DIA; ADJABLI; LONGUEPEE; RAYMOND:

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

Case 1: Traditional Cournot equilibriums:

$$\mathrm{P(Q)} = 600 - Q$$

- For all cases, the cost of production c is equal to 0.
- For each firm i we can write the profit $\Pi_i = P(Q_i).Q_i - c = P(Q_i).Q_i$

Monopoly:

Let q_1: the quantity sold on the market by the monopolist: We have the following equations:

- $\mathrm{P(Q_1)} = 600 - Q_1$
- $\Pi_1 = P(Q_1).Q_1 = (600 - Q_1).Q_1$

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

$$\frac{\partial \Pi_1(Q_1)}{\partial Q_1} = 0$$

Hence:

$$\frac{\partial[(600 - Q_1).Q_1]}{\partial Q_1} = 0 \Leftrightarrow 600 - 2Q_1 = 0 \Leftrightarrow Q_1 = \frac{600}{2} = 300$$

For the monopoly, we have an equilibrium quantity $Q_1 = 300$.
The market price is equal to $P(Q_1) = 600 - Q_1 = 600 - 300$

$$\boldsymbol{P = 300}$$

Duopoly:

Let q_1: the quantity sold on the market by the firm 1.
Let q_2: the quantity sold on the market by the firm 2: We have the following equations:

- $\mathrm{P(Q_1}, Q_2) = 600 - (Q_1 + Q_2)$
- $\Pi_1 = P(Q_1, Q_2).Q_1 = (600 - (Q_1 + Q_2).Q_1$
- $\Pi_2 = P(Q_1, Q_2).Q_2 = (600 - (Q_1 + Q_2).Q_2$

Each firm wants to maximize its profit leading to:

$$\frac{\partial \Pi_1(Q_1, Q_2)}{\partial Q_1} = 0$$
$$\frac{\partial \Pi_2(Q_1, Q_2)}{\partial Q_2} = 0$$

We can solve the system:
For the firm 1:

$$\frac{\partial[(600 - (Q_1 + Q_2)).Q_1]}{\partial Q_1} = 0 \Leftrightarrow 600 - 2Q_1 - Q_2 = 0 \Leftrightarrow Q_1 = \frac{600 + Q_2}{2} = 300$$

For the firm 2:

$$\frac{\partial[(600 - (Q_1 + Q_2)).Q_2]}{\partial Q_2} = 0 \Leftrightarrow 600 - 2Q_2 - Q_1 = 0 \Leftrightarrow Q_2 = \frac{600 + Q_1}{2} = 300$$

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

$$A.Q = B \Leftrightarrow \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = \begin{pmatrix} 600 \\ 600 \end{pmatrix}$$

With the invert of A, A^{-1}:

$$A^{-1} = \frac{1}{3}\begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$$

We have:

$$Q = A^{-1}.B$$

$$\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}.\begin{pmatrix} 600 \\ 600 \end{pmatrix}$$

$$\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} 600 \\ 600 \end{pmatrix} = \begin{pmatrix} 200 \\ 200 \end{pmatrix}$$

Finally, it gives us the following market price: $\mathrm{P}(\mathrm{Q}_1, Q_2) = 600 - (Q_1 + Q_2) = 200$

$$\boldsymbol{P = 200}$$

Triopoly:

Let q_1: the quantity sold on the market by the firm 1.
Let q_2: the quantity sold on the market by the firm 2.
Let q_3: the quantity sold on the market by the firm 3: We have the following equations:

- $\mathrm{P}(\mathrm{Q}_1, Q_2, Q_3) = 600 - (Q_1 + Q_2 + Q_3)$
- $\Pi_1 = P(\mathrm{Q}_1, Q_2, Q_3).Q_1 = (600 - (Q_1 + Q_2 + Q_3).Q_1$
- $\Pi_2 = P(\mathrm{Q}_1, Q_2, Q_3).Q_2 = (600 - (Q_1 + Q_2 + Q_3).Q_2$
- $\Pi_3 = P(\mathrm{Q}_1, Q_2, Q_3).Q_3 = 600 - (Q_1 + Q_2 + Q_3).Q_3$

