Micro Simplified Hayekian Market

Matteo Morini[1] and Pietro Terna[2]

October 23, 2018

[1]University of Torino, Italia
[2]University of Torino, Italia

Abstract

We propose a simplified version of the Hayek's decentralized market hypothesis, considering elementary processes of price adaptation in exchanges.

Sections 1 and 2 report the technical setup and the structure of the model. In Section 3 we introduce an elementary agent-based model of a market, with emergent (quite interesting) price dynamics.

A counter example is also introduced in Section 4, showing that—with tiny modification—we generate implausible price dynamics.

In Appendix A we report some technical analyses of the cases with unmatching numbers of buyers and sellers. These analyses are strongly related the the Oligopoly[1] simulation project.

In Appendix B we deepen the problem of the agents that seem to be idle, having the level of their reservation price too low to buy. The Appendix is mainly of interest to better understand the price dynamics within the Oligopoly project.

[1] Clic to go to https://terna.github.io/oligopoly/

Contents

List of Figures

Introduction to a Micro Simplified Hayekian Market

The purpose of the note is that of introducing a very simple agent-based model of a market, with emergent (quite interesting) price dynamics.

A counter example is also introduced, showing how—with tiny modifications—we generate implausible/impossible price dynamics.

The code uses the IPython[2] language (an interactive layer upon Python[3] using the Jupyter[4] infrastructure) and can be dowloaded from https://github.com/terna/microHayekianMarket via the *Clone or download* button; it is also possible to run it directly on line thanks to the MyBinder project[5].

A suggested reading about Hayek is a quite recent paper of Bowles *et al.* (2017).

Quoting from the introduction:

> Friedrich A. Hayek (1899-1992) is known for his vision of the market economy as an information processing system characterized by spontaneous order, the emergence of coherence through the independent actions of large numbers of individuals each with limited and local knowledge, coordinated by prices that arise from decentralized processes of competition.

A simplified version—proposed here—is that of considering decentralized elementary processes of price adaptation in exchanges, with surprising results.

1 The technical setup

The IPython (or Python 3.x) code requires the following setup to start:

Listing 1: Setup of the program

```
%pylab inline
%pylab inline
import statistics as s
import numpy as np
import pylab as plt
from ipywidgets import Output
```

[2] https://ipython.org
[3] https://www.python.org
[4] http://jupyter.org
[5] Click to go to https://mybinder.org/v2/gh/terna/microHayekianMarket/master?filepath=microHayekianMarket.ipynb

```
from IPython.display import clear_output
from IPython.display import display
import time
import math
```

`%pylab inline` is a *magic* command of Jupyter.

2 The structure of the model and the *warming up* phase

Our agents are simply prices, to be interpreted as reservation prices.[6]

The agents act over the time that we organize in cycles; within a *cycle*, all the buyers act once, in random order.

We have two price vectors: pL^b with item pL^b_i for the buyers, and pL^s with item pL^s_j for the sellers. The i^{th} or the j^{th} elements of the vectors are prices, but we can use them also as agents.

Both in the simplified Hayekian perspective (Section 3) and in the unstructured one (Section 4) we have to pre-run a *warning up* action. This happens automatically, calling the specific function in the beginning of both the cases.

With the *warming up* phase, we define:

- d_0 - the lower bound of the random uniform numbers, both for the buyers and the sellers, in the warming up phase;

  in the running phase, the lower bound is 0;
- d_1 - the upper bound of the random uniform numbers for the buyers;
- d_2 - the upper bound of the random uniform numbers for the sellers;
- $nCycles$ - number of simulation cycles;
- $nBuyers$ - number of the buyers;
- $nSellers$ - number of the sellers;
- $seed$ - the seed of the random numbers;
- the initial buyer i reservation price, different for each buyer: $p_{b,i} = \frac{1}{1+u_i}$ with $u_i \sim \mathcal{U}(d_0, d_1)$;
- the initial seller j reservation price, different for each seller: $p_{s,j} = 1 + u_j$ with $u_j \sim \mathcal{U}(d_0, d_2)$;
- $buyersSellersRatio$ - the ratio $\frac{nBuyers}{nSellers}$;
- $sellersBuyersRatio$ - the ratio $\frac{nSellers}{nBuyers}$;
- $usingRatios$ - a logic variable activating limitations to d_1 or d_2
- $squeezeRate$ - always < 1, as further compression of d_1 or d_2
- $usingSqueezeRate$ - a logic variable to further squeeze d_1 or d_2

[6] The *max* price a buyer could pay and the *min* one a seller could accept.

With $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, sorting in decreasing order the vector pL^b and in increasing order the vector pL^s, we obtain in Fig. 1 two not overlapping price sequences that we can interpret as a demand curve (red) and an offer one (blue).

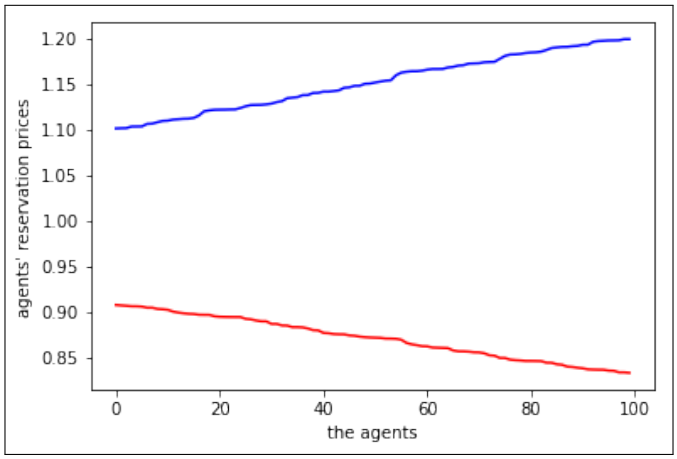

Figure 1: An example of initial not overlapping demand curve (red) and offer curve (blue)

To generate new examples related to Section 3 and to Section 4, it is necessary to repeat this phase. This happens automatically, calling the specific function in the beginning of both the cases.

The IPython (or Python 3.x) code is:

Listing 2: Warming up of the model

```
# warming up

# execute before both:
#                       - the Hayekian perspective or
#                       - the unstructured case
def warmingUp():
    global nCycles, nBuyers, nSellers,\
        buyersSellersRatio, sellersBuyersRatio,\
        usingRatios, usingSqueezeRate, squeezeRate,\
        d0, d1, d2, buyerPriceList, sellerPriceList

    nCycles=10000
    nBuyers= 50
    nSellers=100

    buyersSellersRatio=nBuyers/nSellers
    sellersBuyersRatio=nSellers/nBuyers
    usingRatios=True
    squeezeRate=0.3 # always < 1
    usingSqueezeRate=False

    seed=111
    np.random.seed(seed)

    d0=0.1
    d1=0.2
    d2=0.2
```

```
buyerPriceList=[]
sellerPriceList=[]

for i in range(nBuyers):
    buyerPriceList.append(1/(1+np.random.uniform(d0,d1)))
for j in range(nSellers):
    sellerPriceList.append(1+np.random.uniform(d0,d2))

plt.figure(0)
plt.plot(sorted(buyerPriceList,reverse=True),"r");
plt.plot(sorted(sellerPriceList),"b");
xlabel("the␣agents");
ylabel("agents'␣reservation␣prices");
```

3 The simplified Hayekian version

The buyers and the sellers meet randomly. Buyer i and seller j exchange if $pL_i^b \geq pL_j^s$; the deal is recorded at the price of the seller pL_j^s.[7]

In this version, representing the key point in this note, the running prices are multiplied in each cycle by the following correction coefficients:

- for the buyer: (i) $c_b = \frac{1}{1+u_b}$ if the deal succeeds (trying to pay less next time) or (ii) $c_b = 1 + u_b$ if the deal fails (preparing to pay more next time); in (i) and (ii) we have $u_b \sim \mathcal{U}(0, d_1)$
- for the seller: (iii) $c_s = 1 + u_s$ if the deal succeeds (trying to obtain a higher revenue next time) or (iv) $c_s = \frac{1}{1+u_s}$ if the deal fails (preparing to obtain a lower revenue next time); in (iii) and (iv) we have $u_s \sim \mathcal{U}(0, d_2)$.

With $seed = 111$, $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ set to $10,000$ we obtain sequences of mean prices (mean within each cycle) quite realistic, with a very low variance within each cycle (see Fig. 2 and 3).

[7] In the *mall*, sell prices are public.

