Homework 6

Homework 6-1

0.0/3.0 points (graded)

The original dealer model that we have used in the course is a minimal model. As such, there is ample room for improvement. One of the obvious modifications, to obtain a more realistic description, is to consider $N > 2$ dealers. The dynamics of the dealers are unchanged,

$$p_i(t+\Delta t) = p_i(t) + d\langle \Delta P\rangle_M \Delta t + c f_i(t), \qquad i = 1, 2 \qquad \text{(M1)}$$

$$f_i(t) = \begin{cases} +\Delta p & \text{prob.}1/2 \\ -\Delta p & \text{prob.}1/2 \end{cases} \qquad \text{(M2)}$$

$$\langle \Delta P\rangle_M = \frac{2}{M(M+1)} \sum_{k=0}^{M-1} (M-k)\,\Delta P(n-k) \qquad \text{(M3)}$$

as well as the transaction condition

$$|p_i(t) - p_j(t)| \geq L. \qquad \text{(M4)}$$

Thus, we have N independent random walks (evolving according to Eq.(M1-M2)). Whenever the distance between the prices of any two dealers is greater than the spread L (Eq. (M4)), these two (say i and j) can carry out a transaction. If a transaction takes place between i and j, the market price is updated to reflect this, ($P_{n+1} = \frac{1}{2}(p_i(t) + p_j(t))$), and the mid-prices of i and j are reset accordingly ($p_i(t+1) = p_j(t+1) = P_{n+1}$). We must also care about how the mid-prices of the remaining dealers ($k \neq i \neq j$) are affected by this transaction. Assume the following:

- If more than two pairs of dealers satisfy the transaction condition, the dealer with the highest price will buy from the dealer with the lowest price. If more than two dealers have the same price, we chose only one of them. In this case, only one transaction between two dealers can take place at any given time.

- The remaining dealers switch their prices to the new market price $(p_k\,(t+1) = P_{n+1})$ with a probability r, and they keep their old prices with probability $(1-r)$. Here, $0 \leq r \leq 1$ and r is a constant equal for all dealers.

To simulate a system with $N = 100$ dealers with a switching rate $r = 0.6$ the following erroneous modification of the model 2 code has been proposed.

```
params={'N':100,'r':0.6,'L':0.01,'c':0.01,'dp':0.01,'dt':0.01**2, 'd':1.00, 
p0 = np.ones(params['N'])*100.25

def model2(params,p0,numt):
    pswitch  = np.array([params['r'], 1.0-params['r']]) #switching rates

    # compute running average Eq.(L6)
    def avgprice(dpn):
        M = len(dpn)
        weights  = np.array(range(1,M+1))*2.0/(M*(M+1))
        return weights.dot(dpn)

    # return transaction candidate: pair (i,j) with larget price difference
    def transactionPair(pt):
        return [np.min(pt), np.max(pt)]

    # update prices for transaction between the two dealers (i,j) and  return
    def transactionUpdate(i,j,pt):
        newprice = np.average([pt[i], pt[j]]) # set mid price to new marker p
        switch   = np.random.choice([True,False], p=pswitch,size=params['N'])
        switch[i]=switch[j]=True
        pt[switch] = newprice
        return newprice

    # ... variable initializations ...
    mktprice = np.zeros(numt)   # initialize market price P(n)
    dmktprice= np.zeros(numt)   # initialize change in price dP(n) needed for
    ticktime = np.zeros(numt,dtype=np.int) #initialize array for tick times
    price    = p0               #initialize dealer's mid-price (p1,p2)
    time,tick= 0,0 # real time(t) and time time (n)
    deltapm  = 0.0 # trend term d <dP>_m dt for current random walk
    cdp      = params['c']*params['dp'] # define random step size
    ddt      = params['d']*params['dt'] # define amplitude of trend term

    while tick < numt: # loop over ticks
        [i,j] = transactionPair(price)
        while np.abs(price[i]-price[j]) < params['L']: # transaction criterio
            price = price + deltapm + np.random.choice([-cdp,cdp], size=param
            [i,j] = transactionPair(price)            # update transaction c
            time += 1 #update real time

        mktprice[tick] = transactionUpdate(i,j,price)  # save market price

        # ... finalize loop ...
        dmktprice[tick]= mktprice[tick] - mktprice[np.max([0,tick-1]) # save
        ticktime[tick] = time # save transaction time
        tick += 1             #update ticks
        tick0 = np.max([0, tick - params['M']])       #compute tick start for
        deltapm = avgprice(dmktprice[tick0:tick])*ddt #compute updated trend
    return ticktime,mktprice
```

Identify the mistake in the code from the following options.