Each firm wants to maximize its profit leading to:

$$\frac{\partial \Pi_1(Q_1, Q_2, Q_3)}{\partial Q_1} = 0$$

$$\frac{\partial \Pi_2(Q_1, Q_2, Q_3)}{\partial Q_2} = 0$$

$$\frac{\partial \Pi_3(Q_1, Q_2, Q_3)}{\partial Q_3} = 0$$

We can solve the system:

For the firm 1:

$$\frac{\partial\left[\left(600 - (Q_1 + Q_2 + Q_3)\right).Q_1\right]}{\partial Q_1} = 0 \iff 600 - 2Q_1 - Q_2 - Q_3 = 0$$

$$Q_1 = \frac{600 + Q_2}{2} = 300$$

For the firm 2:

$$\frac{\partial[(600 - (Q_1 + Q_2 + Q_3)).Q_2]}{\partial Q_2} = 0 \iff 600 - 2Q_2 - Q_1 - Q_3 = 0$$

$$Q_2 = \frac{600 + Q_1 + Q_3}{2} = 300$$

For the firm 2:

$$\frac{\partial[(600 - (Q_1 + Q_2 + Q_3)).3]}{\partial Q_3} = 0 \iff 600 - 2Q_3 - Q_1 - Q_2 = 0$$

$$Q_3 = \frac{600 + Q_2 + Q_3}{2} = 300$$

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

$$A.Q = B \iff \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}.\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} = \begin{pmatrix} 600 \\ 600 \\ 600 \end{pmatrix}$$

With the invert of A, A^{-1}:

$$A^{-1} = \frac{1}{4}\begin{pmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{pmatrix}$$

We have:

$$Q = A^{-1}.B$$

$$\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{pmatrix}.\begin{pmatrix} 600 \\ 600 \\ 600 \end{pmatrix}$$

With the invert of A, A^{-1}:

$$\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 600 \\ 600 \\ 600 \end{pmatrix} = \begin{pmatrix} 150 \\ 150 \\ 150 \end{pmatrix}$$

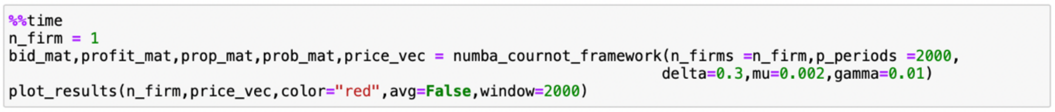
Finally, it gives us the following market price: $\mathrm{P}(\mathrm{Q}_1, Q_2, Q_3) = 600 - (Q_1 + Q_2 + Q_3) = 150$

$$\boldsymbol{P = 150}$$

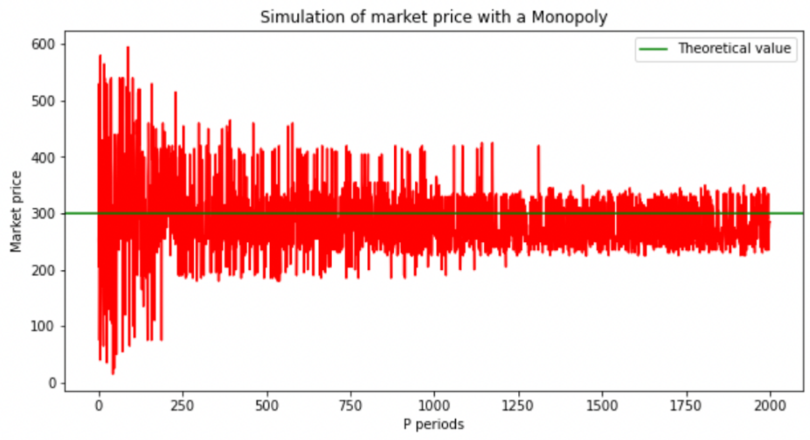
Synthesis:
We can summarize these results in the following table:

Market structure	**Price value $[P(Q) = 600 - Q]$**
Monopoly	300
Duopoly	200
Triopoly	150
n-opoly	$$P = \frac{600}{n+1}$$ With $n \geq 1$.