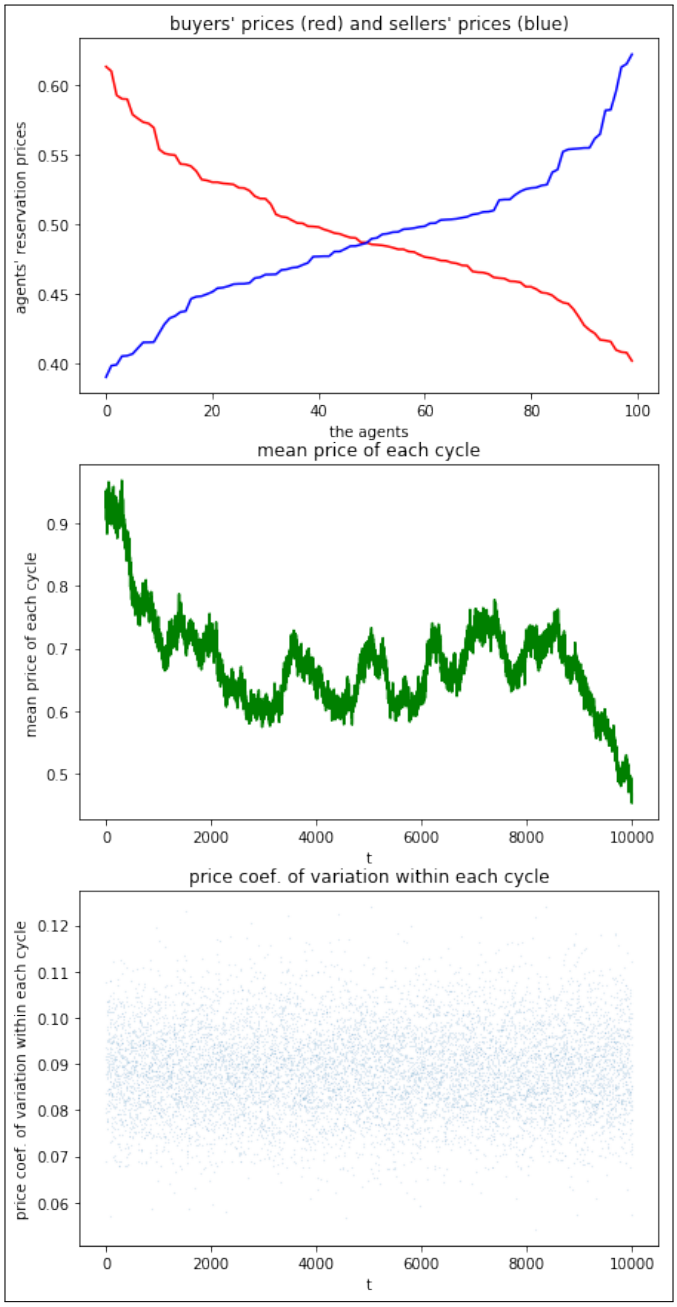

Figure 2: Simplified Hayekian case: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

The *coefficient of variation* at time t is calculated as:

$$\frac{standard\ deviation_t}{mean_t}$$

.

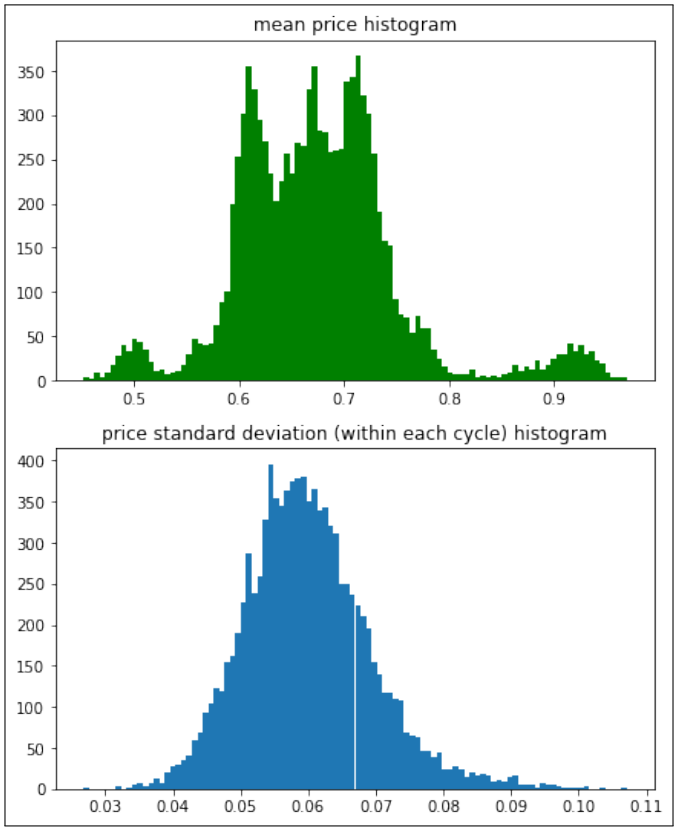

Figure 3: Simplified Hayekian case: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A comment: we have a plausible series of mean prices, with a complicated behavior, and with a high stability of the dispersion of the values within each cycle.
The right side of the buyer and seller curves shows another plausible situation: that of the presence of agents not exchanging. A note: this observation is related to a very important emergent effect for the *Oligopoly* model.

Have a look to the Appendix A for the cases of not balancing number of buyers and sellers. The IPython (or Python 3.x) code is:

Listing 3: The model in the simplified Hayekian perspective

```
# Hayekian perspective

warmingUp()
```

```
out = Output()
display(out)

meanPrice_ts=[]
meanPriceStDev_ts=[]
meanPriceVar_ts=[]

if usingRatios and not usingSqueezeRate:
    if buyersSellersRatio>1: d2*=sellersBuyersRatio
    if sellersBuyersRatio>1: d1*=buyersSellersRatio

if usingRatios and usingSqueezeRate:
    if buyersSellersRatio>1: d2*=sellersBuyersRatio*squeezeRate
    if sellersBuyersRatio>1: d1*=buyersSellersRatio*squeezeRate


for t in range(1,nCycles+1):
    dealPrices=[]
    agNum=max(nBuyers,nSellers)

    for n in range(agNum):

        i = np.random.randint(0,nBuyers)
        j = np.random.randint(0,nSellers)

        if buyerPriceList[i]>=sellerPriceList[j]:
            dealPrices.append(sellerPriceList[j])
            buyerPriceList[i] *=1/(1+np.random.uniform(0,d1))
            sellerPriceList[j]*=1+np.random.uniform(0,d2)
        else:
            buyerPriceList[i] *=1+np.random.uniform(0,d1)
            sellerPriceList[j]*=1/(1+np.random.uniform(0,d2))

    if len(dealPrices) > 2:
        meanPrice_ts.append(s.mean(dealPrices))
        meanPriceVar_ts.append(s.variance(dealPrices))
        meanPriceStDev_ts.append(s.stdev(dealPrices))
    else:
        meanPrice_ts.append(np.nan)
        meanPriceStDev_ts.append(np.nan)

    if t % 1000==0:
        with out:
            clear_output()
        with out:
            print('time', t, 'and␣n.␣of␣exchanges␣in␣the␣last␣cycle', \
              len(dealPrices))
            print(\
        'mean␣and␣var␣of␣exchange␣prices␣in␣the␣last␣cycle:␣%1.3e,␣%1.3e' %\
              (meanPrice_ts[-1],meanPriceVar_ts[-1]))

        plt.figure(1,figsize=(7,15),clear=True)

        plt.subplot(311)
        plt.plot(sorted(buyerPriceList,reverse=True),"r")
        plt.plot(sorted(sellerPriceList),"b")
        plt.title(\
            "buyers'␣prices␣(red)␣and␣sellers'␣prices␣(blue)")
        xlabel("the␣agents")
        ylabel("agents'␣reservation␣prices")

        plt.subplot(312)
```

```
        plt.title("mean price of each cycle")
        xlabel("t")
        ylabel("mean price of each cycle")
        plt.plot(meanPrice_ts,"g")

        plt.subplot(313)
        plt.title("price coef. of variation within each cycle")
        coefOfVariation=[]
        for m in range(len(meanPriceStDev_ts)):
            coefOfVariation.append(meanPriceStDev_ts[m]/
                                   meanPrice_ts[m])
        plt.plot(coefOfVariation,".",markersize=0.1)
        xlabel("t")
        ylabel("price coef. of variation within each cycle")
        #plt.show() #activate to see intermadiate plots
        #time.sleep(0.1)

# hist crashes with NaN
meanPrice_ts_hist=[]
for k in range(len(meanPrice_ts)):
    if not math.isnan(meanPrice_ts[k]):
        meanPrice_ts_hist.append(meanPrice_ts[k])
meanPriceStDev_ts_hist=[]
for k in range(len(meanPriceStDev_ts)):
    if not math.isnan(meanPriceStDev_ts[k]):
        meanPriceStDev_ts_hist.append(meanPriceStDev_ts[k])
plt.figure(2,figsize=(7,9))
plt.subplot(211)
if meanPrice_ts_hist != []:
    plt.title("mean price histogram")
    plt.hist(meanPrice_ts_hist,100,color="g");
plt.subplot(212)
if meanPriceStDev_ts_hist != []:
    plt.title("price standard deviation (within each cycle) histogram")
    plt.hist(meanPriceStDev_ts_hist,100);
```

4 The unstructured version

The buyers and the sellers meet randomly as in Section 3. Buyer i and seller j exchange in any case; the deal is recorded at the mean of the price of the seller pL_j^s and of the price pL_i^b of the buyer.

In this version the running prices are are multiplied in each cycle by the following correction coefficients:

- with the same probability for the buyer: (i) $c_b = \frac{1}{1+u_b}$ or (ii) $c_b = 1 + u_b$); in (i) and (ii) we have $u_b \sim \mathcal{U}(0, d_1)$
- with the same probability for the seller: (iii) $c_s = 1 + u_s$ or (iv) $c_s = \frac{1}{1+u_s}$; in (iii) and (iv) we have $u_s \sim \mathcal{U}(0, d_2)$.

With $seed = 111$, $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ set to $10,000$ we obtain exploding sequences of the means of the prices (mean in each cycle), and exploding standard deviation within each cycle (see Fig. 4 and 5).

