**DIA; ADJABLI; LONGUEPEE; RAYMOND:**

**Problem IV: Getting confidence in market equilibriums – an agent-based perspective:**

**Case 1: Traditional Cournot equilibriums:**

* For all cases, the cost of productionis equal to 0.
* For each firm we can write the profit

***Monopoly:***

Let the quantity sold on the market by the monopolist: We have the following equations:

* = (

We know that the firm wants to maximize this profit leading to:

Hence:

For the monopoly, we have an equilibrium quantity .

The market price is equal to

***Duopoly:***

Let the quantity sold on the market by the firm 1.

Let the quantity sold on the market by the firm 2: We have the following equations:

* + = (
  + = (

Each firm wants to maximize its profit leading to:

We can solve the system:

*For the firm 1:*

*For the firm 2:*

We can write the system with a matrix to ease solving:

With the invert of

We have:

Finally, it gives us the following market price:

***Triopoly:***

Let the quantity sold on the market by the firm 1.

Let the quantity sold on the market by the firm 2.

Let the quantity sold on the market by the firm 3: We have the following equations:

* + = (
  + = (
  + =

Each firm wants to maximize its profit leading to:

We can solve the system:

*For the firm 1:*

*For the firm 2:*

*For the firm 2:*

We can write the system with a matrix to ease solving:

With the invert of

We have:

With the invert of

Finally, it gives us the following market price:

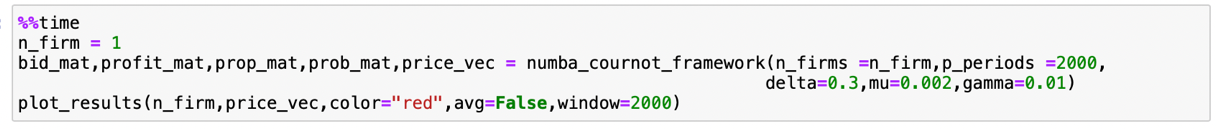
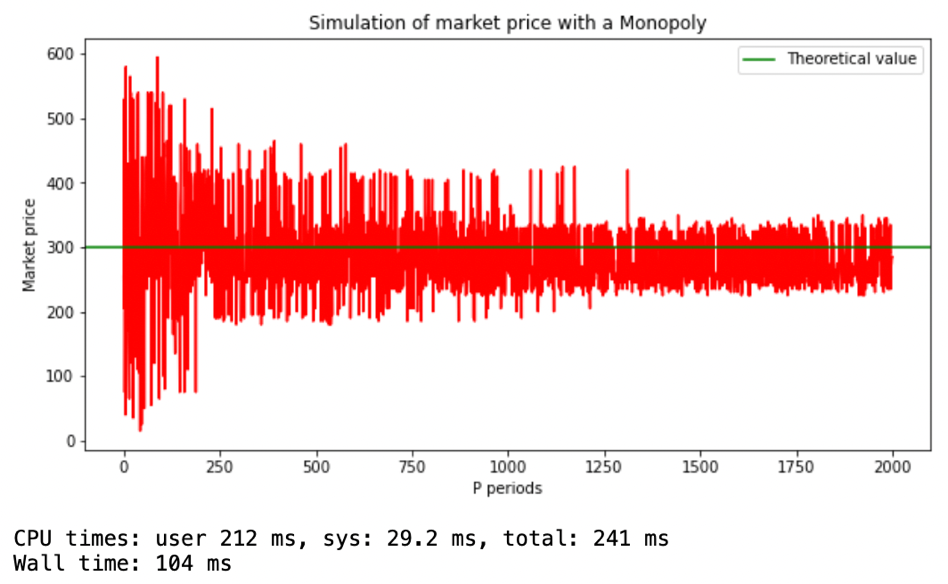
**Synthesis:**

