Inventory Modeling in SimPy

Jimmy McLaughlin

October 18, 2018

Abstract

This report aims to find the most efficient way to manage inventory in a simplified supply chain network with only a single store over a four day period. In particular, we examine the relevant decision-making criteria and balance the various trade-offs that arise in inventory management. Two ordering policies are considered: a constant delivery policy and an order-up-to policy. We create a simulation in SimPy to model the store and compare the respective total store costs for each of the two ordering strategies. Assuming customer demand to be constant, the Poisson Process is used to model customer arrivals at our store. The simulation keeps track of stock level, customer backorder, number of deliveries ordered by the store, and number of items stored in inventory.

1 Introduction

Companies need good inventory systems in order to minimize operating costs while managing customer demand. For example, goods-based companies such as Apple and Nike must monitor their inventory levels to determine when to restock and choose the ordering quantity accordingly so that customer demand can be met. J.P. Morgan, although being a service-based rather than goods-based corporation, analyzes inventory management policies and inventory/sales ratios of potential assets to help determine which would be sound investments. Retailers like Walmart use a technique known as vendor-managed inventory policy, in which suppliers have direct access to databases detailing current inventory levels at each individual outlet. This allows inventory to be restocked quickly and helps reduce personnel costs. In general, inventory systems implement some inventory replenishment policies to manage storage space and respond to customer demand. These

policies must account for limited capacity and cost constraints as well as be sensitive to differences between production and demand.

Inventory problems are common in operations research, and due to the difficulty of solving such problems explicitly, simulation of inventory systems is commonly used to tackle them. Simulation is a useful tool for highlighting any issues which may exist in a model, as it can account for often unforeseen possibilities. Simulations are comprehensive and give companies a good idea of what to expect before implementing a real-life system. Computer software is used to run simulations, which are generally interactive and user-friendly. The more information and parameters a user can provide for a simulation, the more effective it will be. Simulations are often created in C++ and R, although Python's SimPy package was the method of choice in this report.

As mentioned earlier, inventory management is practiced by various types of companies including retailers, goods-based companies, service-based companies, and E-commerce companies. However, this report will focus specifically on an example of a retailer. For simplicity, it will examine only one retail store rather than a chain of many stores. Thus it will not discuss warehouses, which act as middle men between suppliers and stores. In a large retail network, stores will place supply requests to the supplier, who will then distribute these requests to a warehouse. The warehouse, in turn, will distribute the requested items to each store in a certain region. Assume for this report that the store receives items directly from the supplier, and observe that delivery time is not factored into the model.

2 Research Objective

The goal of this report is to find the best inventory management strategy for a store given a set of cost constraints. Two ordering policies are compared: a Constant Delivery Ordering Policy and an Order-up-to Ordering Policy. This section will discuss the differences between these two policies as well as the various costs which inventory management incurs.

2.1 Trade-offs in Inventory Management

Inventory cost can be broken down into three categories: holding, shipping, and backorder. Holding cost is the price businesses must pay to store goods. Money lost from goods which expire, are

damaged, or become obsolete factors into holding cost. Shipping cost is what companies pay to restock their inventory. It has two elements: a flat delivery fee and a per-item delivery fee. The flat fee is what prevents continual supply ordering. Backorder cost is the loss of potential profit from failing to meet customer demand. Insufficient inventory levels will generally result in customers taking their business elsewhere.

2.2 Constant Delivery Ordering Policy

In the Constant Delivery Ordering Policy, a company will order a predetermined amount of goods once every review period. The review period can be any given amount of time: hours, days, weeks, etc. In this model, the review period is one day. The amount of items ordered is called the **order quantity**. The order quantity should not cause the inventory level to exceed the maximum store capacity, or else items will be lost or sent back, which is something companies must consider when constructing their policy.