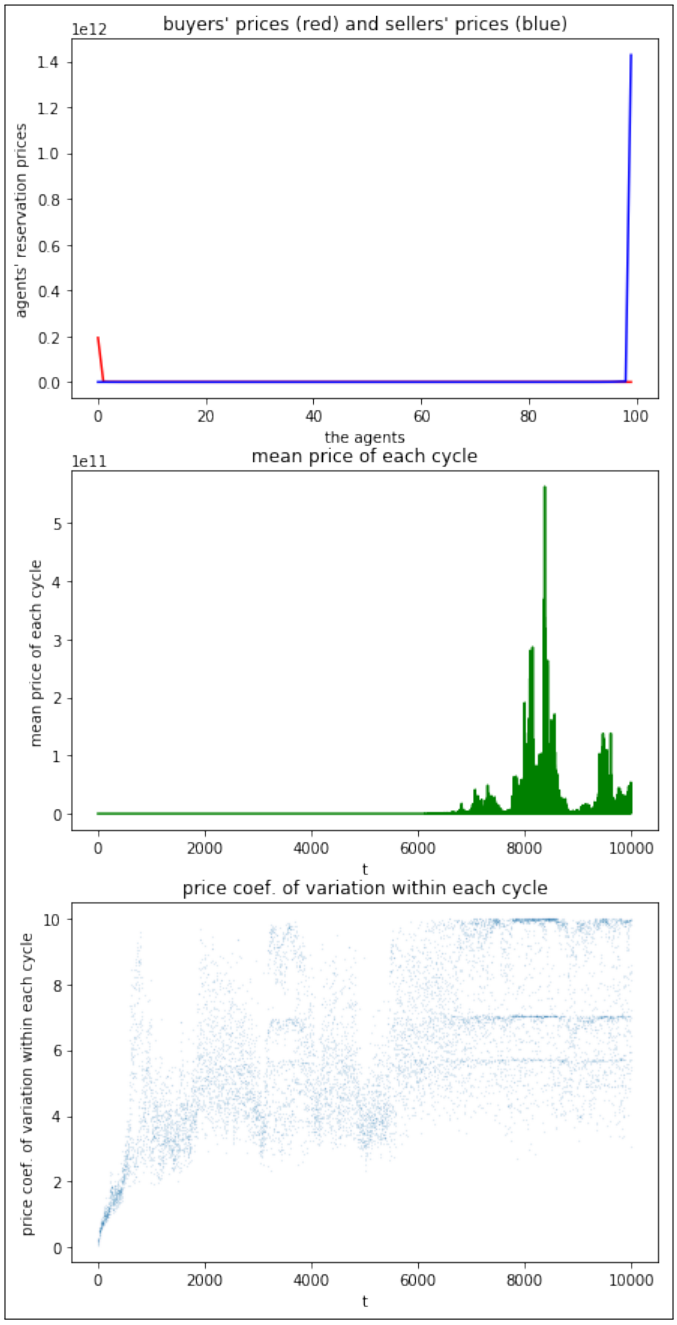

Figure 4: Unstructured case: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

The *coefficient of variation* at time t is calculated as:

$$\frac{standard\ deviation_t}{mean_t}$$

.

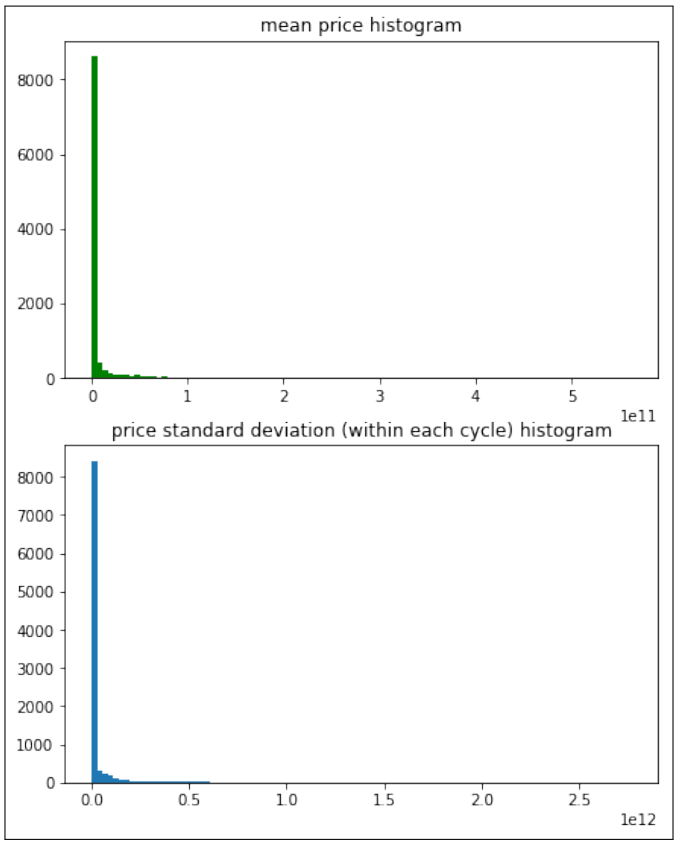

Figure 5: Unstructured case: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A comment: this counter-example shows that, missing the *intelligence* in the correction of the prices (implicitly propagating among all the agents), a system of pure random price settings is absolutely far from being plausible.

The IPython (or Python 3.x) code is:

Listing 4: The unstructured version

```
# unstructured case

warmingUp()

out = Output()
display(out)
```

```
meanPrice_ts=[]
meanPriceStDev_ts=[]
meanPriceVar_ts=[]

if usingRatios and not usingSqueezeRate:
    if buyersSellersRatio>1: d2*=sellersBuyersRatio
    if sellersBuyersRatio>1: d1*=buyersSellersRatio

if usingRatios and usingSqueezeRate:
    if buyersSellersRatio>1: d2*=sellersBuyersRatio*squeezeRate
    if sellersBuyersRatio>1: d1*=buyersSellersRatio*squeezeRate

for t in range(1,nCycles+1):
    dealPrices=[]
    agNum=max(nBuyers,nSellers)
    for n in range(agNum):
        i = np.random.randint(0,nBuyers)
        j = np.random.randint(0,nSellers)

        dealPrices.append((sellerPriceList[j]+buyerPriceList[i]/0.5))

        if np.random.uniform(0,1)>=0.5:
            buyerPriceList[i] *=1/(1+np.random.uniform(0,d1))
            sellerPriceList[j]*=1+np.random.uniform(0,d2)
        else:
            buyerPriceList[i] *=1+np.random.uniform(0,d1)
            sellerPriceList[j]*=1/(1+np.random.uniform(0,d2))


    if len(dealPrices) > 2:
        meanPrice_ts.append(s.mean(dealPrices))
        meanPriceVar_ts.append(s.variance(dealPrices))
        meanPriceStDev_ts.append(s.stdev(dealPrices))
    else:
        meanPrice_ts.append(np.nan)
        meanPriceStDev_ts.append(np.nan)

    if t % 1000==0:
        with out:
            clear_output()
        with out:
            print('time', t, 'and␣n.␣of␣exchanges␣in␣the␣last␣cycle', \
              len(dealPrices))
            print(\
        'mean␣and␣var␣of␣exchange␣prices␣in␣the␣last␣cycle:␣%1.3e,␣%1.3e' %\
              (meanPrice_ts[-1],meanPriceVar_ts[-1]))

        plt.figure(3,figsize=(7,15),clear=True)

        plt.subplot(311)
        plt.plot(sorted(buyerPriceList,reverse=True),"r")
        plt.plot(sorted(sellerPriceList),"b")
        plt.title(\
            "buyers'␣prices␣(red)␣and␣sellers'␣prices␣(blue)")
        xlabel("the␣agents")
        ylabel("agents'␣reservation␣prices")

        plt.subplot(312)
        plt.title("mean␣price␣of␣each␣cycle")
        xlabel("t")
        ylabel("mean␣price␣of␣each␣cycle")
        plt.plot(meanPrice_ts,"g")
```

```
        plt.subplot(313)
        plt.title("price␣coef.␣of␣variation␣within␣each␣cycle")
        coefOfVariation=[]
        for m in range(len(meanPriceStDev_ts)):
            coefOfVariation.append(meanPriceStDev_ts[m]/
                                   meanPrice_ts[m])
        plt.plot(coefOfVariation,".",markersize=0.1)
        xlabel("t")
        ylabel("price␣coef.␣of␣variation␣within␣each␣cycle")
        #plt.show() #activate to see intermadiate plots
        #time.sleep(0.1)

# hist crashes with NaN
meanPrice_ts_hist=[]
for k in range(len(meanPrice_ts)):
    if not math.isnan(meanPrice_ts[k]):
        meanPrice_ts_hist.append(meanPrice_ts[k])
meanPriceStDev_ts_hist=[]
for k in range(len(meanPriceStDev_ts)):
    if not math.isnan(meanPriceStDev_ts[k]):
        meanPriceStDev_ts_hist.append(meanPriceStDev_ts[k])
plt.figure(4,figsize=(7,9))
plt.subplot(211)
if meanPrice_ts_hist != []:
    plt.title("mean␣price␣histogram")
    plt.hist(meanPrice_ts_hist,100,color="g");
plt.subplot(212)
if meanPriceStDev_ts_hist != []:
    plt.title("price␣standard␣deviation␣(within␣each␣cycle)␣histogram")
    plt.hist(meanPriceStDev_ts_hist,100);
```

Appendices

Appendix A

Two triple cases of not balancing numbers of buyers and sellers

A.1 Case $nBuyers \gg nSellers$

With $nBuyers \gg nSellers$ (e.g., $nBuyers = 100$ and $nSellers = 50$, as in Fig. A.1), we have three possible paths of analysis.