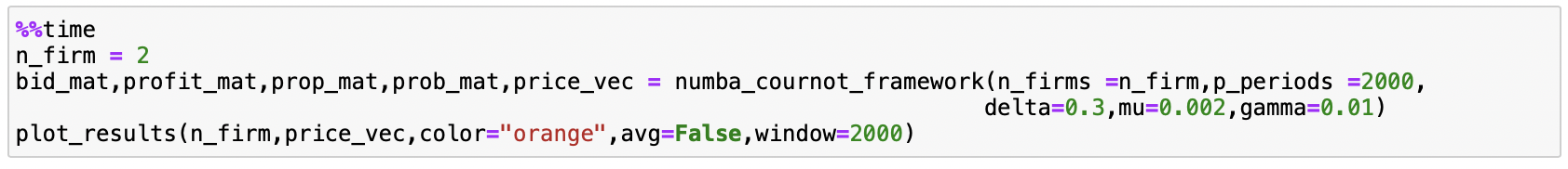
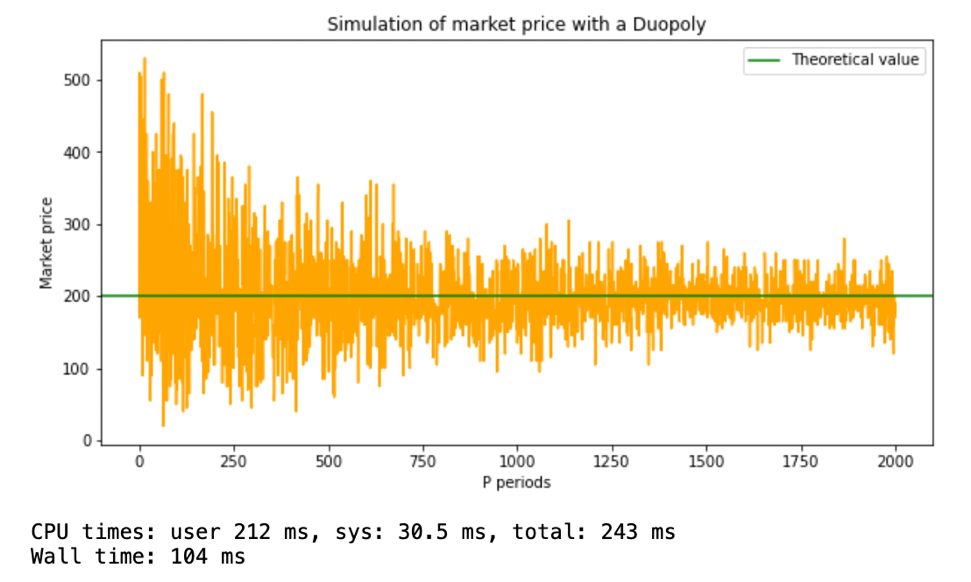
We can summarize these results in the following table:

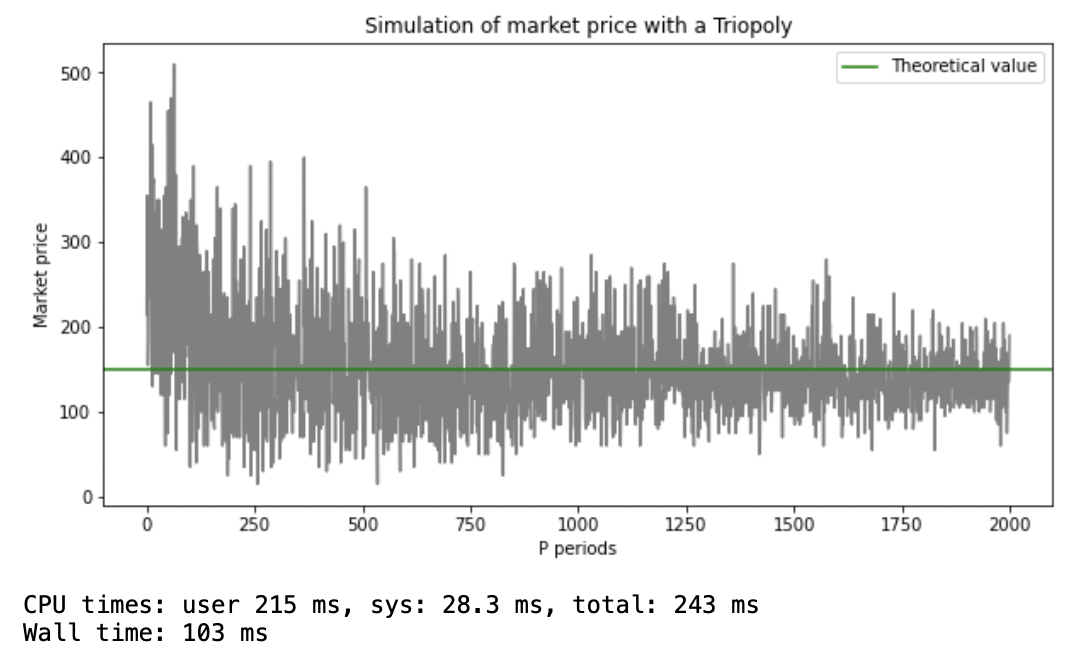
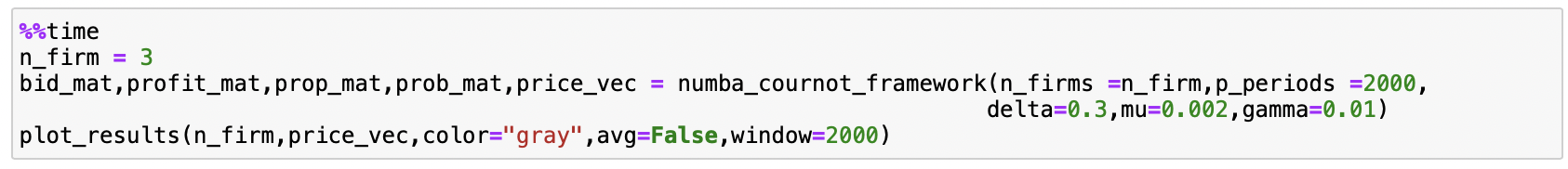
|  |  |
| --- | --- |
| **Market structure** | **Price value [** |
| Monopoly | 300 |
| Duopoly | 200 |
| Triopoly | 150 |
| -opoly | With |

**Case 2: Roth & Erev, a Reinforcement learning process**

2 & 3. Studying convergence and plotting.

*Monopoly:*

*Duopoly:*

*****Triopoly:*

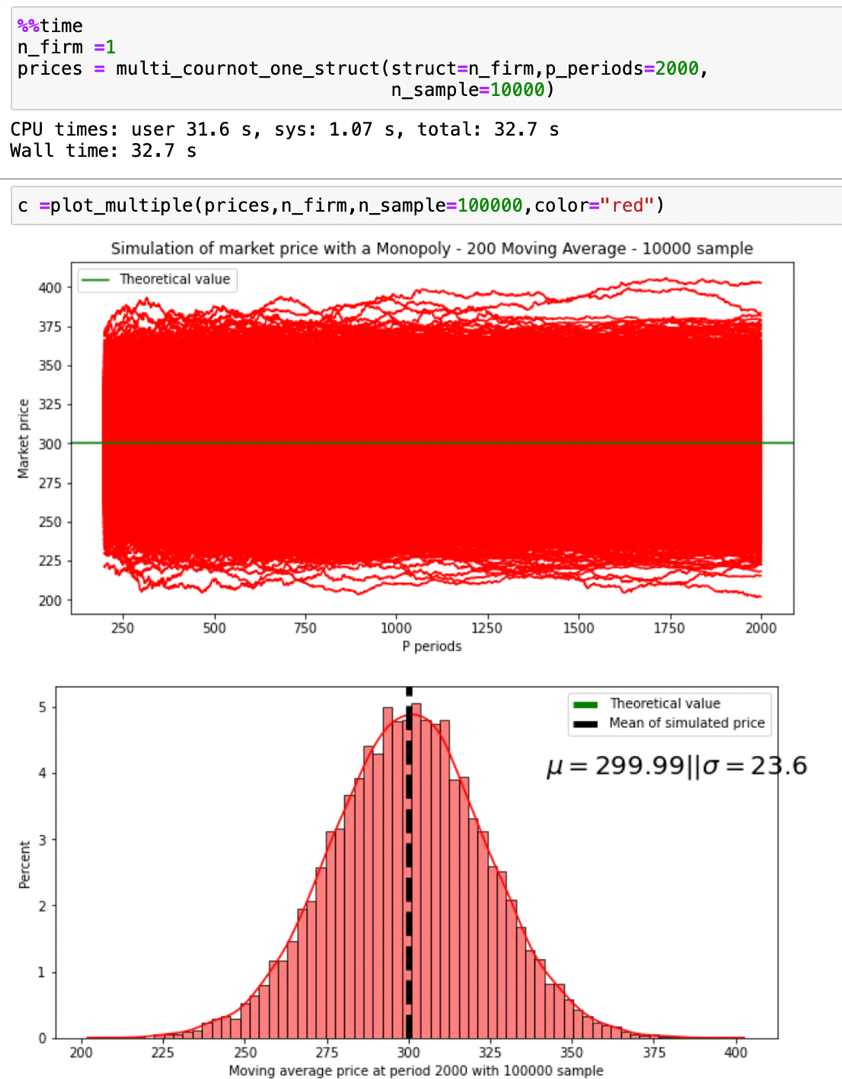
For each market structure, we can see that each simulation stabilizes around the theoretical value, after 1000 periods. Indeed, we have for each market structure the simulation and the theoretical value that we have computed before. Our framework seems to be reliable.