2.3 Order-up-to Ordering Policy

In the Order-up-to Ordering Policy, a company will choose a certain inventory level that it wants to have at the end of each review period, and order just enough items so that it reaches this inventory level (which is known as the **delivery threshold**). If the current inventory level is greater than or equal to the delivery threshold, then no order will be placed. The times at which items are ordered are called **reorder points**. Often, companies will have a **safety stock**, which is a certain amount of items that must always be in storage to avoid backorder. This safety stock will be included as part of the delivery threshold. If inventory is frequently depleted, then a company may want to consider increasing the delivery threshold.

2.4 Comparing the Two Ordering Policies

The Constant Delivery Ordering Policy relies on customer demand being relatively consistent. Otherwise it is impossible to compute a single order amount that will continually satisfy the demand. The Order-up-to Policy, however, is less dependent on predictable demand. Having more items purchased than usual during a given day will not be an issue (unless so many items are

purchased that the supply runs out) since the company can restock to their delivery threshold afterwards. Additionally, an advantage of the Order-up-to Policy is that if only a small amount of items are purchased in a day, the company can save money on shipping costs by only having to order a few items for the next day.

Below are included graphs of inventory level over the four day period for each of the two ordering policies after running the simulation once. The graphs show that the inventory level in the Constant Delivery Ordering Policy is trending downwards over time, whereas in the Order-up-to Ordering Policy the inventory level resets every 24 hours. Note that for the Order-up-to graph, the dip in inventory level is larger on the third day than on the second. This will not always be the case; it is simply a consequence of having random daily demand.