A.1.a Case $nBuyers \gg nSellers$, with different rates of per-capita correction

If $nBuyers \gg nSellers$, we have in each cycle one call—in mean—to a *seller* from each *buyer*, the number of per-capita actions of the *sellers* in each cycle is greater of the number of per-capita actions of the *buyers*.

As a consequence, the probability that a *seller* decreases her price to meet that of a *b*uyer is greater than the probability that a *buyer* increases her price to meet that of a *seller*.

We can observe that in Figs. A.2 and A.3 the prices are—in the end—lower than in Figs. 2 and 3 and, must of all, the price tendency has a strong negative slope. We always have $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, and $seed = 111$.

This result is inconsistent with the microeconomic theory, where we could expect that an excess of demand will generate the rise of the prices.

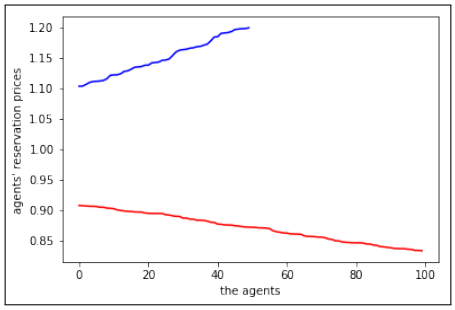

Figure A.1: An example of initial not overlapping demand curve and offer curve, case $nBuyers \gg nSellers$

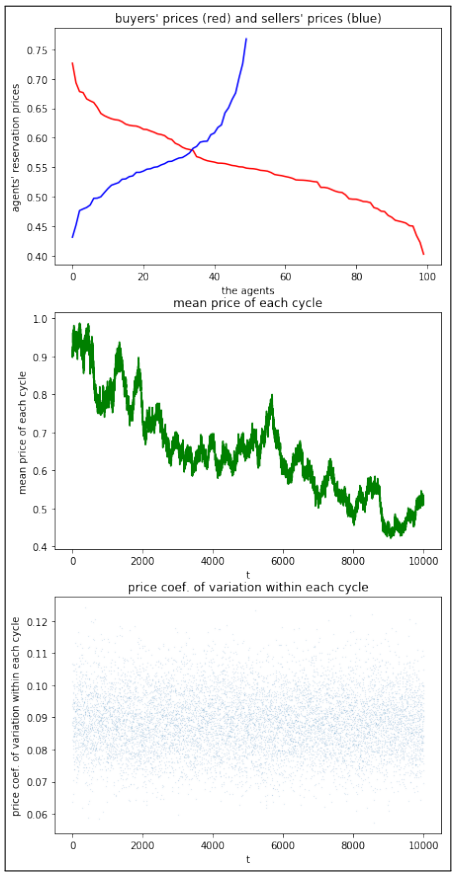

Figure A.2: Simplified Hayekian case, with $nBuyers \gg nSellers$: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

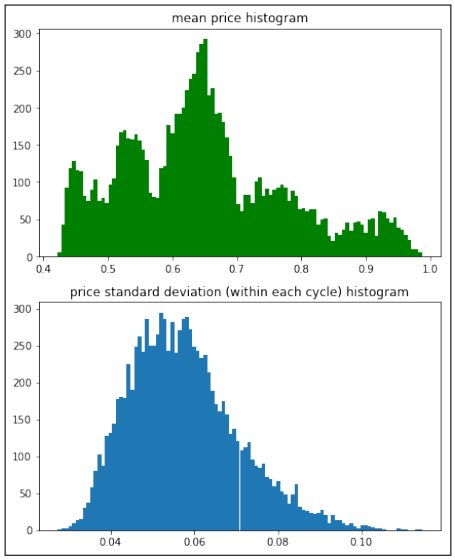

Figure A.3: Simplified Hayekian case, with $nBuyers \gg nSellers$: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A.1.b Case $nBuyers \gg nSellers$, with unequal rates of per-capita correction, with equivalent effects

Again, with $nBuyers \gg nSellers$, and always having in each cycle one call—in mean—to a *seller* from each *buyer*, the number of per-capita actions of the *sellers* in each cycle is greater of the number of per-capita actions of the *buyers*.

In this second version of the case $nBuyers \gg nSellers$, we always have $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, and $seed = 111$.

The novelty is that of setting $usingRatios = True$, so we are activating limitations to d_1 or d_2.

The limitations work as follow:

- if the $\frac{nBuyers}{nSellers} > 1$ (our case in this example), d_2, i.e. the upper limit of the rate of correction of the price of the sellers, is multiplied by $\frac{nSellers}{nBuyers}$;[8]
- if the $\frac{nSellers}{nBuyers} > 1$, d_1, i.e. the upper limit of the rate of correction of the price of the buyers, is multiplied by $\frac{nBuyers}{nSellers}$.

We have now unequal rates of per-capita correction, with equivalent effects. The interpretation is that if the number of sellers is smaller than the number of buyers, the sellers act with a slow pace of price correction (proportional to $\frac{nSellers}{nBuyers}$) because in this way they can cherry-pick the best buyers (those with the higher reservation price). In this way, they avoid to contribute to the fall of the prices.

[8] An example to clarify: in this Section we have $nBuyers = 100$ and $nSellers = 50$, so $\frac{nBuyers}{nSellers} \equiv 2$ and $\frac{nSellers}{nBuyers} \equiv 0.5$; d_2 is reduced of the 50%.

Always with Fig. A.1 as the starting configuration of the prices, in Fig.s A.4 and A.5 we see now interesting price oscillations roughly confined between the limits of Fig. A.1.

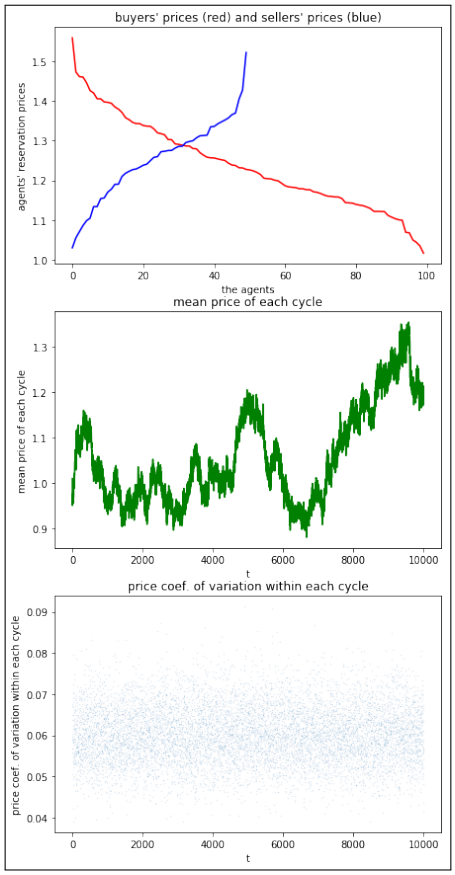

Figure A.4: Simplified Hayekian case, with $nBuyers \gg nSellers$ but with equivalent effects: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

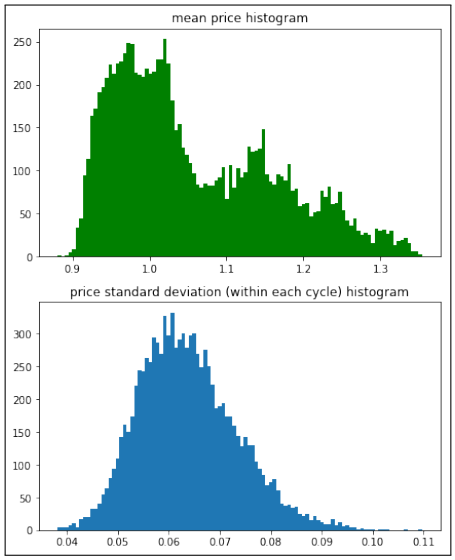

Figure A.5: Simplified Hayekian case, with $nBuyers \gg nSellers$ but with equivalent effects: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A.1.c Case $nBuyers \gg nSellers$, with unequal rates of per-capita correction, but squeezing the effects

In this third version of the case $nBuyers \gg nSellers$, we always have $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, and $seed = 111$.

The second novelty, after that of Section A.1.b, is that of setting $usingSqueeze = True$ (and setting $usingRatios = True$ as in Section A.1.b), so we are activating further limitations to d_1 or d_2. We also have $squeezeRate = 0.3$.

- if the $\frac{nBuyers}{nSellers} > 1$ (our case in this example), d_2, i.e. the upper limit of the rate of correction of the price of the sellers, is multiplied by $squeezeRate$;
- if the $\frac{nSellers}{nBuyers} > 1$, d_1, i.e. the upper limit of the rate of correction of the price of the buyers, is multiplied by $squeezeRate$;

Always with Fig. A.1 as the starting configuration of the prices, in Fig.s A.6 and A.7 we see a limited price dynamics, very close to the top band of Fig. A.1. This result is perfectly consistent with microeconomic theory.

