micro Hayekian Market

Matteo Morini[1] and Pietro Terna[2]

July 23, 2018

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

Contents

List of Figures

Introduction to a micro 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 modification we generate implausible price dynamics.

The code uses the IPython[1] language (interaction with Python[2]) and can be dowloaded from https://github.com/terna/microHayekianMarket using the *Clone or download* button; it is also possible to run it directly on line at https://mybinder.org/v2/gh/terna/microHayekianMarket/master?filepath=microHayekianMarket.ipynb.

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

1 The technical setup

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

Listing 1: Warming up of the model

```
%pylab inline
import statistics as s
import numpy as np
import pylab as plt
from IPython.display import clear_output
import time
```

`%pylab inline` is a *magic* command of Jupyter.[3]

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

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

Our agents are simply prices, to be interpreted as reservation prices.[4]

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 in this case we can use them also as agents.

Both in the hayekian perspective (Section 3) and in the unstructured one (Section 4) we have a common *warning up* action.

In this phase we define:

- $nCycles$ - number of simulation cycles;
- $nBuyers$ - number of the buyers;
- $nSellers$ - number of the sellers;
- d_0 - the lower bound for 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 for random uniform numbers for the buyers;
- d_2 - the upper bound for random uniform numbers for the sellers;
- the initial buyer i reservation price, different for each buyer: $p_{b,i} = \frac{1}{1+u_i}$ with $u_i \sim \mathcal{U}(d_0, d_1)$;
- initial seller j reservation price, different for each seller: $p_{s,j} = 1 + u_j$ with $u_j \sim \mathcal{U}(d_0, d_2)$.

With $d_0 = 0.1$, $d_1 = 0.2$, $d_2 = 0.2$ and 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 and an offer curves (Fig. 1).

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.

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

Listing 2: Warming up of the model

```
# warming up

# run it before executing both - the hayekian perspective or
```

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

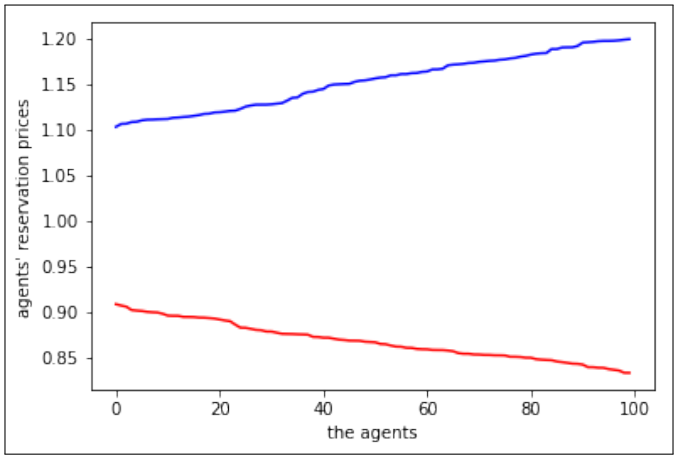

Figure 1: An example of initial not overlapping demand and offer curves

```
#                                   - the unstructured case

d0=0.1
d1=0.2
d2=0.2

nCycles=10000
nBuyers= 100
nSellers=100

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.plot(np.sort(buyerPriceList)[::-1],"r");
plt.plot(np.sort(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.[5]

In this version, which is the important one in this note, the running prices are changing following the correction coefficients:

- for the buyer: (i) $c_b = \frac{1}{1+u_b}$ if the deal succeeds (trying the 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) $c_s = \frac{1}{1+u_s}$ if the deal succeeds (preparing to obtain a higher revenue next time) or (iv) $c_s = 1 + u_s$ 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 $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ set to $10,000$ we obtain sequences of mean prices (mean in each cycle) quite realistic, with a very low variance within each cycle (see Fig. 2 and 3).

The *coefficient of variation* is calculated as $\frac{standard\ deviation}{mean}$.

A comment: we have a plausible series of mean price, 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 with agents not exchanging.

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

Listing 3: Warming up of the model