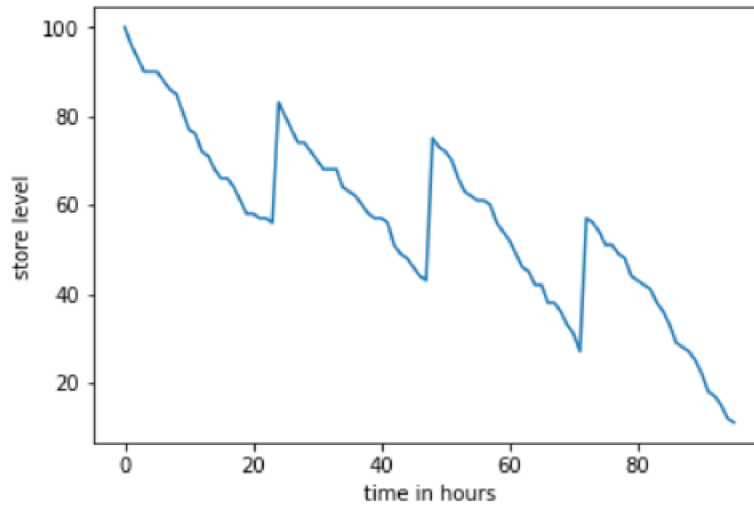

Figure 1: Inventory level of store using Constant Delivery Ordering Policy

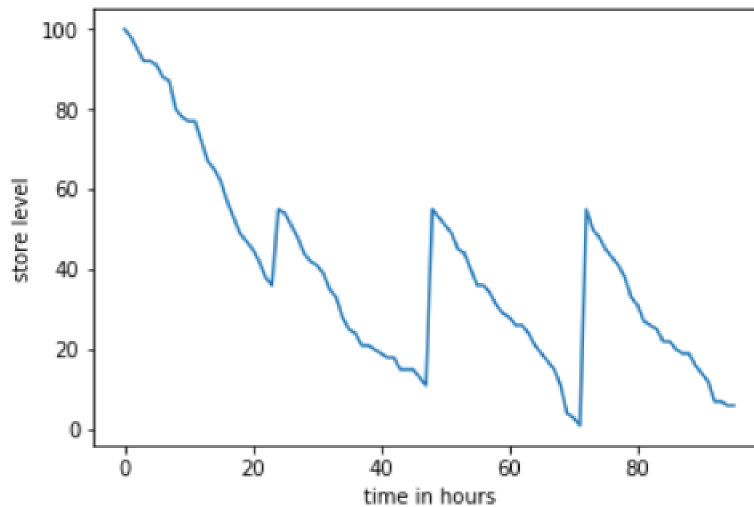

Figure 2: Inventory level of store using Order-up-to Ordering Policy

3 Solution using SimPy

3.1 Introduction to SimPy

According to the SimPy website, "SimPy is a process-based discrete-event simulation framework based on standard Python." It is a relatively new package, as it was first released in 2002, and the current version was released in 2016. SimPy is useful for modeling the relationship between events and time, as it enables the user to create an environment which simulates real time. In contrast to standard Python, which utilizes functions, SimPy utilizes something known as **generators**. Whereas functions use the **return** command, generators use the command **yield**. Yielding an event suspends the simulation process, which will resume once the event has occurred. Unlike functions, which can be iterated through repeatedly, generators can be iterated through only once. Thus after a generator runs it is considered to be "empty." This makes generators ideal for saving memory and allowing the user to see step-by-step what is occurring in a process. Consequently, SimPy is ideal for modeling situations characterized by a process of arrivals and departures such as gas stations, intersections and bank counters.

3.2 Modeling Customer Arrivals in SimPy

In this report, customer arrivals are modeled using the Poisson Process. The Poisson Process is a counting process which models random points in space and time. It is useful for scenarios in

which events occur randomly but according to some rate or distribution. Some examples where the Poisson Process is commonly employed as a model include the timing of earthquakes, the location of Internet users on a network, or the rate of occurrence of car accidents in a region. The Poisson Process is a valuable tool for examining the times at which events occur, the time between different events occurring (the inter-event time), and the probability of a certain number of events happening in a given time period. Given an arrival rate r, the probability of having k events occur in a time interval t is given by:

$$P(k \ events \ in \ interval \ t) = e^{-rt}\frac{rt^k}{k!}$$

The Poisson Process relates to several common statistical distributions, including the Poisson distribution, the exponential distribution, and the gamma distribution. In this report, the exponential distribution is used to model customer arrivals. This means that the inter-arrival times follow an exponential distribution with mean $\frac{1}{r}$. r was set to 0.5, which means that on average a customer will arrive at the store every 0.5 hours. Thus approximately 2 customers will arrive every hour, so about 48 customers will arrive every day. Therefore 48 items is our expected daily demand. This number will be influential in determining the optimal ordering policy, as it will affect how many items the store will need to have on hand. For this model each customer purchases only one item at the store. In more complex examples, however, it is common to have another distribution used to model individual customer demand.

4 Computational Study

4.1 Setting up the Simulation

The first step to creating the simulation was defining values for the three costs. The holding cost was set to \$8 /item. The shipping cost was set to \$50+ \$20 /item. The backorder cost was set to \$92 /customer. (Note that these values were drawn from an example in *Applied Probability and Stochastic Processes*; they were not chosen randomly). The store capacity was set to 100 items. Inventory is checked once every 24 hours, and a delivery is then placed (if necessary for the Order-up-to Policy). The simulation runs for 4 days, at the end of which total cost is computed

using the values of the parameters. The goal for the Constant Delivery Ordering Policy was to test every possible delivery size (1-100) to determine which one produced the optimal cost, and the goal for the Order-up-to Ordering Policy was to do the same for every possible delivery threshold. The simulation was run 10,000 times for each value (10,000 is considered a large enough number to produce accurate results). Note that the reason the simulation must be run more than once is because of the randomness of customer arrivals.

4.2 Discussion of Results

The Order-up-to Policy turns out to be the more cost efficient of the two policies. The optimal delivery threshold is 55 items, which produces a total cost of approximately $3573.93 under our parameters. (For the Constant Delivery Policy, the optimal delivery size is 32 items, which produces a total cost of approximately $4042.56). Below is included a graph of the breakdown of shipping, holding, and backorder costs for the Order-up-to Policy. Note that backorder cost comprises only a small percentage of total cost. This, interestingly, is a result of the fact that the per item backorder cost ($92) is very high relative to the per item holding cost ($8). Since having frequent backorder will make the total cost increase significantly, the optimal delivery threshold will be high enough that backorder will not occur often. This is why it makes sense that the optimal delivery threshold (55 items) is higher than the expected daily demand (48 items). If per item backorder cost were to decrease, then the optimal delivery threshold would decrease as well, since individual backorders would contribute less to the total cost. Conversely, if per item holding cost were to increase, then the optimal delivery threshold would also decrease, as it becomes more expensive to store items. Note, however, that if per item holding cost were to decrease, there would not be a significant change in the optimal delivery threshold unless per item shipping cost also decreased. This is because the cost of ordering more items would outweigh the money saved by paying less to store them.