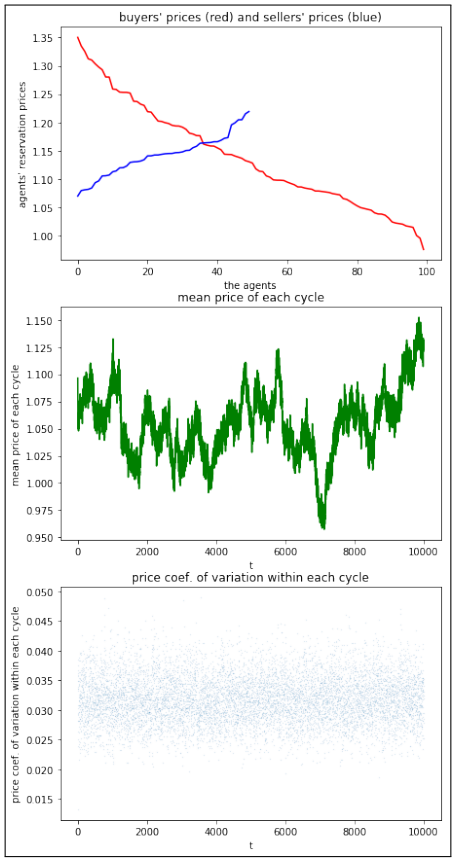

Figure A.6: Simplified Hayekian case, with $nBuyers \gg nSellers$ but squeezing the effects: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

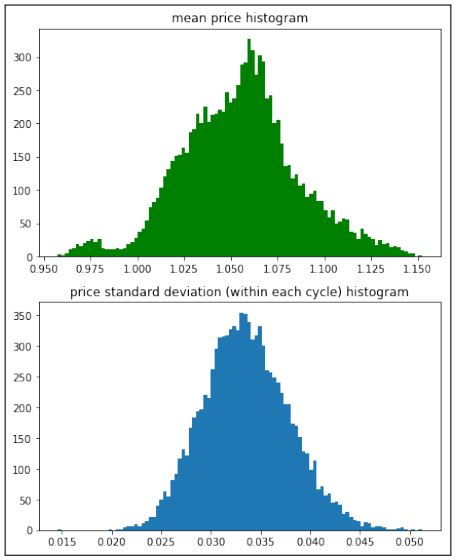

Figure A.7: Simplified Hayekian case, with $nBuyers \gg nSellers$ but squeezing the effects: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A.2 Case $nBuyers \ll nSellers$

With $nBuyers \ll nSellers$ (e.g., $nBuyers = 50$ and $nSellers = 100$, as in Fig. A.8), we again have three possible paths of analysis.

A.2.a Case $nBuyers \ll nSellers$, with different rates of per-capita correction

If $nBuyers \ll nSellers$, we have in each cycle one call—in mean—to a *buyer* from each *seller*, the number of per-capita actions of the *buyers* in each cycle is greater of the number of per-capita actions of the *sellers*.

As a consequence, the probability that a *buyer* increases her price to meet that of a *seller* is greater than the probability that a *seller* decreases her price to meet that of a *buyer*.

We can observe that in Figs. A.9 and A.10 the prices are—in the end—greater than in Figs. 2 and 3 and, must of all, the price tendency has a strong positive slope. We always have $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, and $seed = 111$.

This result is inconsistent with the microeconomic theory, where we could expect that an excess of offer will generate the fall of the prices.

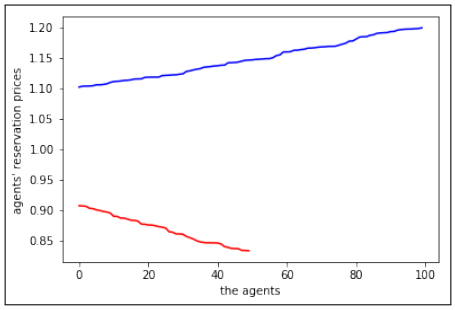

Figure A.8: An example of initial not overlapping demand curve and offer curve, case $nBuyers \ll nSellers$

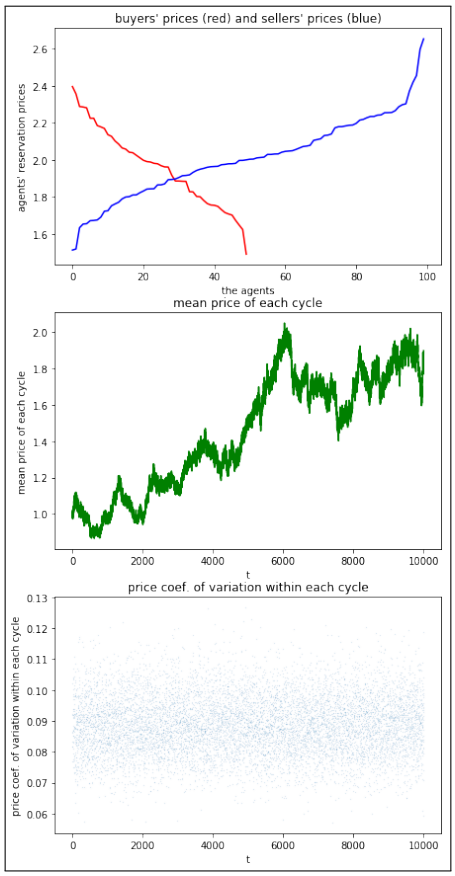

Figure A.9: Simplified Hayekian case, with $nBuyers \ll nSellers$: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

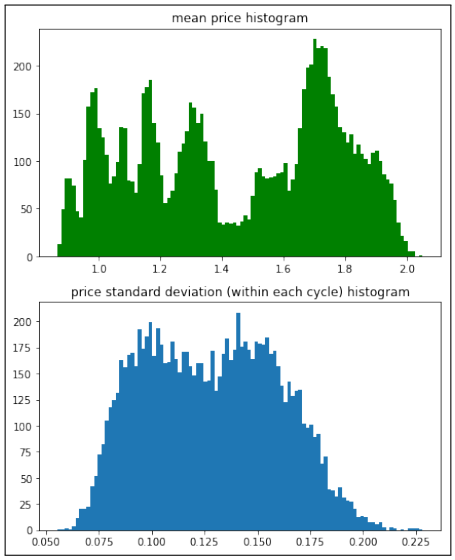

Figure A.10: Simplified Hayekian case, with $nBuyers \ll nSellers$: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A.2.b Case $nBuyers \ll nSellers$, with unequal rates of per-capita correction, with equivalent effects

In this second version of the case $nBuyers \ll nSellers$, we always have $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, and $seed = 111$.

As in Section A.1.b, the novelty is that of setting $usingRatios = True$, so we are activating limitations to d_1 or d_2.

The limitations work as follow:

- if the $\frac{nBuyers}{nSellers} > 1$, d_2, i.e. the upper limit of the rate of correction of the price of the sellers, is multiplied by $\frac{nSellers}{nBuyers}$;
- if the $\frac{nSellers}{nBuyers} > 1$ (our case in this example), d_1, i.e. the upper limit of the rate of correction of the price of the buyers, is multiplied by $\frac{nBuyers}{nSellers}$.[9]

We have now unequal rates of per-capita correction, with equivalent effects. The interpretation is that if the number of buyers is smaller than the number of sellers, the buyers act with a slow pace of price correction (proportional to $\frac{nBuyers}{nSellers}$) because in this way they can cherry-pick the best sellers (those with the lower reservation price). In this way, they avoid to contribute to the rise of the prices.

Always with Fig. A.8 as the starting configuration of the prices, in Fig.s A.11 and A.12 we see now a compressed price oscillations roughly close to the bottom limits of Fig. A.8.

[9] An example to clarify: in this Section we have $nBuyers = 50$ and $nSellers = 100$, so $\frac{nSellers}{nBuyers} \equiv 2$ and $\frac{nBuyers}{nSellers} \equiv 0.5$; d_1 is reduced of the 50%.

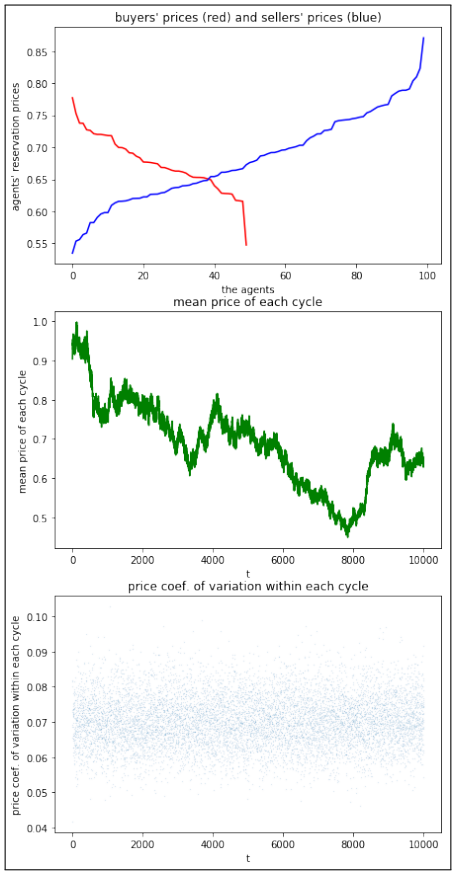

Figure A.11: Simplified Hayekian case, with $nBuyers \gg nSellers$ with equivalent effects: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

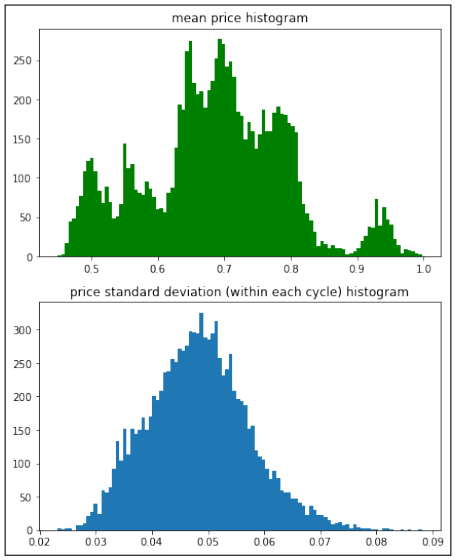

Figure A.12: Simplified Hayekian case, with $nBuyers \gg nSellers$ but with equivalent effects: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

A.2.c Case $nBuyers \ll nSellers$, with unequal rates of per-capita correction, but squeezing the effects

In this third version of the case $nBuyers \ll nSellers$, we always have $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$, and $seed = 111$.

