micro Hayekian Market

Matteo Morini[1] and Pietro Terna[2]

August 1, 2018

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

Abstract

In Section 3 we introduce a very simple 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 the Appendices, we have some technical analyses of the cases with un-matching numbers of buyers and sellers. These analyses are strongly related the the Oligopoly simulation project.

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[1] language (an interactive layer upon Python[2] using the Jupyter[3] 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[4].

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 @@@

1   The technical setup

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

Listing 1: Setup of the program

```
%pylab inline
```

---

[1] https://ipython.org
[2] https://www.python.org
[3] http://jupyter.org
[4] Clic directly.

```
%pylab inline
import statistics as s
import numpy as np
import pylab as plt
from ipywidgets import Output
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.[5]

We have two price vectors: pL^b with item pL_i^b for the buyers, and pL^s with item pL_j^s for the sellers. The ith or the jth elements of the vectors are prices, but we can use them also as agents.

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

In this 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 $u_i \sim \mathcal{U}(d_0, d_1)$;

---

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

- 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 to limit 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

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 two not overlapping price sequences that we can interpret as a demand curve and an offer one (Fig. 1).
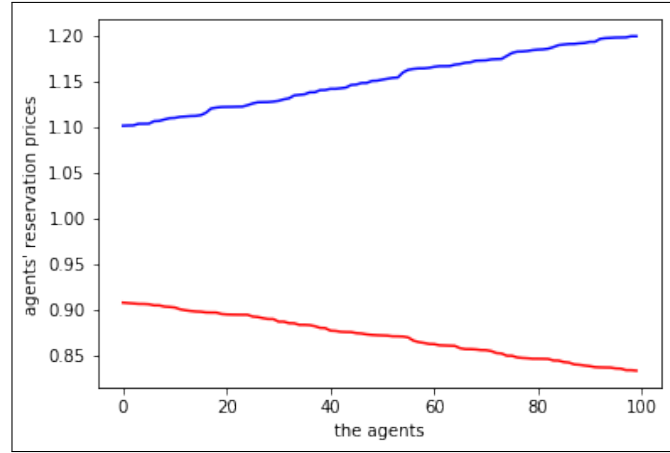
Figure 1: An example of initial not overlapping demand curve and offer curve

This is the *warming up*, or starting situation, of the model. 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 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.[6]

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

---

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

- 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 (preparing 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 $seed = 111$, $d_1 = 0.2$, $d_2 = 0.2$ 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).

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

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

.

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 for Matteo and Pietro: this is a very important effect for the *Oligopoly* model.