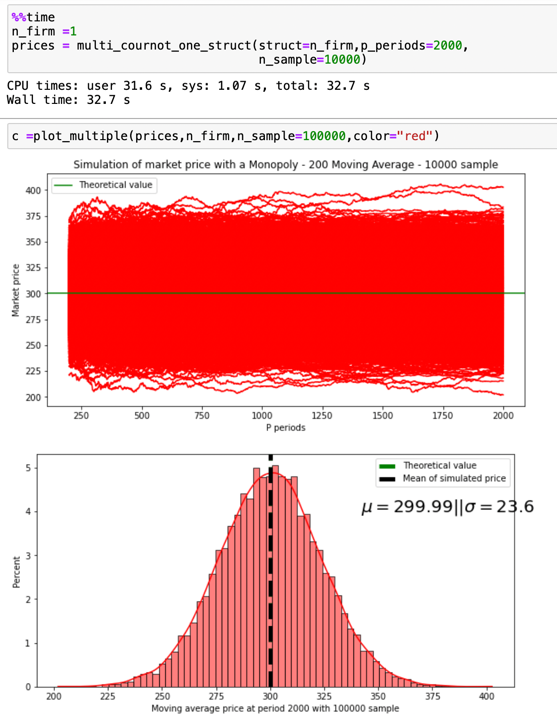
Besides regarding time performance, as we have used numpy library in our code (cf. appendix) and [numba](https://numba.pydata.org/) (a code compiler for python designed for numpy), we converge to a stationary state really quickly.