Case 2: Roth & Erev, a Reinforcement learning process
2 & 3. Studying convergence and plotting.
Monopoly:



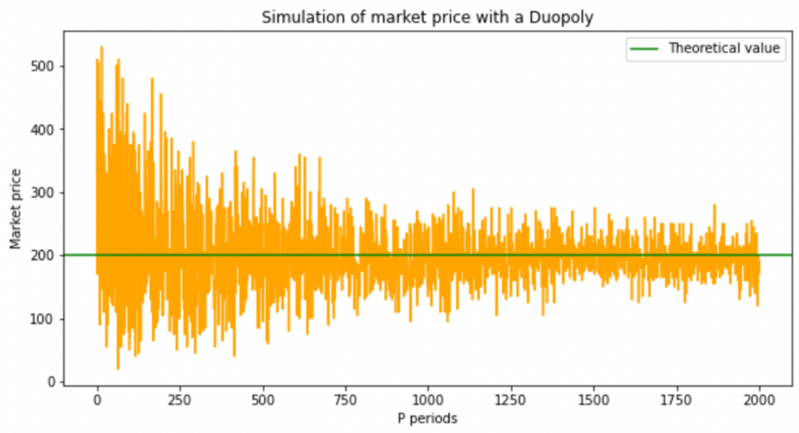
```
%%time
n_firm = 1
bid_mat,profit_mat,prop_mat,prob_mat,price_vec = numba_cournot_framework(n_firms =n_firm,p_periods =2000,
                                                     delta=0.3,mu=0.002,gamma=0.01)
plot_results(n_firm,price_vec,color="red",avg=False,window=2000)
```



```
CPU times: user 212 ms, sys: 29.2 ms, total: 241 ms
Wall time: 104 ms
```

Duopoly:

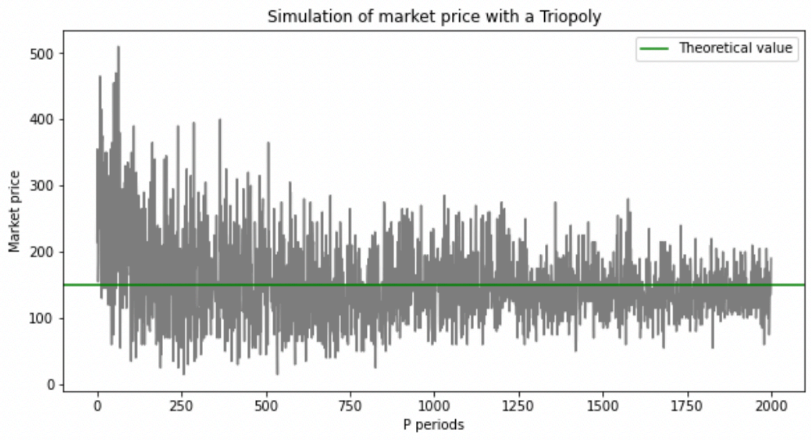
```
%%time
n_firm = 2
bid_mat,profit_mat,prop_mat,prob_mat,price_vec = numba_cournot_framework(n_firms =n_firm,p_periods =2000,
                                                                         delta=0.3,mu=0.002,gamma=0.01)
plot_results(n_firm,price_vec,color="orange",avg=False,window=2000)
```



```
CPU times: user 212 ms, sys: 30.5 ms, total: 243 ms
Wall time: 104 ms
```

Triopoly:

```
%%time
n_firm = 3
bid_mat,profit_mat,prop_mat,prob_mat,price_vec = numba_cournot_framework(n_firms =n_firm,p_periods =2000,
                                                                         delta=0.3,mu=0.002,gamma=0.01)
plot_results(n_firm,price_vec,color="gray",avg=False,window=2000)
```



```
CPU times: user 215 ms, sys: 28.3 ms, total: 243 ms
Wall time: 103 ms
```

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 (a code compiler for python designed for numpy), we converge to a stationary state really quickly.