C1. The traders not involved in the transaction do not update their prices correctly

The random choice in the transactionUpdate function should be

```
def transactionUpdate(i,j,pt):
    #...
    switch   = np.random.choice([False,True], p=pswitch,size=params['N'])
    #...
```

C2. The price of a new transaction is incorrect

The new price in the transactionUpdate function should be computed as

```
def transactionUpdate(i,j,pt):
    newprice = np.average(pt)
    #...
```

C3. The calculation of the transaction pair candidates (i, j) is incorrect

The transactionPair function should be modified as follows

```
def transactionPair(pt):
    return [np.argmin(pt), np.argmax(pt)]
```

C4. The price update for the individual dealers is incorrect

The correct update should be given as

```
while tick < numt: # loop over ticks
    #...
    while np.abs(price[i]-price[j]) < params['L']:
        price = price + deltapm + np.random.choice([-cdp,cdp], 2)
    #...
```

- ◯ C1
- ◯ C2
- ◯ C3
- ◯ C4

Submit You have used 0 of 2 attempts

Homework 6-2

0.0/3.0 points (graded)

Continuing with the previous problem, we also would like to consider the case where the trend-following parameter d_i is different for different dealers, such that

$$p_i(t+\Delta t) = p_i(t) + d_i \langle \Delta P \rangle_M \Delta t + c f_i(t), \qquad i = 1,2 \qquad \text{(M1)}$$

$$d_i = \bar{d}(n) + \Delta d_i$$

where $\bar{d}$ is the average trend-following effect (which is changing in time), and Δd_i describes each dealer's deviation from this average. We assume that the Δd_i are given by a random normal distribution with zero average and standard deviation σ. Note that the Δd_i, for a given i, are constant (characteristic of each dealer). However, the average trend-following effect is evolving in time according to the following rule (Yamada et al., PRE, 2009)

$$\bar{d}(n+1) = (1-e_0)\,\bar{d}(n) + \phi(n)$$

where e_0 is a constant ($0 < e_0 < 1$) and $\phi(n)$ is an independent random noise

$$\phi(n) = \begin{cases} +0.01 & \text{probability } 1/2 \\ -0.01 & \text{probability } 1/2 \end{cases}$$

The following modifications to the (corrected) code of the previous exercise have been proposed and used to simulate a system with $e_0 = 10^{-4}$ and $\sigma = 0.1$. Note that unchanged blocks from the previous code are not shown.

```
params={'e0':1e-4,'sigma':0.1,'N':100,'r':0.6,'L':0.01,'c':0.01,'dp':0.01,'dt
p0  = np.ones(params['N'])*100.25
def model2(params,p0,numt):
    pswitch  = np.array([1.0-params['r'], params['r']]) #switching rates

    # compute running average Eq.(L6)
    def avgprice(dpn):
    #...

    # return indices (i,j) of dealers with smallest and largest prices
    def transactionPair(pt):
    # ...

    # update prices for transaction between the two dealers (i,j) and  return
    def transactionUpdate(i,j,pt):
    # ...

    # ... variable initializations
    deltapm  = 0.0                                          # trend term <dP
    dbar     = params['d']                                  # define average
    di       = params['sigma']*np.random.randn(params['N'])  # trend followin

    while tick < numt: # loop over ticks
        [i,j]    = transactionPair(price)   # transaction candidate
        while np.abs(price[i]-price[j]) < params['L']: # transaction criterio
            price = price + deltapm*di + np.random.choice([-cdp,cdp], size=pa
            [i,j] = transactionPair(price) # update transaction candidate
            time += 1                      # update real time

        dbar = (1.0 - params['e0'])*dbar + np.random.choice([0.01,-0.01])
        mktprice[tick] = transactionUpdate(i,j,price)

        # ... finalize loop
        tick += 1             # update ticks
        tick0 = np.max([0, tick - params['M']])       # compute tick start fo
        deltapm = avgprice(dmktprice[tick0:tick])*params['dt'] # compute upda
  return ticktime,mktprice
```

However, there there is still a problem with this new code. Identify the mistake from the following options

C1. The average trend following behavior, as given by $\bar{d}$, is not calculated correctly.

The correct value for $\bar{d}$ si given by

```
def model2(params,p0,numt):
    # ...
    while tick < numt: # loop over ticks
        while np.abs(price[i]-price[j]) < params['L']: # transaction criterio
            #...
        #...
        dbar = (1.0 - params['e0'])*dbar + np.random.choice([0.01,-0.01], siz
        #...
```

C2. The price is not correctly updated

The correct value for the new price should be calculated as follows

```
def model2(params,p0,numt):
    # ...
    while tick < numt: # loop over ticks
        while np.abs(price[i]-price[j]) < params['L']: # transaction criterio
            price = price + deltapm*(dbar + di) + np.random.choice([-cdp, cdp
            #...
        #...
```

C3. The average trend following behavior, as given by $\bar{d}$, is not calculated correctly.

$\bar{d}$ should be updated at every time step as follows

```
def model2(params,p0,numt):
    # ...
    while tick < numt: # loop over ticks
        while np.abs(price[i]-price[j]) < params['L']: # transaction criterio
            # price update ...
            dbar = (1.0 - params['e0'])*dbar + np.random.choice([0.01,-0.01])
            # transaction candidate ...
        #...
```

C4. The price deviations of the individual dealers, as given by Δd_i are not updated

Δd_i should be updated at each time step

```
def model2(params,p0,numt):
    # ...
    while tick < numt: # loop over ticks
        while np.abs(price[i]-price[j]) < params['L']: # transaction criteric
            di = params['sigma']*np.random.randn(params['N'])
            # price update...
            # transaction candidate
        # ...
```

- C1
- C2
- C3
- C4

Submit You have used 0 of 2 attempts