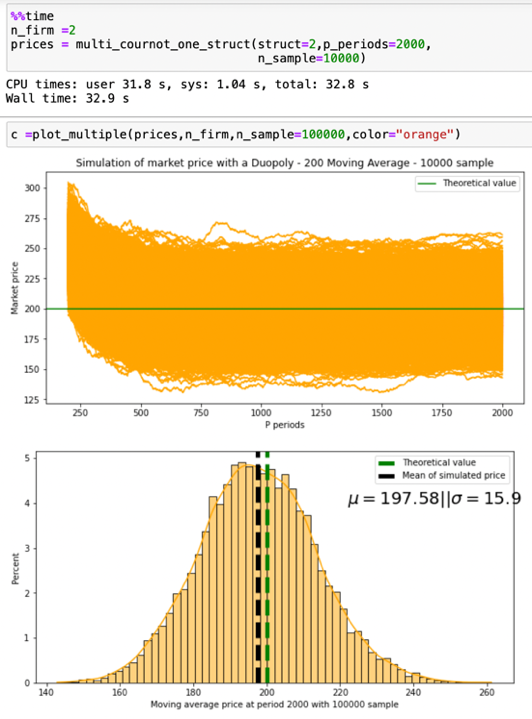
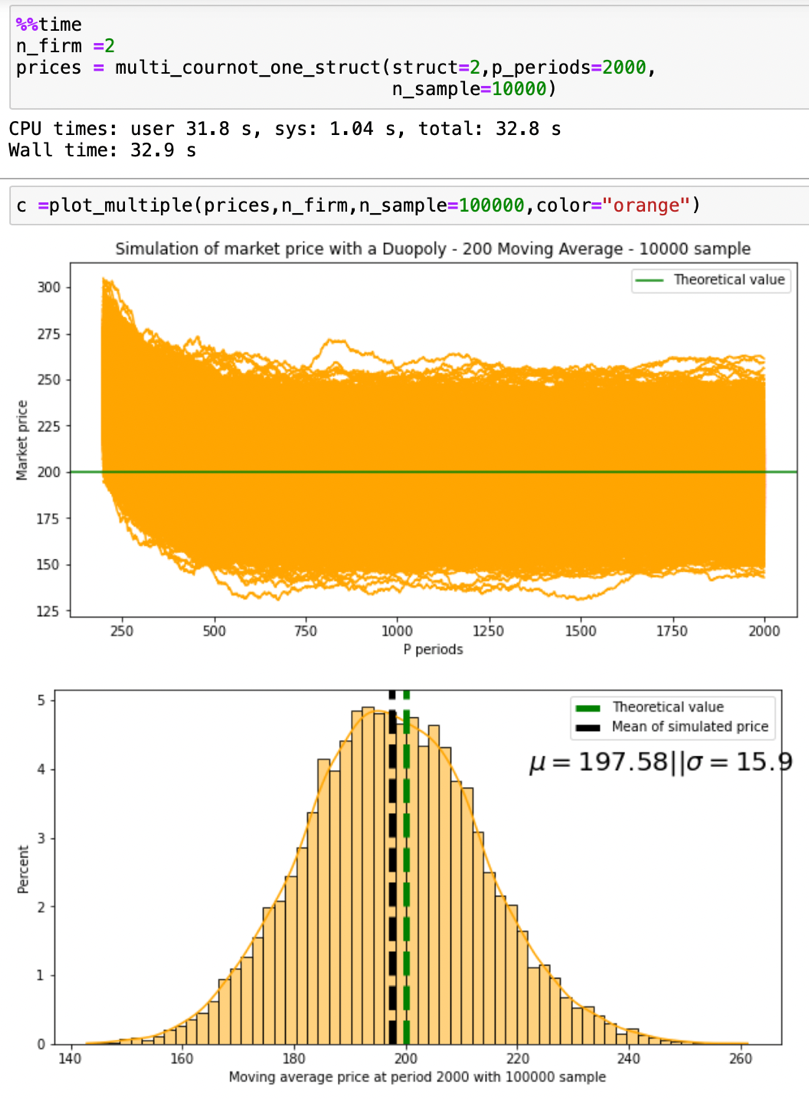
**4. Multiple simulation results:**