4. Multiple simulation results:

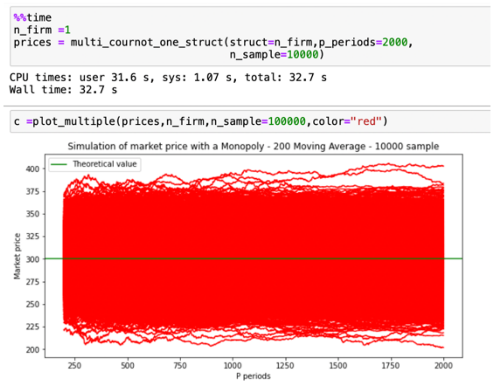
As our framework can do a lot of simulations in a reduce amount of time, we have decided to run 10 000 samples to see what are the results. For each sample, we are looking at the 200 Moving Average.

$$MA_{i,200} = \frac{1}{200} \sum_{j=i}^{i+200-1} p_j$$

We will look at the last moving average i.e $MA_{2000,200}$ *, 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:

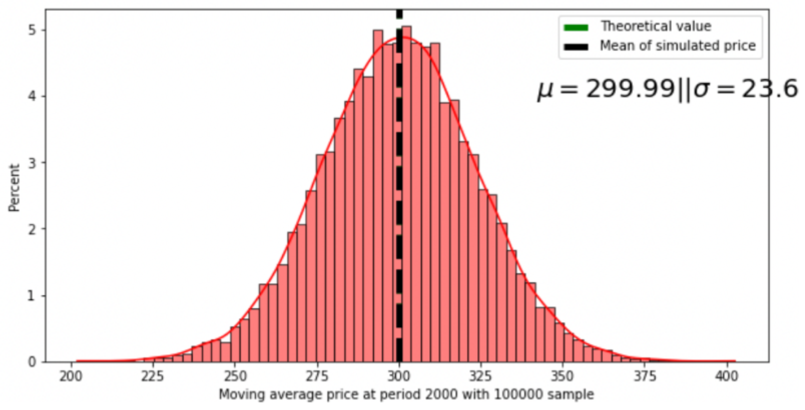


```
%%time
n_firm =1
prices = multi_cournot_one_struct(struct=n_firm,p_periods=2000,
                                  n_sample=10000)
```

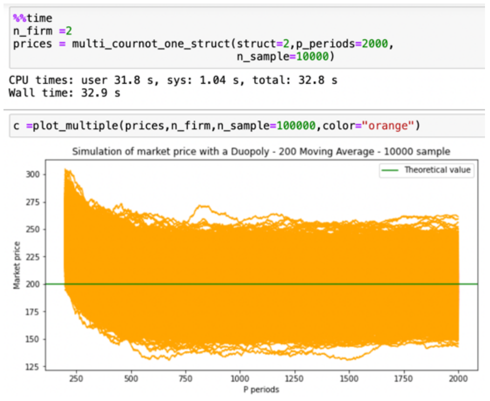
```
CPU times: user 31.6 s, sys: 1.07 s, total: 32.7 s
Wall time: 32.7 s
```

```
c =plot_multiple(prices,n_firm,n_sample=100000,color="red")
```

Simulation of market price with a Monopoly - 200 Moving Average - 10000 sample
Theoretical value
Market price
P periods



Duopoly:

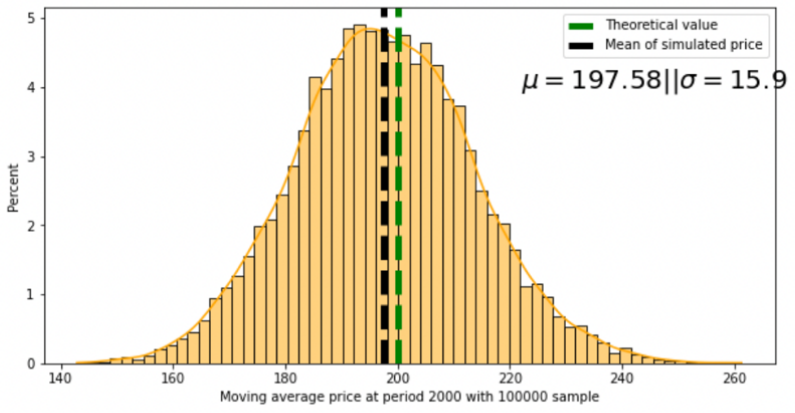


```
%%time
n_firm =2
prices = multi_cournot_one_struct(struct=2,p_periods=2000,
                                  n_sample=10000)
```