Have a look to the Appendix 1 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 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
```
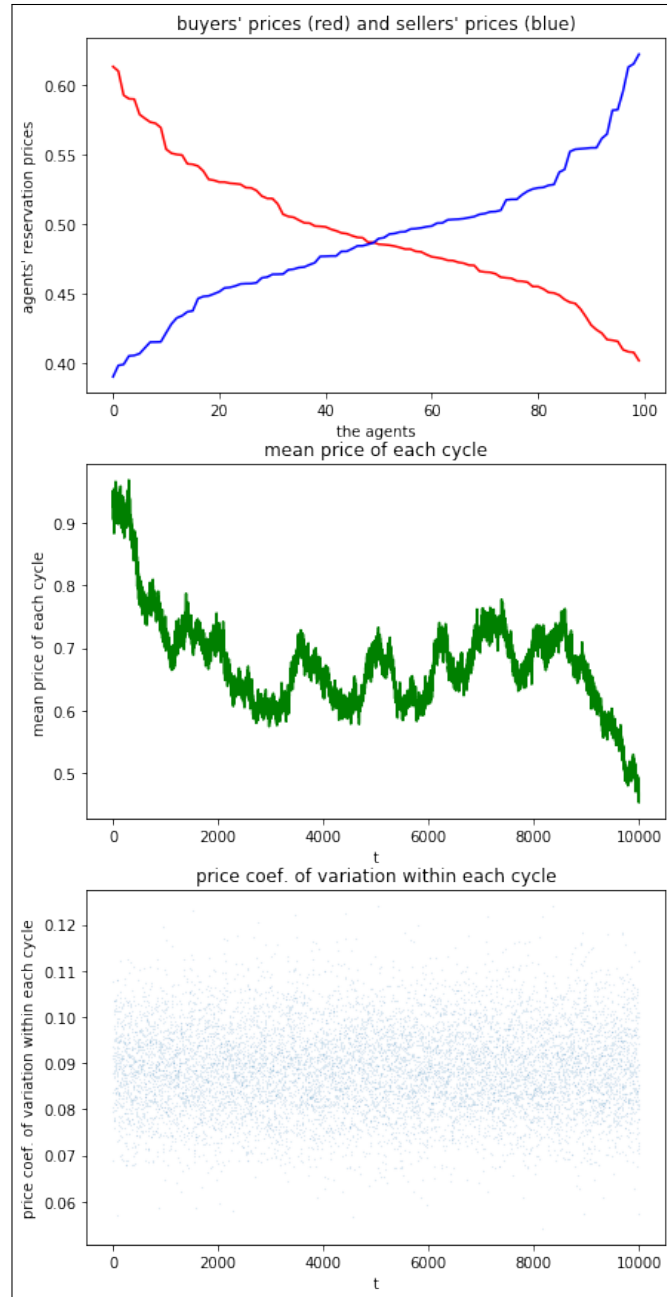
Figure 2: Hayekian case: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)
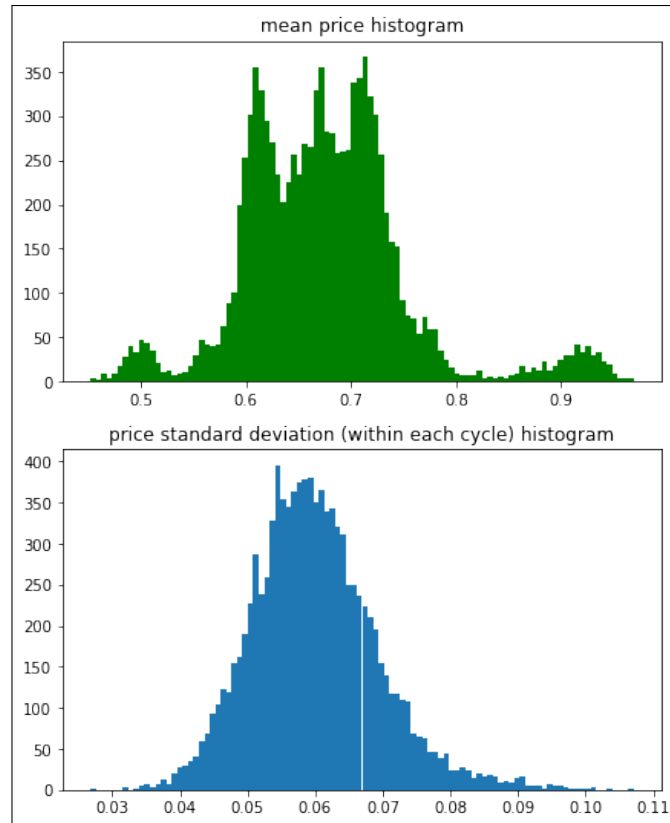
Figure 3: Hayekian case: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

```
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")
        #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 changing being multiplied in each cycle by following the 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 $seed = 111$, $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ set to $10,000$ we obtain exploding sequences of mean prices (mean in each cycle), and exploding standard deviation within each cycle (see Fig. 4 and 5).

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

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

.

> 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=[]
```
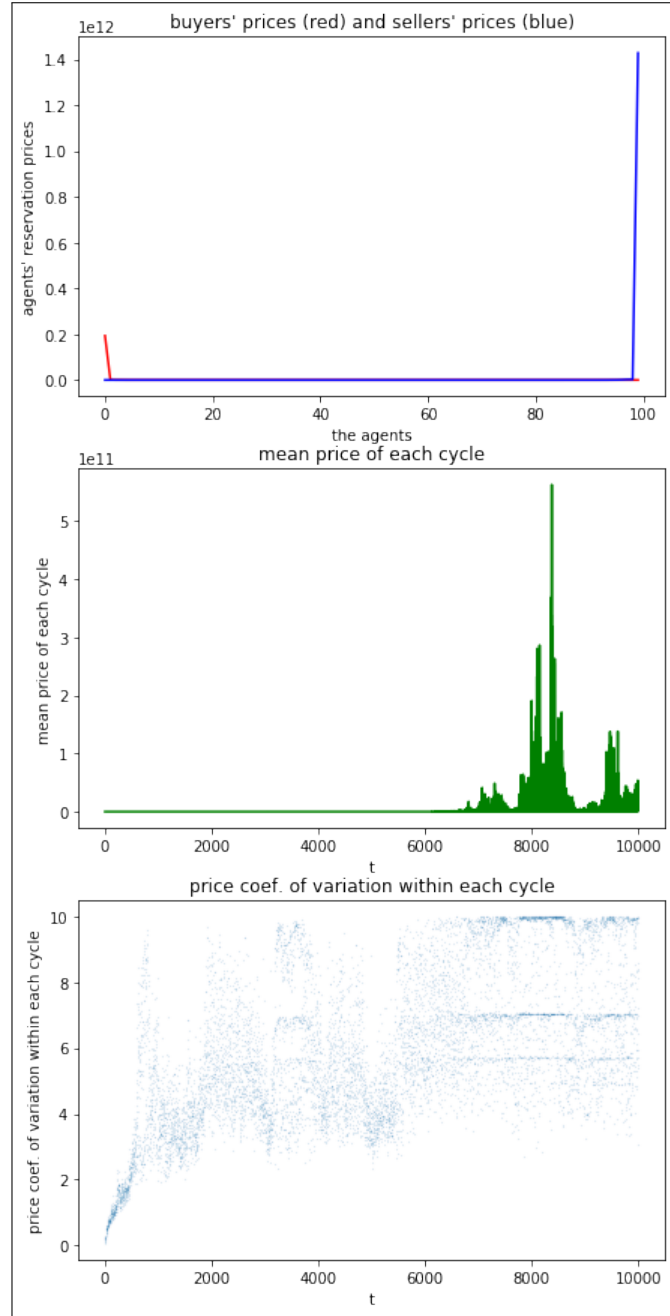
Figure 4: Unstructured case: ((i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)
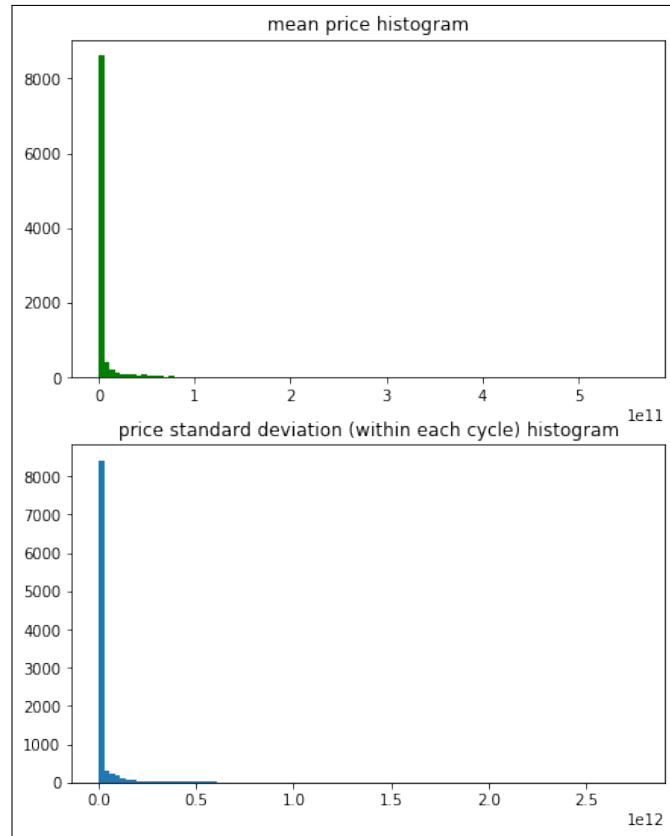
Figure 5: Unstructured case: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

```
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")
```