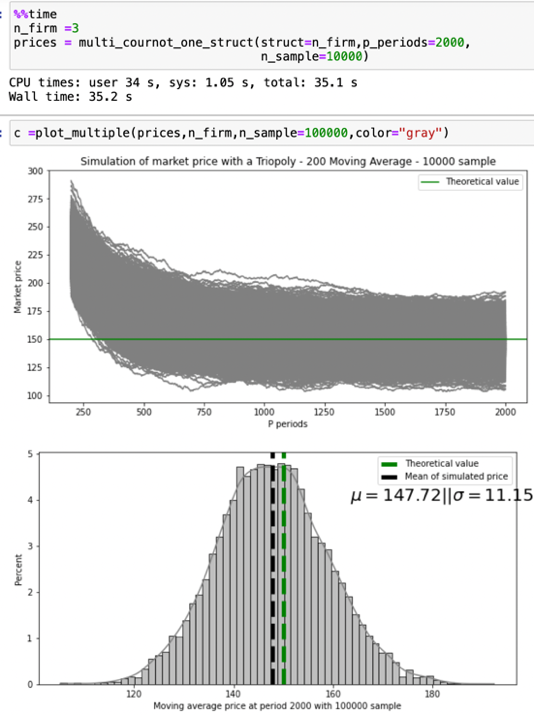
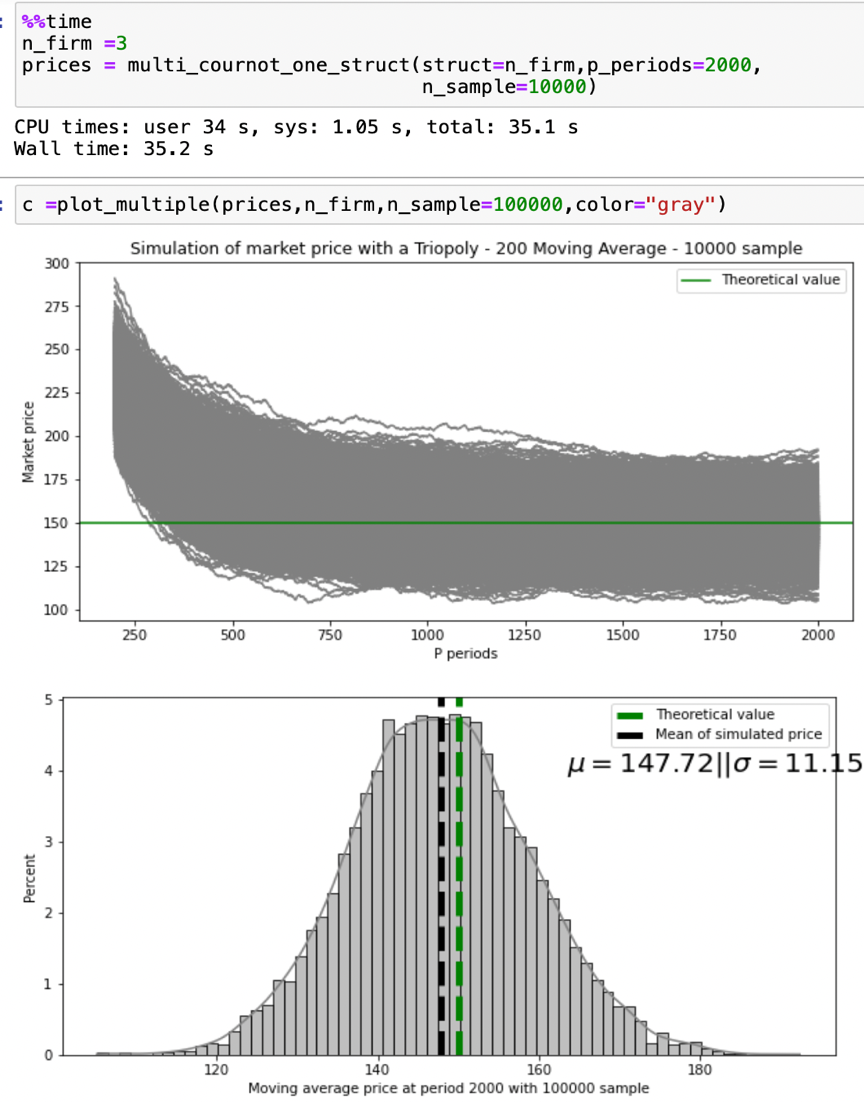
*As our framework can do a lot of simulations in a reduce amount of time, we have decided to run 10 000 samples to see what are the results. For each sample, we are looking at the 200 Moving Average.*

*We will look at the last moving average i.e, the moving average price at the last period of simulation.*

**Hence, we can plot for each market structures, the simulation and the associated price price, with the mean and the standard deviation:

*******Monopoly:*

*Duopoly:*

*Triopoly:*

**5. Results comparison**

We can summarize this in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Market Structure | Simulation results | | | Part 1.  Theoretical value |
| - | – |  |
| Monopoly | 299.99 | 23.6 | 7.8% | 300 |
| Duopoly | 197.58 | 15.9 | 8.0% | 200 |
| Triopoly | 147.72 | 11.15 | 7.5% | 150 |

As we have conduced 10 000 samples thanks to the speed of our framework, we can say that we have consistent result regarding the theoretical value computed in the Part1. We obtain gaussian distribution when we look at the , which has interesting properties as we obtain a ratio which is similar across all simulations.

