Garman

December 11, 2020

1 Simulating the Garman Inventory Model

NULL SPACE RESEARCH

Here are the main variables in the algorithm:

1. $I_s(t)$ denotes the **stock inventory** of the Dealer at discrete time step t.
2. $I_c(t)$ denotes the **cash inventory** of the Dealer at discrete time step t.
3. At each successive time step, if a Dealer receives a **sell order** - that is if other people want to sell to the Dealer at his **bid price** - then we add one stock unit to the Dealer's **stock inventory** and we subtract the bid price from the Dealer's **cash inventory**.

$$I_s(t+1) = I_s(t) + 1$$

$$I_c(t+1) = I_c(t) - bid$$

4. At each successive time step, if a Dealer receives a **buy order** - that is if other people want to buy from the dealer at his **ask price** - then we subtract one stock unit from the Dealer's **stock inventory** and we add the ask price to the Dealer's **cash inventory**

$$I_s(t+1) = I_s(t) - 1$$

$$I_c(t+1) = I_c(t) + ask$$

5. We have assumed that the probability of getting a sell order in the next time step is 0.75 and the probability of getting a buy order is 0.25. This satisfies the condition:

$$\lambda_d > \lambda_a$$

6. Also make sure that when you enter your ask and bid prices in the program - in order for you to be rational in this model - you must set your ask price more than your bid price. This satisfies another model condition:

$$ask > bid$$

2 Going about the Program

Here are the steps of execution of this program:

1. You are first required to run the function **registration_desk()** - This serves to prompt the user to essentially become a Dealer and enter the following values - $(stock, cash, ask, bid, name)$
2. The input values then turn you into a **MarketMaker** object wherein the **step function** is specified - the rate of buy and sell orders and the model equation increments are specified here.
3. You are then required to implement the function **status_after_market()** - This function invokes yet another function within itself called **start_market_operations()** which simulates the market 10000 times.
4. Finally the status report is printed - this status report contains a **dataframe** of your stock and cash inventory level at each time step. And it also prints a graph that cumulatively shows the trend in stock and cash inventory over time.

[99]:
```
import random
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')
```

[170]:
```
def registration_desk():
    print("========")
    print()
    print("Welcome to the Dealer's registration desk!!")
    print()
    print("Please enter the following details to participate in the market:")
    print()

    name = input("What is your name: ")
    stock_inv = int(input("amount of stock inventory you carry: "))
    cash_inv = int(input("amount of cash inventory you have: "))
    ask_price = int(input("enter the ask price you want to set at the start: "))
    bid_price = int(input("enter the bid price you want to set at the start: "))

    return MarketMaker(stock_inv, cash_inv, ask_price, bid_price, name)
```

[171]:
```
d1 = registration_desk()
```

```
========

Welcome to the Dealer's registration desk!!
```

```
Please enter the following details to participate in the market:


What is your name:  Akash
amount of stock inventory you carry:  50
amount of cash inventory you have:  500
enter the ask price you want to set at the start:  104
enter the bid price you want to set at the start:  101
```

[173]:
```python
class MarketMaker(object):
    def __init__(self, s, c, ask, bid, name):
        self.s = s
        self.c = c
        self.ask = ask
        self.bid = bid
        self.name = name

    def getStockInventory(self):
        return self.s

    def getCashInventory(self):
        return self.c

    def getName(self):
        return self.name

    def oneStep(self):
        possibilities = {'sell_order': (1, -self.bid), 'buy_order': (-1, self.
 ↪ask)}
        return random.choice(list(possibilities.items()), p = [0.25, 0.75])

    def __str__(self):
        return self.name + ':' + '(' + 'stock: '+ str(self.s) + ',' + ' ' +␣
 ↪'cash: ' + str(self.c) \
                + ',' + ' ' + 'ask: ' + str(self.ask) + ',' + ' ' + 'bid: ' +␣
 ↪str(self.bid) + ')'
```