15

```
        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")
        #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

1 Two triple cases of not balancing numbers of buyers and sellers

1.1 Case $nBuyers \gg nSellers$

With $nBuyers \gg nSellers$ (e.g., $nBuyers = 100$ and $nSellers = 50$, as in Fig. 6), we have two possible path of analysis.

1.1.1 Case $nBuyers \gg nSellers$, with different rates of per-capita correction

If $nBuyers \gg nSellers$, 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*.

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

We can observe that in Figs. 7 and 8 the prices are—in the end—lower than in Figs. 2 and 3 and, must of all, the price tendence has a strong negative slope. We have always $seed = 111$.

This result is inconsistent with the microeconomic theory, where an excess of demand is supposed to increase the prices.
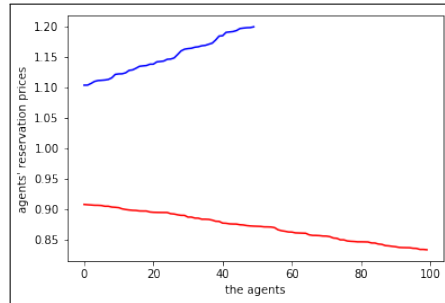

Figure 6: An example of initial not overlapping demand curve and offer curve, case $nBuyers \gg nSellers$

1.1.2 Case $nBuyers \gg nSellers$, with unequal rates of per-capita correction, with equivalent effects

Still with $nBuyers \gg nSellers$, 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
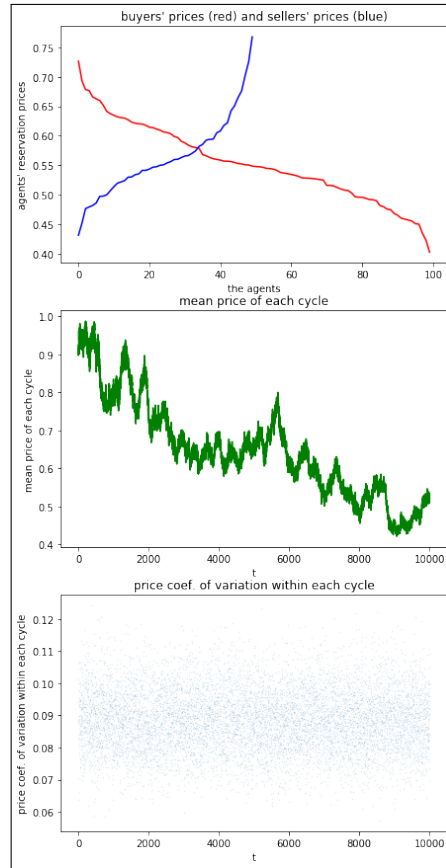