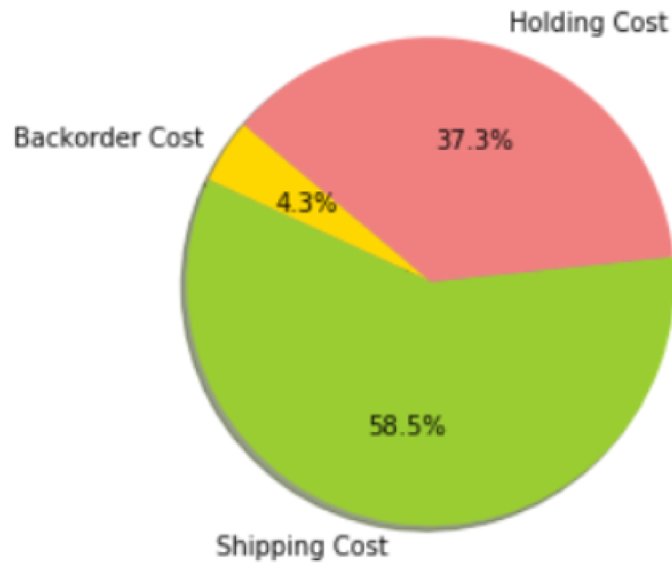

Figure 3: Distribution of Cost

5 Conclusion

The findings of this report, which are that the Order-up-to Ordering Policy is more efficient than the Constant Delivery Ordering Policy, are consistent with other work that has been done in the field of inventory management. The code corresponding to each model has been attached in the Appendix. These models did not take into account delivery time, variable customer demand, changes over time to holding, shipping, and backorder costs, multiple types of items being sold, or multiple retail outlets. These are some of the things which I would like to incorporate into my models in the future. I would like to thank Professor Anh Ninh for working with me on this project.

6 Connections to other projects

I learned a lot from listening to all the different presentations. From blackjack to music software to athletic training, the students in our class covered a variety of topics. There are a few ways in which I think I could apply concepts from other projects to my own. The presentation about matching students with their advisors got me thinking about how if I was modelling more than one type of good, I could use a similar weighted system to determine which goods are most important to the store and therefore need to be prioritized for delivery. If I was trying to figure out optimal

delivery routes to minimize cost, then perhaps I could draw on Sarah or Will's work. I am sure that Bayesian theory also has an application in my project, although I am unsure of how exactly it would fit in.

References

[1] Nada Sanders and CSCMP. "Operations Management Defined," *Informit*. Jan. 2014. http://www.informit.com/articles/article.aspx?p=2167438.

[2] Siegrist, Kyle. "13. The Poisson Process," *Vietnam Draft Lottery Data*. www.randomservices.org/random/poisson/index.html.

[3] “Basic Concepts of the Poisson Process,” *Introduction to Probability*. https://www.probabilitycourse.com/chapter11/11_1_2_basic_concepts_of_the_poisson_process.php.

[4] “Inventory Model & Types,” *What Is Six Sigma? – Certification, Training, Lean*. 2018. www.whatissixsigma.net/inventory-model-types/.

[5] Lebovitz, David. “Taking Inventory of Inventories,” *J.P. Morgan Institutional Asset Management*, JP Morgan Chase and Co., 19 Apr. 2016. am.jpmorgan.com/us/en/asset-management/gim/adv/insights/market-insights/taking-inventory-of-inventories.

[6] Greenspan, Roberta. “Walmart: Inventory Management,” *Panmore Institute*, 25 Mar. 2017. panmore.com/walmart-inventory-management.

