X_m + beta3*Z_3m + e_3


             for i in range(m):
                 if (int(Outcome_1[i,0]) == 1 and int(Outcome_1[i,1]) == 1 and int(Outcome_1[i
                     probability[0,i] = norm.cdf(Profit_1m_2enters[0][i]/Scale) * norm.cdf(Pro

                 elif (int(Outcome_1[i,0]) == 1 and int(Outcome_1[i,1]) == 1 and int(Outcome_1
                     probability[0,i] = norm.cdf(Profit_1m_1enter[0][i]/Scale) * norm.cdf(Prof
                 elif (int(Outcome_1[i,0]) == 1 and int(Outcome_1[i,1]) == 0 and int(Outcome_1
                     probability[0,i] = norm.cdf(Profit_1m_1enter[0][i]/Scale) * (1 - norm.cdf

                 elif (int(Outcome_1[i,0]) == 1 and int(Outcome_1[i,1]) == 0 and int(Outcome_1
                     probability[0,i] = norm.cdf(Profit_1m_notenter[0][i]/Scale) * (1 - norm.c
                 elif (int(Outcome_1[i,0]) == 0 and int(Outcome_1[i,1]) == 1 and int(Outcome_1
                     probability[0,i] = (1 - norm.cdf(Profit_1m_2enters[0][i]/Scale)) * norm.c

                 elif (int(Outcome_1[i,0]) == 0 and int(Outcome_1[i,1]) == 1 and int(Outcome_1
                     probability[0,i] = (1 - norm.cdf(Profit_1m_1enter[0][i]/Scale)) * norm.cd
                 elif (int(Outcome_1[i,0]) == 0 and int(Outcome_1[i,1]) == 0 and int(Outcome_1
                     probability[0,i] = (1 - norm.cdf(Profit_1m_1enter[0][i]/Scale)) * (1 - no

                 elif (int(Outcome_1[i,0] == 0) and int(Outcome_1[i,1]) == 0 and int(Outcome_1
                     probability[0,i] = (1 - norm.cdf(Profit_1m_notenter[0][i]/Scale)) * (1 - n
             #PRobability = tf.constant(probability)
             return probability

In [14]: def MLE(x):
             return -np.sum(np.log(Probability(x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]))
```

```
In [80]: x0 = np.array([2,1,0, 2,2,1, -2, 1])
         res = minimize(MLE, x0, method='nelder-mead', options={'xtol': 1e-3, 'disp': True})

Optimization terminated successfully.
         Current function value: 2998.618478
         Iterations: 747
         Function evaluations: 1094


In [81]: print(res.x)

[ 1.84767614  0.88428146  0.08938407  1.09808428  1.11875927  0.45021849
 -1.29475127  0.9586243 ]
```

2.5 Results

Variable	Estimated Value
alpha1	1.84767614
alpha2	0.88428146
alpha3	0.08938407
beta1	1.09808428
beta2	1.11875927
beta3	0.45021849
delta	-1.29475127
Scale	0.9586243

3. References

[1] Ellickson, P. B. and Misra, S. (2011) 'Estimating Discrete Games', Marketing Science, 30(6), pp. 997–1010.

[2] Zhaokun Zhang, Zhesheng Zheng. (2019) 'Estimation of Discrete Game and its Realization in R'

[3] Jiaming Mao, 'Classication and Discrete Choice', https://jiamingmao.github.io/data-analysis/assets/Lectures/Classication_and_Discrete_Choice_Models.pdf

[4] Timothy F. Bresnahan and Peter C. Reiss (1990) 'Entry in Monopoly Markets', The Review of Economic Studies, 57(4), p. 531.

[5] Timothy F. Bresnahan and Peter C. Reiss (1991) 'Entry and Competition in Concentrated Markets', Journal of Political Economy, 99(5), p. 977.