```
# hayekian perspective
meanPrice_ts=[]
meanPriceStDev_ts=[]
meanPriceVar_ts=[]

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)
        #print ('%2d %2d %.3f %.3f %.3f'% \
        #       (i,j,buyerPriceList[i]-sellerPriceList[j],\
        #        buyerPriceList[i],sellerPriceList[j]))

        if buyerPriceList[i]>=sellerPriceList[j]:
```

[5] In the *mall* sell price are public.

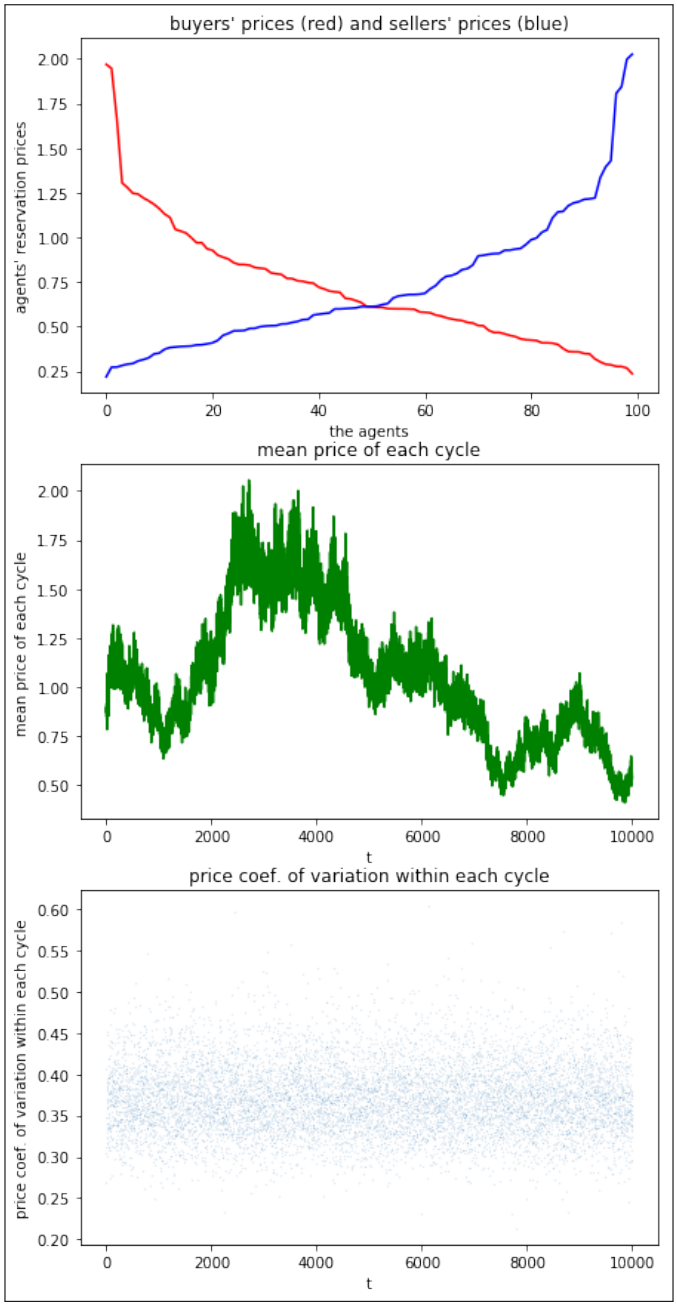

Figure 2: Hayekian case: (i) an example of final demand and offer curves, (ii) the history of mean prices, (iii) their coeffcients of variation within each cycle

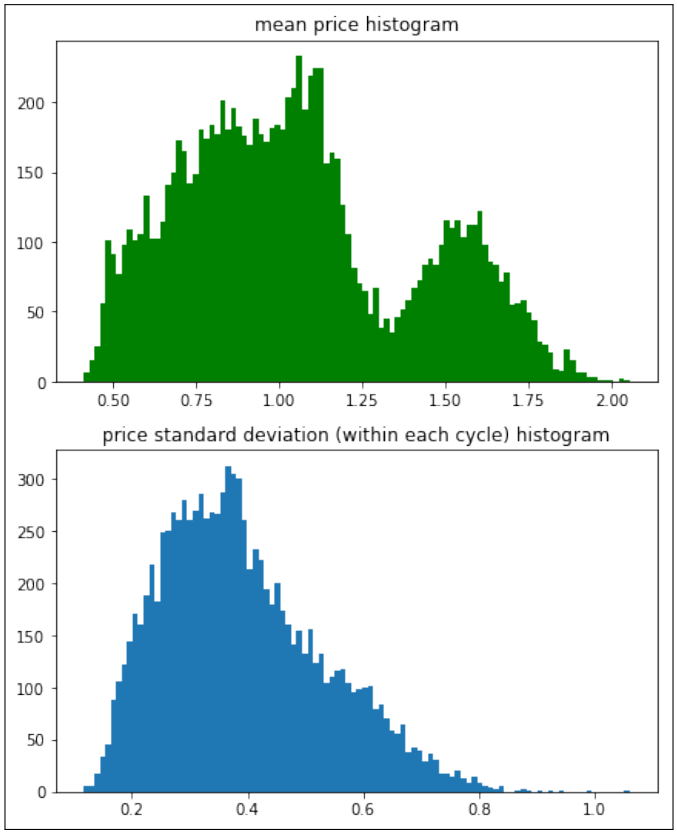

Figure 3: Hayekian case: (i) distribution of mean prices in each cycle and (ii) of their standard deviations

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

        #print ('%2d %2d %.3f %.3f %.3f \n'% \
        #      (i,j,buyerPriceList[i]-sellerPriceList[j],\
        #       buyerPriceList[i],sellerPriceList[j]))

    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:
        clear_output()
        print('time', t, 'and␣n.␣of␣exchanges␣in␣the␣last␣cycle', \
              len(dealPrices))
        print(\
        'mean␣and␣var␣of␣exchange␣prices␣in␣the␣last␣cycle:␣%.3f,␣%.3f' %\
              (meanPrice_ts[-1],meanPriceVar_ts[-1]))

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

        plt.subplot(311)
        plt.plot(np.sort(buyerPriceList)[::-1],"r")
        plt.plot(np.sort(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")
        show()
        #time.sleep(0.1)

plt.figure(2,figsize=(7,9))
plt.subplot(211)
plt.title("mean price histogram")
plt.hist(meanPrice_ts,100,color="g");
plt.subplot(212)
plt.title("price standard deviation (within each cycle) histogram")
plt.hist(meanPriceStDev_ts,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 of the buyer.

In this version the running prices are changing following the correction coefficients:

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

With $d_1 = 0.2$, $d_2 = 0.2$ and $nCycles$ set to $10,000$ we obtain exploding sequences of mean prices (mean in each cycle), and explodinf variance within each cycle (see Fig. 4 and 5).

The *coefficient of variation* is calculated as $\frac{standard\ deviation}{mean}$.

A comment: this counter-example shows the missing the intelligent correction of the price that implicitly propagate the price 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: Warming up of the model