**6. Commentaries.**

What we can say regarding this “agent-based simulation” model is that is does provide consistent results comparing to the results obtained with the theoretical approach. However, parameter tuning is a key to obtain relevant results. As we have been building the model, we have seen that some parameters such as plays a critical role in the simulation. A too high value of would penalize untested bid too early and will prevent us from reaching the market equilibrium value.

Besides, the learning mechanism linked to this model is really interesting, as we could imagine that on a new market, firm would try different production configuration, and adjust over time based on the previous result (or profit), which is what it is done here. However, the random aspect of the production pick might not fit the reality even though evolving propension captures the evolution of preferences of the actor.

To summarize, we can say that these model offers relevant insights regarding the formation of equilibrium for different market structures, but parameters are key to approach theoretical market equilibrium. This model does confirm the reliability and the confidence that we can have in these theories and also *reinforces* our faith of reinforcement learning model for games theory.

*Code appendix is available on next page.*

**Cournot framework – Python code.**

The code is available on [github](https://github.com/keyserwood/EcoIndus/blob/master/1.0-er_industrial_economy.ipynb).

*Global framework*

@njit(parallel=False) # Using njit refers to numba, a compiler in python, which convert python to C equivalent.

# By compiling the functions with function we can increase dramatically the speed

def numba\_cournot\_framework(n\_firms, p\_periods,initial\_propension=90000,delta=0.3,mu=0.002,gamma=0.01,total\_demand=600):

"""Cournot framework - Simulate market situations

Args :

n\_firms (int): Number of firms : n\_firms>=1

p\_periods (int): Number of simulation periods

initial\_propension (float) : initial\_propension for each value

mu (float) : extinction in finite time

gamma (float) : gradual forgetting

delta (float) : persistent local experimentation

total\_demand (int): total demand of market

Returns :

bid\_mat (np.matrix): Matrix of bid for each firm at each period

profit\_mat (np.matrix): Matrix of profit for each firm at each period

prop\_mat (np.matrix) : Propension matrix for each bid for each firm for each period

price\_vec (np.array) : Market price for each period"""

s\_discrete = int((120/n\_firms)//1) #Max of allowable bids - Integer part of 120/n

bid\_mat = np.zeros((p\_periods,n\_firms)) # Bid matrix : (p\_periods,n\_firms)

profit\_mat = np.zeros((p\_periods,n\_firms)) # Profit matrix : (p\_periods,n\_firms)

prop\_mat = np.zeros((s\_discrete,n\_firms,p\_periods)) # Propensions matrix : (S,n\_firms,p\_periods)

prop\_mat[:,:,0] = initial\_propension # Initialize propension - setting up for first iteration

secrete\_bids = np.arange(1,s\_discrete+1) #Set of allowable bids

bid\_possibilities = np.arange(1,s\_discrete+1)\*5 #Bids possibities (\*5) [5,10 ...5\*S]

price\_vec = np.zeros(p\_periods) # Price vector : (p\_periods)

for period in range(p\_periods):

#For each period we compute the propbability matrix based on propension vector

prob\_mat = prop\_mat[:,:,period]/np.sum(prop\_mat[:,:,period],axis=0) #probability matrix

for firm in prange(n\_firms):

### Across all firms

# We determine for each firm the bid

# rand\_choice\_nb(vec, prob) returns a value in array based on a probability vector

# prob[:,firm] : probability for each bid for a given firm

bid\_pick = rand\_choice\_nb(bid\_possibilities, prob=prob\_mat[:,firm])

bid\_mat[period,firm] = bid\_pick # We set bid picked for the firm at this period

# Once bid are made we can compute price

market\_price = total\_demand-np.sum(bid\_mat[period,:]) # P(Q)=600-SUM(Qi)

price\_vec[period] = market\_price # We save the price of this period

profit\_mat[period,:] = market\_price\*bid\_mat[period,:] # We compute the profit (no cost of prod.)