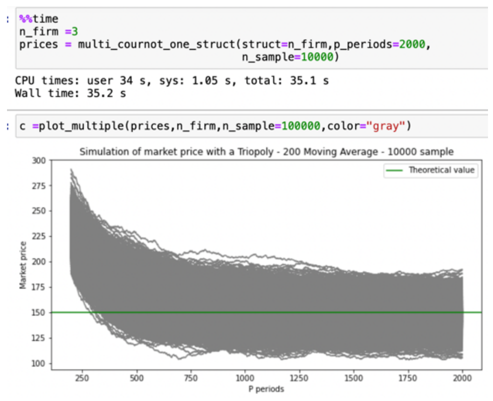
```
CPU times: user 31.8 s, sys: 1.04 s, total: 32.8 s
Wall time: 32.9 s
```

```
c =plot_multiple(prices,n_firm,n_sample=100000,color="orange")
```

Simulation of market price with a Duopoly - 200 Moving Average - 10000 sample
Theoretical value
Market price
P periods



Triopoly:

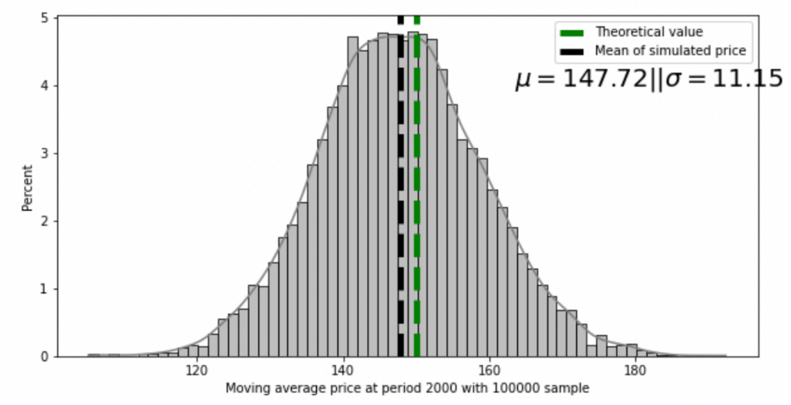


```
%%time
n_firm =3
prices = multi_cournot_one_struct(struct=n_firm,p_periods=2000,
                                  n_sample=10000)
```

```
CPU times: user 34 s, sys: 1.05 s, total: 35.1 s
Wall time: 35.2 s
```

```
c =plot_multiple(prices,n_firm,n_sample=100000,color="gray")
```

Simulation of market price with a Triopoly - 200 Moving Average - 10000 sample
Theoretical value
Market price
P periods



5. Results comparison

We can summarize this in the following table:

Market Structure	Simulation results			Part 1. Theoretical value μ_{theo}
	$MA_{2000,200}$ - μ	$MA_{2000,200} - \sigma$	$\frac{\sigma}{\mu}$	
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 $MA_{2000,200}$, which has interesting properties as we obtain a $\frac{\sigma}{\mu}$ 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 μ plays a critical role in the simulation. A too high value of μ 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.

Global framework

```python
@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:

The propensities are computed in two steps. First, the pre-extinction propensities $r_s^i(t)'$ are:

$$r_s^i(t)' = \begin{cases} (1-\gamma)r_s^i(t)(t-1) + \Pi_i(t) & \text{if } s = \lambda, \\ (1-\gamma)r_s^i(t-1) + (1-\delta)\Pi_i(t) & \text{if } s = \lambda - 1 \text{ or } s = \lambda + 1, \\ (1-\gamma)r_s^i(t-1) & \text{if } s \neq \lambda - 1,\ s \neq \lambda \text{ and } s \neq \lambda + 1. \end{cases}$$

```python
@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:

```python
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()
```