Figure 7: Hayekian case, with $nBuyers \gg nSellers$: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)
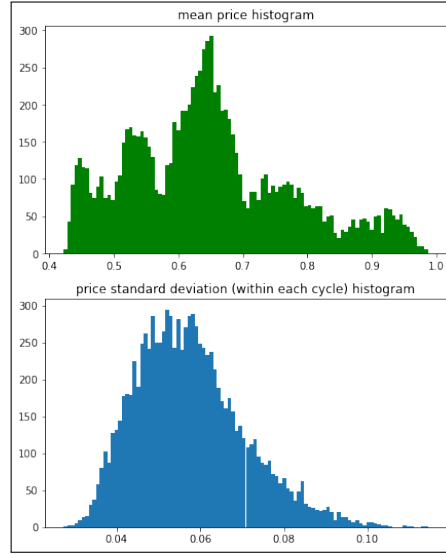
19

Figure 8: 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)

cycle is greater of the number of per-capita actions of the *buyers*.

In this second version of the experiment, the correction is that @@@ with unequal rates of per-capita correction, with equivalent effects

Always Fig. 6, and Fig.s 9, 10

AGGIUSTARE if the number of sellers is smaller than the number of buyers, the sellers act with a slow pace of price correction (proportionally to/with n/N) because in this way they can cherry-pick the best buyers and avoid to contribute to the fall of the prices

1.1.3 Case $nBuyers \gg nSellers$, with unequal rates of per-capita correction, but squeezing the effects

squeeze

1.2 Case $nBuyers \ll nSellers$

With $nBuyers \ll nSellers$ (e.g., $nBuyers = 50$ and $nSellers = 100$, as in Fig. 13), we have two possible path of analysis.
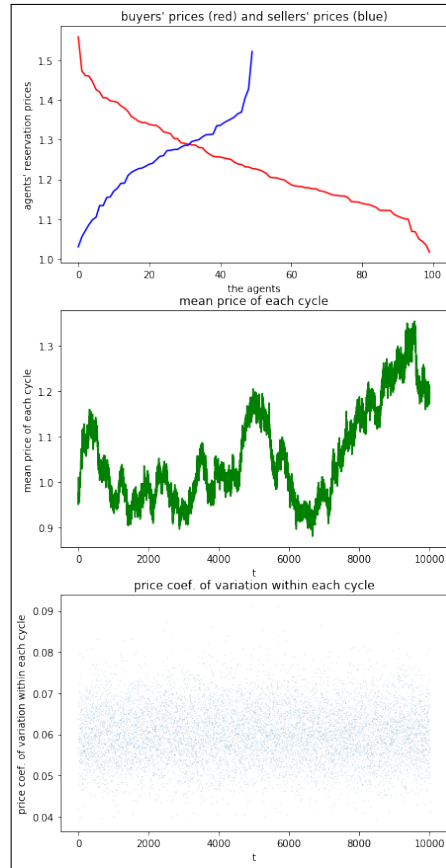
Figure 9: Hayekian case, with $nBuyers \gg nSellers$ but with equivalent effects: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)
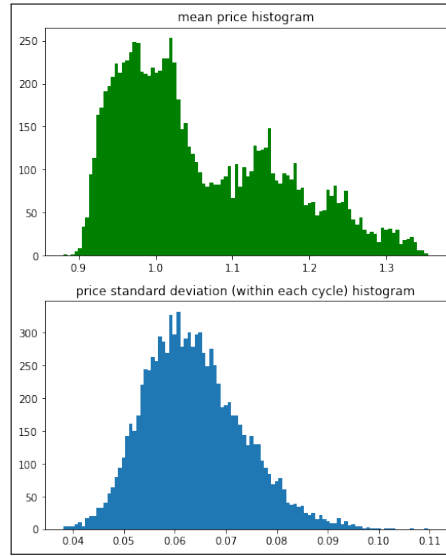
Figure 10: Hayekian case, with $nBuyers \gg nSellers$ but with equivalent effects: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