The second novelty, after that of Section A.2.b, is that of setting $usingSqueeze = True$ (and setting $usingRatios = True$ as in Section A.2.b), so we are activating further limitations to d_1 or d_2. We also have $squeezeRate = 0.3$.

- if the $\frac{nBuyers}{nSellers} > 1$, d_2, i.e. the upper limit of the rate of correction of the price of the sellers, is multiplied by $squeezeRate$;
- if the $\frac{nSellers}{nBuyers} > 1$ (our case in this example), d_1, i.e. the upper limit of the rate of correction of the price of the buyers, is multiplied by $squeezeRate$;

Always with Fig. A.8 as the starting configuration of the prices, in Fig.s A.13 and A.14 we see a limited price dynamics, very close to the bottom band of Fig. A.8. This result is perfectly consistent with microeconomic theory.

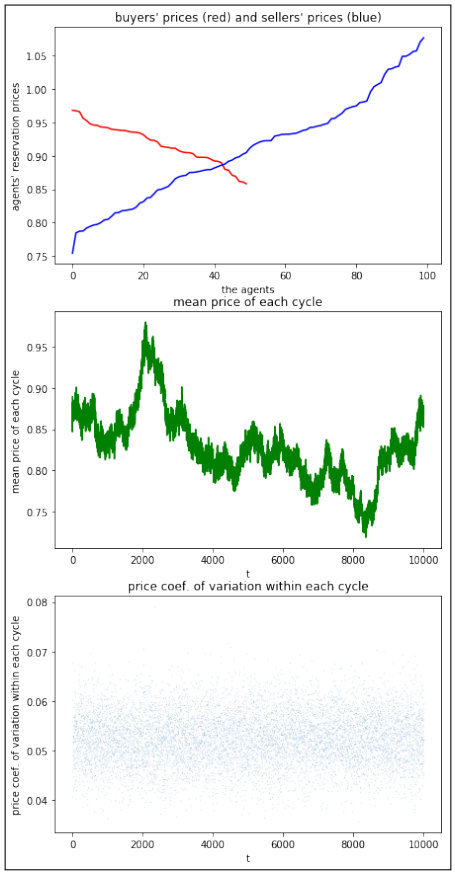

Figure A.13: Simplified Hayekian case, with $nBuyers \gg nSellers$ but squeezing the effects: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

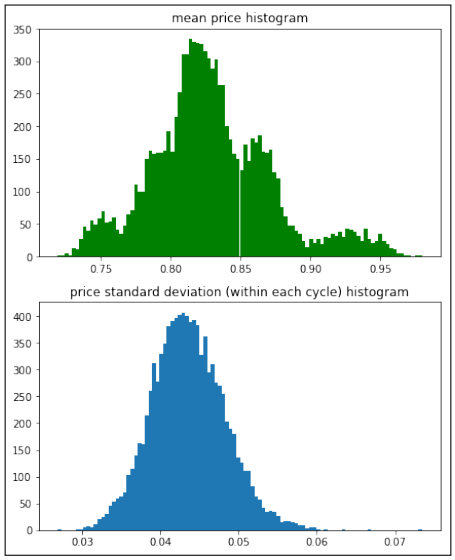

Figure A.14: Simplified Hayekian case, with $nBuyers \gg nSellers$ but squeezing the effects: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

Appendix B

Trying to activate the *idle* agents, with a fundamental unexpected by-product

B.1 Corrupting the simplified Hayekian market model to activate the idle agents: an ingenuous quest for the solution

In all the runs of the model up to here, looking at the demand and offer curves we can see that the larger part of the agents is not acting: specifically, all the sellers with reservation price greater than that of the crossing point of the curves and all the buyers with reservation price lower than that value.

Trying to activate those idle agents, we add two new parameters to the list of p. 4:

- *buyerThreshold* - over this number of failures, a special price correction occurs, multiplying the buyer price by $c_b = 1 + u_b$ with $u_b \sim \mathcal{U}(0, d_{1_{overT}})$;
- *sellerThreshold* - over this number of failures, a special price correction occurs, multiplying the seller price by $c_s = \frac{1}{1+u_s}$ with $u_s \sim \mathcal{U}(0, d_{2_{overT}})$.

With $seed = 111$, $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ set to $10,000$, we add $buyerThreshold = 5$, $sellerThreshold = 50$, $d_{1_{overT}} = 0.4$, and $d_{2_{overT}} = 0.4$. We also have $nBuyers = nSellers = 100$, so none of the corrections of Appendix A are working here.

As you can see[10] in Listing B.1, Listing 2 of p. 5 is modified in the following way: now each agent holds a two element vector reporting: (i) her reservation price and (ii) a counter.

Listing B.1: Warming up for the corrupted version of the model

```
# warming up for corrupted version

# execute before the corrupted Hayekian version
def warmingUp():
    global nCycles, nBuyers, nSellers,\
```

[10] At https://github.com/terna/microHayekianMarket the notebook is now https://github.com/terna/microHayekianMarket/blob/master/microCorruptedHayekianMarket_base.ipynb.

```
    buyersSellersRatio, sellersBuyersRatio,\
    usingRatios, usingSqueezeRate, squeezeRate,\
    d0, d1, d2, buyerPriceList, sellerPriceList,\
    sellerThreshold, buyerThreshold,\
    d1overT, d2overT, fig0

nCycles=10000
nBuyers= 100
nSellers=100

buyersSellersRatio=nBuyers/nSellers
sellersBuyersRatio=nSellers/nBuyers
usingRatios=True
squeezeRate=0.3 # always < 1
usingSqueezeRate=False

seed=111
np.random.seed(seed)

d0=0.1
d1=0.2
d2=0.2

d1overT=0.4
d2overT=0.4

sellerThreshold=50
buyerThreshold =5

buyerPriceList=[]
sellerPriceList=[]

for i in range(nBuyers):
    buyerPriceList.append([1/(1+np.random.uniform(d0,d1)),0])
for j in range(nSellers):
    sellerPriceList.append([1+np.random.uniform(d0,d2),0])

#zip is an iterator and * means iteratable
fig0=plt.figure(0)
plt.plot(list(zip(*sorted(buyerPriceList,reverse=True)))[0],"r");
plt.plot(list(zip(*sorted(sellerPriceList)))[0],"b");
xlabel("the␣agents");
ylabel("agents'␣reservation␣prices");
```

Also Listing 3 of p. 8 is modified, to build Listing B.2, where the counter of each agent records the number of consecutive failures in buying or in selling.

In case of success, the counter is set again to 0.

If the counter is *greater than* one of the thresholds, $buyerThreshold$ or $sellerThreshold$, the buyer or the seller make a special correction of their reservation prices, multiplying the buyer price by $c_b = 1+u_b$ (with $u_b \sim \mathcal{U}(0, d_{1_{overT}})$) or the seller price by $c_s = \frac{1}{1+u_s}$ (with $u_s \sim \mathcal{U}(0, d_{2_{overT}})$).

The idea behind this construction is that of forcing the *idle* agents to operate, amplifying their price corrections. The effect is quite disappointing.

Considering that as a starting point we always have the situation of Fig. 1 of p. 5, in Figs B.1 and B.2 we simply see that in our case, with $buyerThreshold = 5$ and $sellerThreshold = 50$, the price series is quite higher than in Fig. 2 and that a lot of buyers (and sellers too, but here we focus on the buyers) still are not operating. Our attempt of *corrupting* the simplified Hayekian market to force the buyers to buy, has been a complete failure. But ... the next paragraph at 34 shows that it has been a source of inspiration.