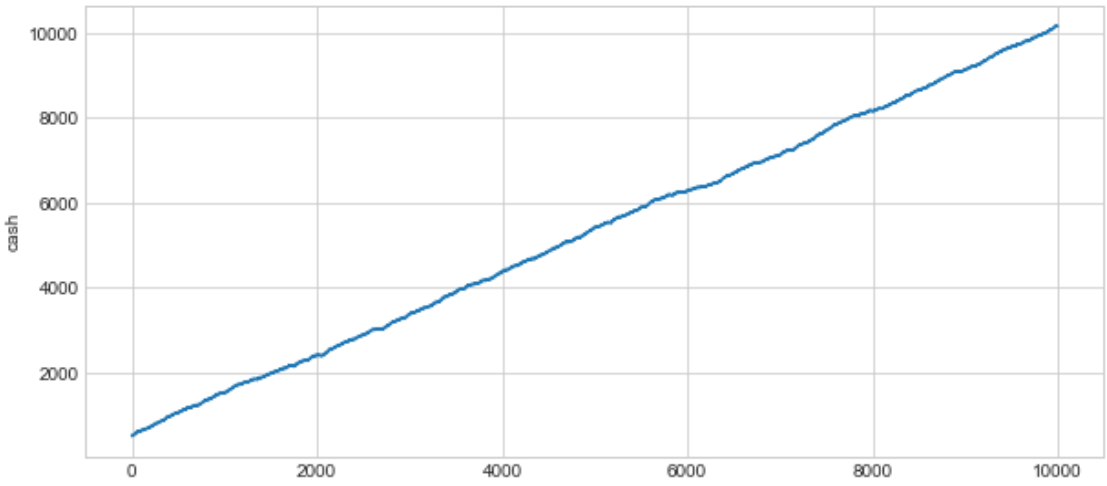
[175]:
```python
def start_market_operations(dealer):
    time = []
    stock = []
    cash = []
    IS = dealer.getStockInventory()
    IC = dealer.getCashInventory()
    stock.append(IS)
    cash.append(IC)
    time.append(0)
    table = pd.DataFrame()
```

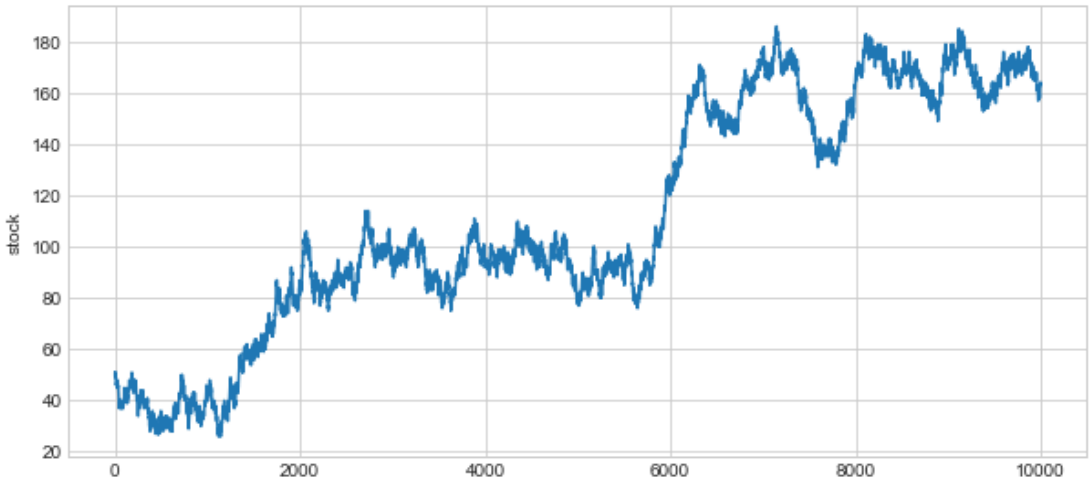
```
    print('The market movements of Dealer: ', dealer.getName())
    print()
    print('Dealer information as provided by dealer: ', dealer)

    for i in range(10000):
        if IS == 0 or IC == 0:
            break

        tup = a.oneStep()
        if tup[0] == 'buy_order':
            IS += tup[1][0]
            IC += tup[1][1]
            stock.append(IS)
            cash.append(IC)
        elif tup[0] == 'sell_order':
            IS += tup[1][0]
            IC += tup[1][1]
            stock.append(IS)
            cash.append(IC)
        time.append(i+1)

    table['time'] = time
    table['stock_inventory'] = stock
    table['cash_inventory'] = cash

    return table
```

[176]:
```
def status_after_market(dealer):
    status = start_market_operations(dealer)
    print("=========")
    print()
    print("Here is the table for stock and cash inventories over time")
    print()
    print(status)
    print()
    print("=========")
    print()
    print("The graphical trajectory of stock and cash")
    print()

    fig, ax = plt.subplots(2, figsize = (10, 10))
    ax[0].plot(status['time'], status['cash_inventory'])
    ax[0].set_ylabel('cash')
    ax[1].plot(status['time'], status['stock_inventory'])
    ax[1].set_ylabel('stock')
```

[177]:
```
status_after_market(d1)
```

```
The market movements of Dealer:  Akash

Dealer information as provided by dealer:  Akash:(stock: 50, cash: 500, ask:
104, bid: 101)
=========

Here is the table for stock and cash inventories over time

        time  stock_inventory  cash_inventory
0          0               50             500
1          1               51             498
2          2               50             502
3          3               49             506
4          4               50             504
…      …               …              …
9996    9996              162           10160
9997    9997              163           10158
9998    9998              164           10156
9999    9999              163           10160
10000  10000              164           10158

[10001 rows x 3 columns]

=========

The graphical trajectory of stock and cash
```


cash
10000
8000
6000
4000
2000
0
2000
4000
6000
8000
10000


stock
180
160
140
120
100
80
60
40
20
0
2000
4000
6000
8000
10000