1.2.1 Case $nBuyers \ll nSellers$, with different rates of per-capita correction

If $nBuyers \ll nSellers$, having 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 a *seller* is greater than the probability that a *seller* decreases her price to meet a *buyer*.

We can observe that in Figs. 14 and 15 the prices are—in the end—greater than in Figs. 2 and 3 and, must of all, the price tendence has a strong positive slope. We have always $seed = 111$.

This result is inconsistent with the microeconomic theory, where an excess of offer is supposed to decrease the prices.

1.2.2 Case $nBuyers \ll nSellers$, with unequal rates of per-capita correction, with equivalent effects

Here we have unequal rates of per-capita correction, with equivalent effects

Always Fig. 13, and Fig.s 16, 17

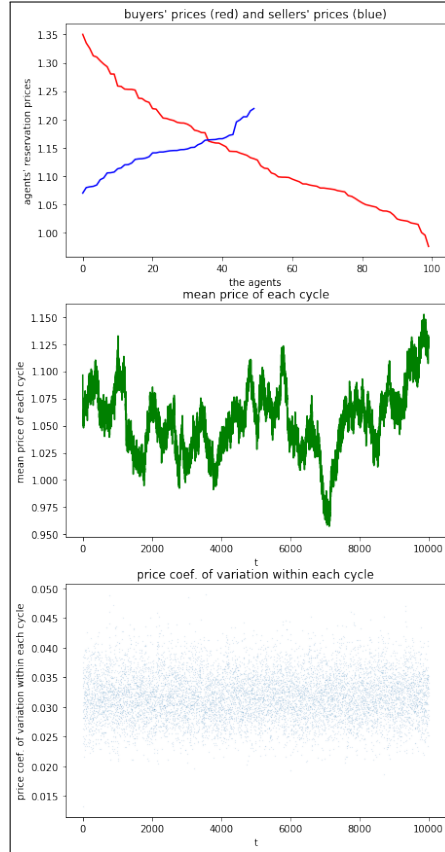AGGIUSTARE if the number of sellers is smaller than the number of buyers,

Figure 11: 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)
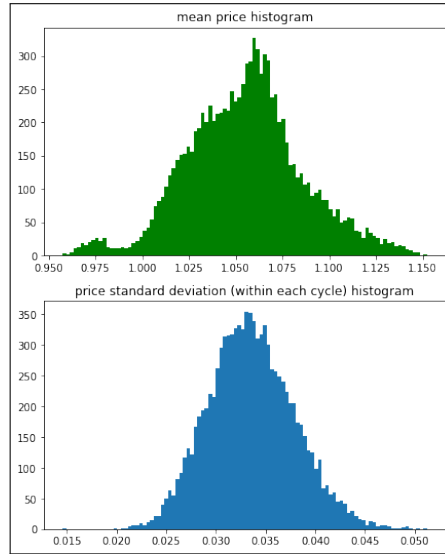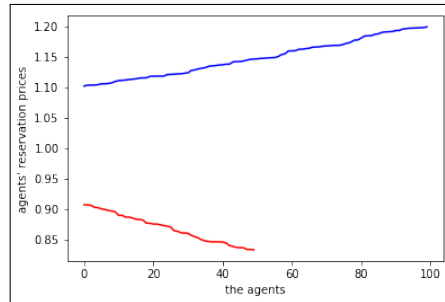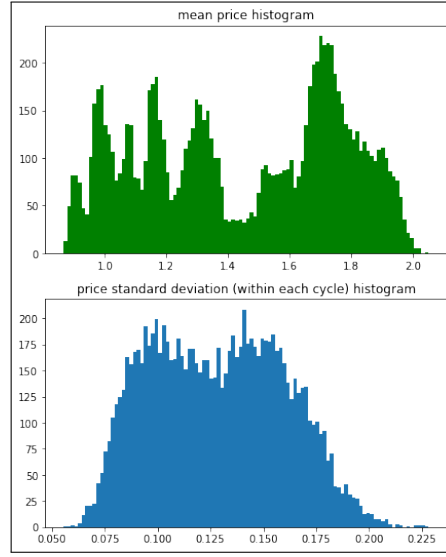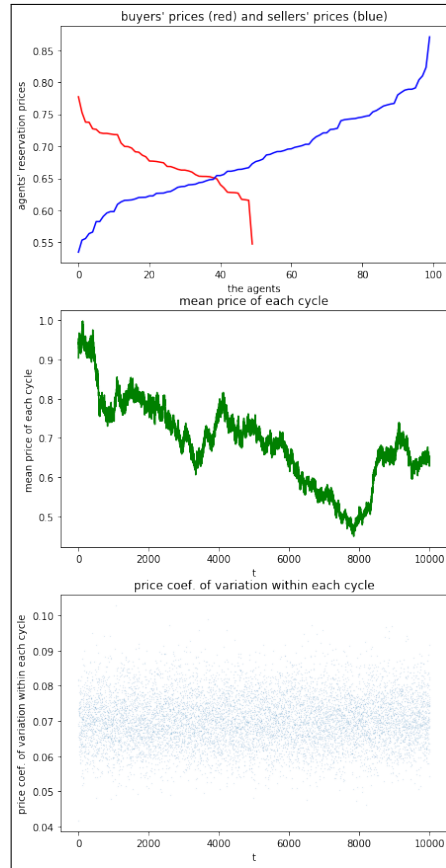