# Need to recompute propensions for next iterations

for firm in range(n\_firms):

lambda\_ = bid\_mat[period,firm]/5 #Get lambda of the firm i.e the bid picked at this period

profit = profit\_mat[period,firm] #Retrieve this associated profit

# For each allowable bid in 1..S

for s\_bid in range(s\_discrete):

# Loop bound reinforcement

######### Part 1 of reinforcement #######################:

# Current propension for this period

propension = prop\_mat[s\_bid,firm,period]

# Warning : s\_bid+1 because python starts at 0

# We resort to a function that recalculates propension given multiple input

new\_propension = propensions\_reinforcement\_part1(s\_bid+1,propension,

profit,lambda\_,delta,gamma)

# Set propension for next period

if period + 1 < p\_periods:

## Do not outbound the number of periods

prop\_mat[s\_bid,firm,period+1] = new\_propension # Set new propension

############## Part 2 : adjust regarding mu #################

if period + 1 < p\_periods:

## Do not outbound the number of periods

propension\_vec = prop\_mat[:,firm,period+1] # Extract propension vector

# We look at the new propension for next period, and before going further

# we adjust propension regarding mu

propension\_vec\_mu\_filter = (propension\_vec/np.sum(propension\_vec))>mu # rs/sum(rs)>mu

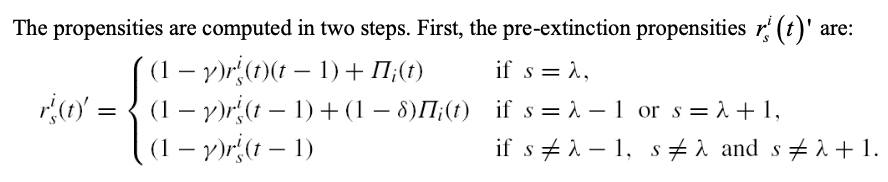
# By multipling like that we implement the indicator function

prop\_mat[:,firm,period+1] = propension\_vec\*propension\_vec\_mu\_filter # (0,1,0,0)\*(propension)

# We can go to the next period !

return (bid\_mat,profit\_mat,prop\_mat,prob\_mat,price\_vec)

Recompute propensions function:

****

@njit #Numba decorator – enable use in Cournot framework

def propensions\_reinforcement\_part1(s\_bid,propension,profit,lambda\_,delta,gamma):

"""Reinforcing s\_bid : s\_bid in 1..S

Args :

s\_bid (int): s\_bid value

propension (float) : propension of s\_bid

profit (float) : profit at period p

lambda\_ (int) : s\_bid picked at period p

gamma (float) : forget rate

delta (float) : propension increaser

Returns:

new\_propension (float): reinforced propension

# First step : comparison between previously picked s\_bid

# and current s\_bid

if s\_bid==lambda\_:

new\_propension = (1-gamma)\*propension+profit

elif s\_bid == lambda\_-1 or s\_bid == lambda\_ +1:

new\_propension = (1-gamma)\*propension+profit\*(1-delta)

else:

new\_propension = (1-gamma)\*propension

*Plotting simulation:*

def plot\_results(n\_firm,price\_vec,color,avg=False,window=20):

"""Plot market price after a simulation fo a configuration. Can display simulate price

or a moving average of it"""

dic = {1:"Monopoly",2:"Duopoly",3:"Triopoly"}

if n\_firm in dic:

t = dic[n\_firm] #Retrieve label

else:

t = f'{n\_firm}-opoly'

plt.figure(figsize=(10,5))

plt.xlabel("P periods")

plt.ylabel("Market price")

if avg:

price\_vec = pd.Series(price\_vec).rolling(window).mean()

plt.plot(range(len(price\_vec)),price\_vec,c=color)#Display price

plt.title(f"Simulation of market price with a {t} - {window} Moving Average")

else:

plt.plot(range(len(price\_vec)),price\_vec,c=color)

plt.title(f"Simulation of market price with a {t}")

plt.axhline(600/(n\_firm+1),color="green",label="Theoretical value")

plt.legend()

plt.show()