[7] Uddin, K. M. Salah, et al. “Modeling and Simulation of an Inventory System - A Case Study of HOMES 71 LTD, Bangladesh,” *Clinical Medicine and Diagnostics, Scientific & Academic Publishing*, 2015. article.sapub.org/10.5923.j.ajor.20150503.03.html.

[8] “Basic Concepts,” *Basic Concepts - SimPy 3.0.11 Documentation*. simpy.readthedocs.io/en/latest/simpy_intro/basic_concepts.html.

A Code for Constant Delivery Ordering Policy Model

```
import random
import simpy
import pandas as pd
import matplotlib.pyplot as plt

SIM_TIME=96 #hours
CAPACITY=100
INITIAL_INVENTORY=100
NUMBER_OF_SHIPMENTS=0
ITEMS_IN_STORAGE=0
negative_counter=0 #keeps track of how many times we have to backlog
cost_tracker=0
backlog_cost_tracker=0
shipping_cost_tracker=0
holding_cost_tracker=0
customer_counter=0
DELIVERY=32 #how much the truck brings with each shipment
NUMBER_OF_SIMULATIONS=10000

Matrix = [[0 for x in range(SIM_TIME)] for y in range(NUMBER_OF_SIMULATIONS)]
inventory_array=[]
big_array=[]

def customer(env, name, store):
    global negative_counter
    if store.level==0:
        negative_counter+=1
        yield store.put(1)
    yield store.get(1)

def customer_arrivals(env, store):
    global customer_counter
    global negative_counter
    while True:
        yield env.timeout(random.expovariate(1 / 0.5))
        env.process(customer(env, 'Customer', store))
        customer_counter+=1

def inventory_checker(env, store):
    global negative_counter
    global NUMBER_OF_SHIPMENTS
    global ITEMS_IN_STORAGE
```

```
    while True:
        yield env.timeout(24)  #move this to the end if we want a delivery at time t=0
        yield store.put(DELIVERY)
        NUMBER_OF_SHIPMENTS+=1
        ITEMS_IN_STORAGE+=store.level

def cost_checker(env, store):
    return backlog_cost(env, store)+shipping_cost(env, store)+holding_cost(env, store)

def backlog_cost(env, store):
    return 92*negative_counter

def shipping_cost(env, store):
    return 50*NUMBER_OF_SHIPMENTS+20*NUMBER_OF_SHIPMENTS*DELIVERY

def holding_cost(env, store):
    return 8*ITEMS_IN_STORAGE

def inventory_array_generator(env, store):
    global inventory_array
    while True:
        inventory_array.append(store.level)
        yield env.timeout(1)

print('Inventory example')
final=[]
for i in range(NUMBER_OF_SIMULATIONS):
    env = simpy.Environment()
    store = simpy.Container(env, CAPACITY)
    store.put(INITIAL_INVENTORY)
    env.process(customer_arrivals(env, store))
    env.process(inventory_checker(env, store))
    env.process(inventory_array_generator(env, store))
    env.run(until=SIM_TIME)
    cost_tracker+=cost_checker(env, store)
    backlog_cost_tracker+=backlog_cost(env, store)
    shipping_cost_tracker+=shipping_cost(env, store)
    holding_cost_tracker+=holding_cost(env, store)
    negative_counter=0
    NUMBER_OF_SHIPMENTS=0
    ITEMS_IN_STORAGE=0
    Matrix[i]=inventory_array
    inventory_array=[]
```

```
final.append([DELIVERY, cost_tracker/NUMBER_OF_SIMULATIONS,
   backlog_cost_tracker/NUMBER_OF_SIMULATIONS, shipping_cost_tracker/NUMBER_OF_SIMULATIONS,
   holding_cost_tracker/NUMBER_OF_SIMULATIONS])

print('Expected cost for initial inventory of 100 with '
   +str(DELIVERY) +' items delivered per day is '+str(cost_tracker/NUMBER_OF_SIMULATIONS))
print('Expected backlog cost for initial inventory of 100 with '
   +str(DELIVERY) +' items delivered per day is '+str(backlog_cost_tracker/NUMBER_OF_SIMULATIONS))
print('Expected shipping cost for initial inventory of 100 with '
   +str(DELIVERY) +' items delivered per day is '+str(shipping_cost_tracker/NUMBER_OF_SIMULATIONS))
print('Expected holding cost for initial inventory of 100 with '
   +str(DELIVERY) +' items delivered per day is '+str(holding_cost_tracker/NUMBER_OF_SIMULATIONS))
print('Expected number of customers is ' +str(customer_counter/(NUMBER_OF_SIMULATIONS*11)))
print(final)

placeholder=0
for hour in range(SIM_TIME):
    for simulation in range(NUMBER_OF_SIMULATIONS):
        placeholder+=Matrix[simulation][hour]
    placeholder=placeholder/NUMBER_OF_SIMULATIONS
    big_array.append(placeholder)
    placeholder=0
print(big_array)

plt.plot(big_array)
plt.xlabel('time in hours')
plt.ylabel('store level')
plt.show()

labels = 'Shipping Cost', 'Holding Cost', 'Backorder Cost'
sizes = [shipping_cost_tracker/NUMBER_OF_SIMULATIONS,
   holding_cost_tracker/NUMBER_OF_SIMULATIONS, backlog_cost_tracker/NUMBER_OF_SIMULATIONS]
colors = ['gold', 'yellowgreen', 'lightcoral']
explode = (0, 0, 0)  # explode 1st slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()

#ourresult = pd.DataFrame(final)
#ourresult.to_csv('method1final.csv', index=True, header=False)
```