Listing B.2: The model in the corrupted simplified Hayekian perspective

```
# corrupted simplified Hayekian perspective

warmingUp()

out = Output()
display(out)

meanPrice_ts=[]
meanPriceStDev_ts=[]
meanPriceVar_ts=[]

if usingRatios and not usingSqueezeRate:
    if buyersSellersRatio>1: d2*=sellersBuyersRatio
    if sellersBuyersRatio>1: d1*=buyersSellersRatio

if usingRatios and usingSqueezeRate:
    if buyersSellersRatio>1: d2*=sellersBuyersRatio*squeezeRate
    if sellersBuyersRatio>1: d1*=buyersSellersRatio*squeezeRate


for t in range(1,nCycles+1):
    dealPrices=[]
    agNum=max(nBuyers,nSellers)
    for n in range(agNum):

        i = np.random.randint(0,nBuyers)
        j = np.random.randint(0,nSellers)

        if buyerPriceList[i][0]>=sellerPriceList[j][0]:
            dealPrices.append(sellerPriceList[j][0])
            buyerPriceList[i][0] *=1/(1+np.random.uniform(0,d1))
            buyerPriceList[i][1] =0 # success
            sellerPriceList[j][0]*=1+np.random.uniform(0,d2)
            sellerPriceList[j][1] =0 # success
        else:
            buyerPriceList[i][0] *=1+np.random.uniform(0,d1)
            buyerPriceList[i][1] +=1 # failure
            sellerPriceList[j][0]*=1/(1+np.random.uniform(0,d2))
            sellerPriceList[j][1]+=1 # failure

        # corrections

        if buyerPriceList[i][1]  > buyerThreshold:
            buyerPriceList[i][0] *=1+np.random.uniform(0,d1overT)
            buyerPriceList[i][1] =0
        if sellerPriceList[j][1] > sellerThreshold:
            sellerPriceList[j][0]*=1/(1+np.random.uniform(0,d2overT))
            sellerPriceList[j][1]=0


    if len(dealPrices) > 2:
        meanPrice_ts.append(s.mean(dealPrices))
        meanPriceVar_ts.append(s.variance(dealPrices))
        meanPriceStDev_ts.append(s.stdev(dealPrices))
    else:
        meanPrice_ts.append(np.nan)
        meanPriceStDev_ts.append(np.nan)

    if t % 1000==0:
        with out:
            clear_output()
```

```
        with out:
            print('time', t, 'and␣n.␣of␣exchanges␣in␣the␣last␣cycle', \
              len(dealPrices))
            print(\
        'mean␣and␣var␣of␣exchange␣prices␣in␣the␣last␣cycle:␣%1.3e,␣%1.3e' %\
              (meanPrice_ts[-1],meanPriceVar_ts[-1]))

        fig1=plt.figure(1,figsize=(7,15),clear=True)

        plt.subplot(311)
        plt.plot(list(zip(*sorted(buyerPriceList, reverse=True)))[0],"r")
        plt.plot(list(zip(*sorted(sellerPriceList)))[0],"b")
        plt.title(\
            "buyers'␣prices␣(red)␣and␣sellers'␣prices␣(blue)")
        xlabel("the␣agents")
        ylabel("agents'␣reservation␣prices")

        plt.subplot(312)
        plt.title("mean␣price␣of␣each␣cycle")
        xlabel("t")
        ylabel("mean␣price␣of␣each␣cycle")
        plt.plot(meanPrice_ts,"g")

        plt.subplot(313)
        plt.title("price␣coef.␣of␣variation␣within␣each␣cycle")
        coefOfVariation=[]
        for m in range(len(meanPriceStDev_ts)):
            coefOfVariation.append(meanPriceStDev_ts[m]/
                                   meanPrice_ts[m])
        plt.plot(coefOfVariation,".",markersize=0.1)
        xlabel("t")
        ylabel("price␣coef.␣of␣variation␣within␣each␣cycle")
        #plt.show() #activate to see intermadiate plots
        #time.sleep(1)

# hist crashes with NaN
meanPrice_ts_hist=[]
for k in range(len(meanPrice_ts)):
    if not math.isnan(meanPrice_ts[k]):
        meanPrice_ts_hist.append(meanPrice_ts[k])
meanPriceStDev_ts_hist=[]
for k in range(len(meanPriceStDev_ts)):
    if not math.isnan(meanPriceStDev_ts[k]):
        meanPriceStDev_ts_hist.append(meanPriceStDev_ts[k])
fig2=plt.figure(2,figsize=(7,9))
plt.subplot(211)
if meanPrice_ts_hist != []:
    plt.title("mean␣price␣histogram")
    plt.hist(meanPrice_ts_hist,100,color="g");
plt.subplot(212)
if meanPriceStDev_ts_hist != []:
    plt.title("price␣standard␣deviation␣(within␣each␣cycle)␣histogram")
    plt.hist(meanPriceStDev_ts_hist,100);
```

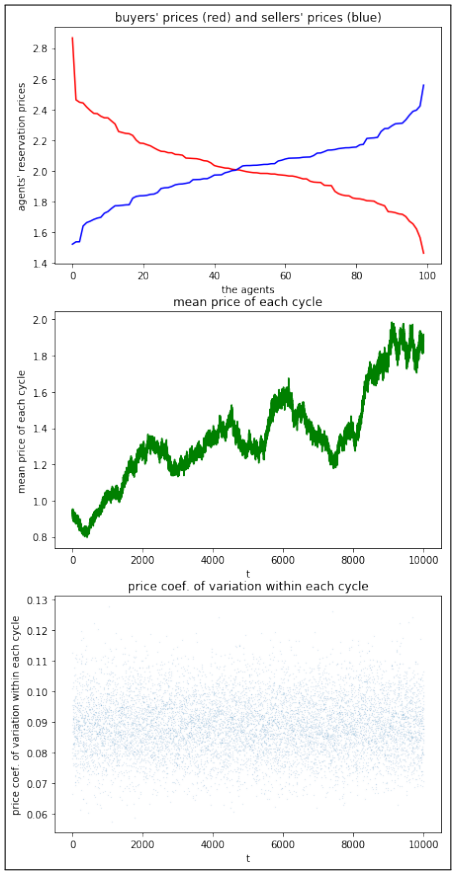

Figure B.1: Corrupting the Simplified Hayekian case: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

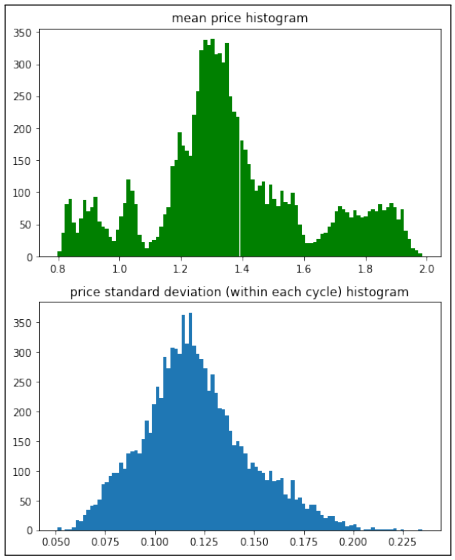

Figure B.2: Corrupting the Simplified Hayekian case: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

Observing now the Fig. B.3, that we generate with the code of the Listing B.3, we discover that the agent not operating are distributed in the whole range of the possible prices of Figs. B.1 and B.2. Consequence: the idle agents continuously change and, in a given instant of the time, the idle one are not necessarily the same of the previous tick.

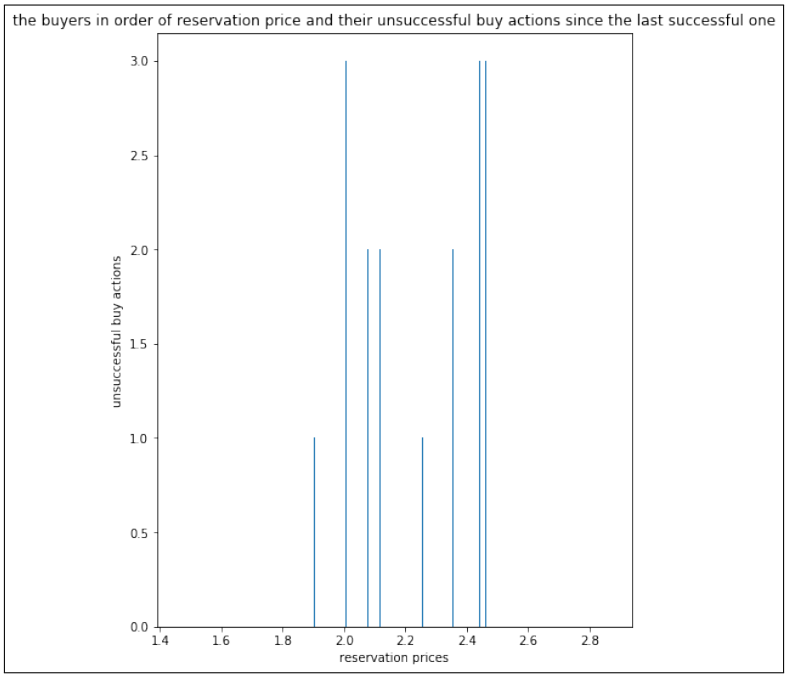

Figure B.3: Extending the Simplified Hayekian case: after 10,000 actions we see here a *flash* image of the buyers ordered following their reservation prices, with the number of recent consecutive unsuccessful buying attempts

Listing B.3: The code to show agents' successful actions

```
# the buyers in order of reservation price
# and their successful buy acctions

fig3=plt.figure(3,figsize=(7,9))
plt.title("the␣buyers␣in␣order␣of␣reservation␣price␣"+\
          "and␣their␣unsuccessful␣buy␣actions␣since␣the␣"+\
          "last␣successful␣one");
plt.bar(list(zip(*sorted(buyerPriceList)))[0],\
        list(zip(*sorted(buyerPriceList)))[1],0.001)
xlabel("reservation␣prices");
ylabel("unsuccessful␣buy␣actions");
```

B.2 A fundamental unexpected by-product: the solution is here, under our eyes

Problem fixed! The solution of the quest for the idle agents' enigma was under our eyes. Due to the continuous price corrections, the buy (or sell, but here we focus on the buyers) action are quite uncertain and in the 50% of the cases they fail. Why 50%? Have a look the the demand and offer curves on the right of their crossing point.