Figure 12: Hayekian case, with $nBuyers \gg nSellers$ but squeezing the effects: (i) the distribution of mean prices of each cycle (i.e., tick-by-tick) and (ii) that of their standard deviations within each tick (cycle)

Figure 13: An example of initial not overlapping demand curve and offer curve, case $nBuyers \ll nSellers$

24
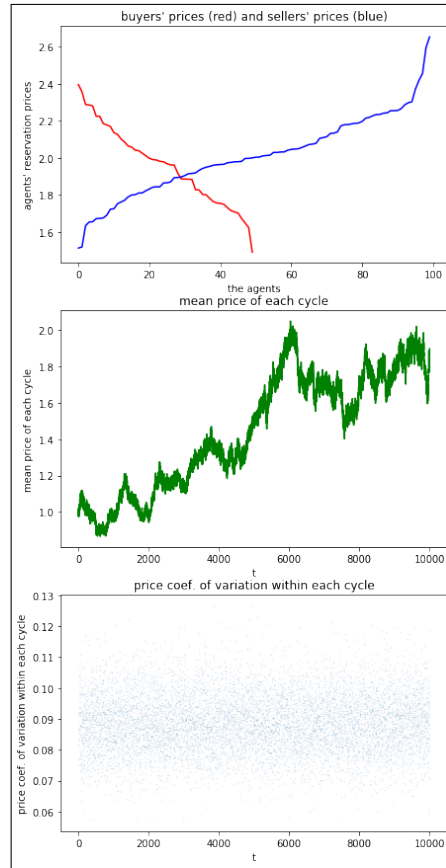
Figure 14: Hayekian case, with $nBuyers \ll nSellers$: (i) an example of final demand and offer curves, (ii) the history of mean prices tick-by-tick, (iii) their coefficients of variation within each tick (cycle)

Figure 15: 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)

the sellers act with a slow pace of price correction (proportionally to/with n/N) because in this way they can cherry-pick the best buyers and avoid to contribute to the fall of the prices

1.2.3 Case $nBuyers \ll nSellers$, with unequal rates of per-capita correction, but squeezing the effects

squeeze

Figure 16: 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)
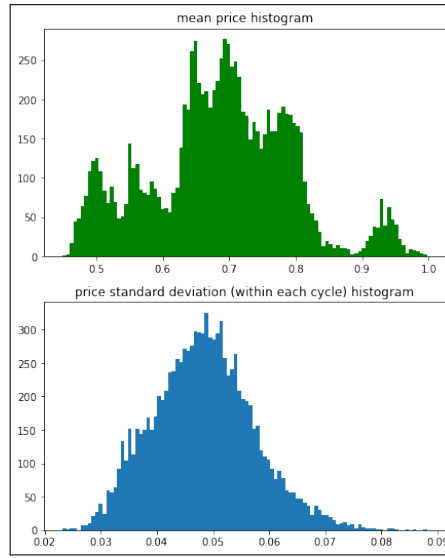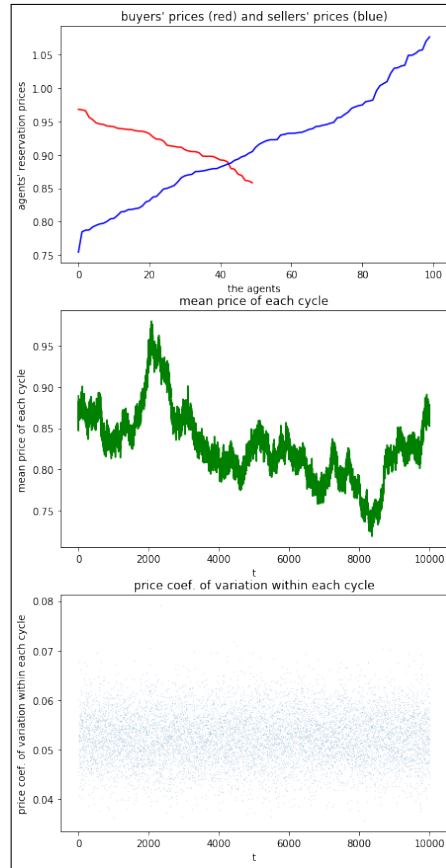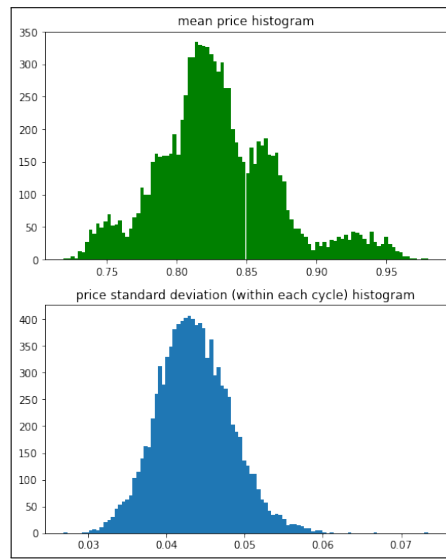
27

Figure 17: 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)

Figure 18: 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 19: 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)

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