```
# unstructured case (remember the warming up step)
meanPrice_ts=[]
meanPriceStDev_ts=[]
meanPriceVar_ts=[]

for t in range(1,nCycles+1):
```

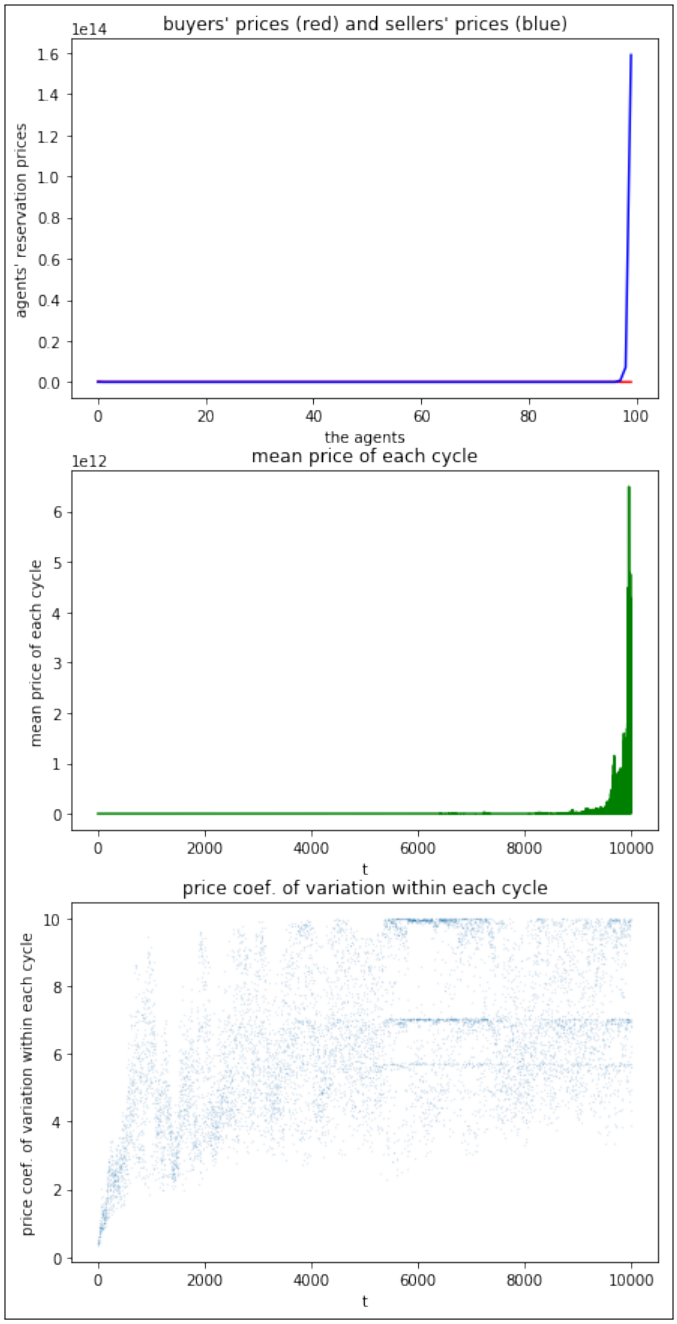

Figure 4: Unstructured case: (i) an example of final demand and offer curves, (ii) the history of mean prices, (iii) their coeffcients of variation within each cycle

Figure 5: Unstructured case: (i) distribution of mean prices in each cycle and (ii) of their standard deviations

```
dealPrices=[]
agNum=max(nBuyers,nSellers)
for n in range(agNum):
    i = np.random.randint(0,nBuyers)
    j = np.random.randint(0,nSellers)
    #print ('%2d %2d %.3f %.3f %.3f'% \
    #      (i,j,buyerPriceList[i]-sellerPriceList[j],\
    #       buyerPriceList[i],sellerPriceList[j]))

    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))

    #print ('%2d %2d %.3f %.3f %.3f \n'% \
    #      (i,j,buyerPriceList[i]-sellerPriceList[j],\
    #       buyerPriceList[i],sellerPriceList[j]))

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:
    clear_output()
    print('time', t, 'and␣n.␣of␣exchanges␣in␣the␣last␣cycle', \
          len(dealPrices))
    print(\
    'mean␣and␣var␣of␣exchange␣prices␣in␣the␣last␣cycle:␣%.3f,␣%.3f' %\
          (meanPrice_ts[-1],meanPriceVar_ts[-1]))

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

    plt.subplot(311)
    plt.plot(np.sort(buyerPriceList)[::-1],"r")
    plt.plot(np.sort(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")
        show()
        #time.sleep(0.1)

plt.figure(2,figsize=(7,9))
plt.subplot(211)
plt.title("mean price histogram")
plt.hist(meanPrice_ts,100,color="g");
plt.subplot(212)
plt.title("price standard deviation (within each cycle) histogram")
plt.hist(meanPriceStDev_ts,100);
```

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