Let us generate a new case, as a by-product of the disappointing model run of Section B.1.

With $seed = 111$, $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ now set to $20,000$, we also have $nBuyers = nSellers = 100$; so, none of the corrections of Appendix A are working here. Note that $buyerThreshold$, $sellerThreshold$, $d1_{overT}$, and $d2_{overT}$ are not in use here.

The code of Listing B.1 is also working here.[11]

In Listing B.4 we report the changes to Listing **??**: the counters are increased of one unit for each successful case and never set again to 0. The corrections dedicated to amplify the price movements in Section **??**, are now commented with the triple hyphen signs, so not working.

Listing B.4: Changes to the listing of model in the corrupted simplified hayekian perspective

```
if buyerPriceList[i][0]>=sellerPriceList[j][0]:
    dealPrices.append(sellerPriceList[j][0])
    buyerPriceList[i][0] *=1/(1+np.random.uniform(0,d1))
    buyerPriceList[i][1] +=1 #=0 # success
    sellerPriceList[j][0]*=1+np.random.uniform(0,d2)
    sellerPriceList[j][1] +=1 #=0 # success
else:
    buyerPriceList[i][0] *=1+np.random.uniform(0,d1)
    #buyerPriceList[i][1] +=1 # failure
    sellerPriceList[j][0]*=1/(1+np.random.uniform(0,d2))
    #sellerPriceList[j][1]+=1 # failure

# corrections
"""
if buyerPriceList[i][1]  > buyerThreshold:
    buyerPriceList[i][0] *=1+np.random.uniform(0,d1overT)
    buyerPriceList[i][1] =0
if sellerPriceList[j][1] > sellerThreshold:
    sellerPriceList[j][0]*=1/(1+np.random.uniform(0,d2overT))
    sellerPriceList[j][1]=0
"""
```

Effects (having as starting point always the situation of Fig. 1 of p. 5: in Figs B.4 we have in the first half (10,000 cycles) exactly the same result of Fig. 2 of p. 2 (same *seed* and same scheme) and in the second half (until the final value of 20,000 cycles) a consistent continuation of the same dynamic. Fig. B.5, with its bimodal price distribution and the low values of the standard deviations within each tick, confirms what we saw in Section **??**.

Must of all, the bar plot in Fig. B.6—generated with the code of Listing B.5—shows that all the agents are effectively very close to succeed in buying in the 50% of the possible cases. This result is consistent with the plot of the demand and offer curves, with about one half of the agents on the right of the crossing point, remembering that those agents are never the same.

[11] At https://github.com/terna/microHayekianMarket the notebook is now https://github.com/terna/microHayekianMarket/blob/master/microCorruptedHayekianMarket_by-product.ipynb.

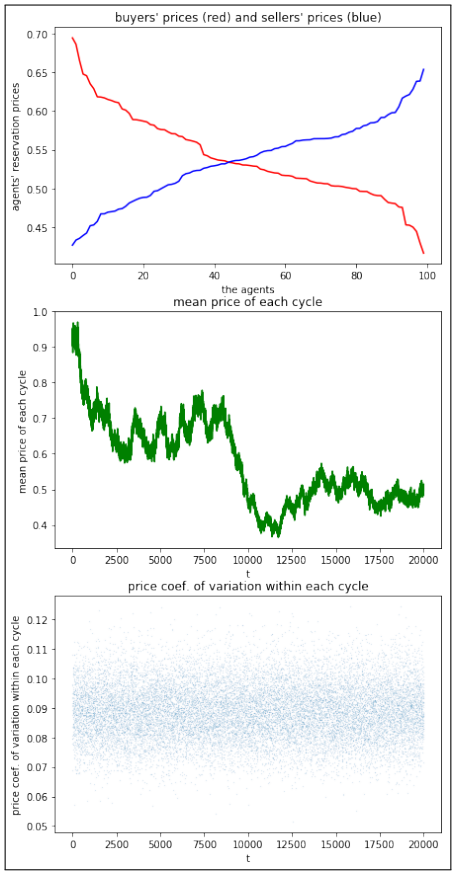

Figure B.4: Extending the Simplified Hayekian case: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

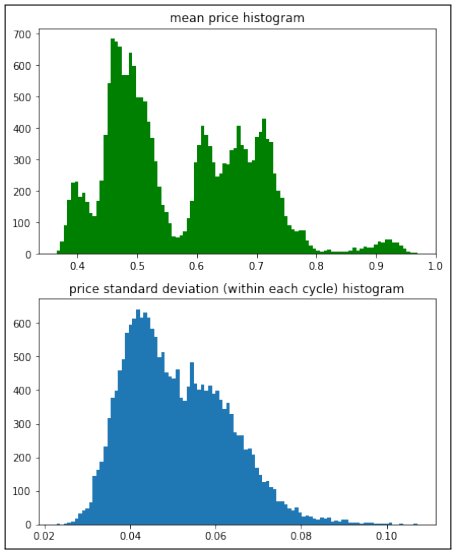

Figure B.5: Extending the Simplified Hayekian case: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

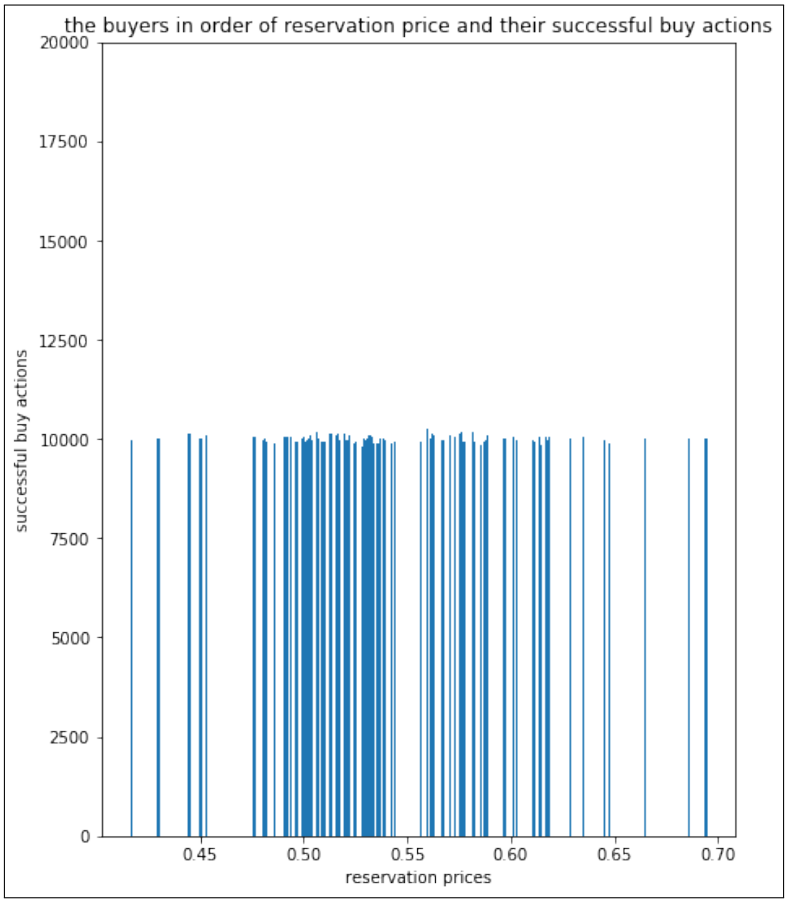

Figure B.6: Extending the Simplified Hayekian case: after 20,000 actions, each buyer (here the buyers are ordered following their reservation prices) has been successful in concluding a deal only in about 10,000 cases

Listing B.5: The code to show agents' successful actions in the by-product case

```
# the buyers in order of reservation price
# and their successful buy acctions

fig3=plt.figure(3,figsize=(7,9))
plt.title("the␣buyers␣in␣order␣of␣reservation␣price␣"+\
          "and␣their␣successful␣buy␣actions");
plt.bar(list(zip(*sorted(buyerPriceList)))[0],\
         list(zip(*sorted(buyerPriceList)))[1],0.001)
xlabel("reservation␣prices");
ylabel("successful␣buy␣actions");
plt.ylim(0, nCycles);
```

B.3 Final considerations related to the Oligopoly project

Here we have the key point for the development of the Oligopoly[12] project.

I To control the level of the prices: we can graduate the intervention described in Section A.1.c, with the case $nBuyers \gg nSellers$, which is intrinsically related to unequal rates of per-capita correction, squeezing—and graduating—the effects beyond the level of equivalence.

The economic goal is that of controlling the level of the profits of entrepreneurs, raising the prices.

II To be sure that the consumption level matches the production: we can increase the number of buying attempts—with price correction—in each cycle, as shown in Section **??** of Appendix B.

The economic goal is that of avoiding the systematic tendency toward the recession that we have found until now (end of July, 2018) in the hayekian version of the Oligopoly work.

[12] https://terna.github.io/oligopoly/

Bibliography

Bowles, S., Kirman, A. and Sethi, R. (2017). *Retrospectives: Friedrich Hayek and the Market Algorithm*. In «Journal of Economic Perspectives», vol. 31(3), pp. 215-30.
URL http://www.aeaweb.org/articles?id=10.1257/jep.31.3.215

Index