B Code for Order-up-to Ordering Policy Model

```
import random
import simpy
import pandas as pd
import matplotlib.pyplot as plt

SIM_TIME=96 #hours
CAPACITY=100
DELIVERY=0
negative_counter=0 #keeps track of how many times we have to backlog
NUMBER_OF_SHIPMENTS=0
ITEMS_IN_STORAGE=0
cost_tracker=0
backlog_cost_tracker=0
shipping_cost_tracker=0
holding_cost_tracker=0
customer_counter=0
NUMBER_OF_SIMULATIONS=10000
time_between_delivery=24
DESIRED_AMOUNT=55
INITIAL_INVENTORY=100

Matrix = [[0 for x in range(SIM_TIME)] for y in range(NUMBER_OF_SIMULATIONS)]
inventory_array=[]
big_array=[]

def customer(env, name, store):
    global negative_counter
    if store.level==0:
        negative_counter+=1
        yield store.put(1)
    yield store.get(1)

def customer_arrivals(env, store):
    global customer_counter
    global negative_counter
    while True:
        yield env.timeout(random.expovariate(1 / 0.5))
        env.process(customer(env, 'Customer', store))
        customer_counter+=1

def delivery(env, store):
    global DESIRED_AMOUNT
```

```
    global NUMBER_OF_SHIPMENTS
    global DELIVERY
    global ITEMS_IN_STORAGE
    global customer_counter
    global time_between_delivery
    while True:
        yield env.timeout(time_between_delivery)
        if store.level<DESIRED_AMOUNT:
            fill_amount=DESIRED_AMOUNT-store.level
            yield store.put(fill_amount)
            NUMBER_OF_SHIPMENTS+=1
            DELIVERY+=fill_amount
            fill_amount=0
        ITEMS_IN_STORAGE+=store.level

def cost_checker(env, store):
    return backlog_cost(env, store)+shipping_cost(env, store)+holding_cost(env, store)

def backlog_cost(env, store):
    return 92*negative_counter

def shipping_cost(env, store):
    return 50*NUMBER_OF_SHIPMENTS+20*DELIVERY

def holding_cost(env, store):
    return 8*ITEMS_IN_STORAGE

def inventory_array_generator(env, store):
    global inventory_array
    while True:
        inventory_array.append(store.level)
        yield env.timeout(1)

print('Inventory example')
final=[]
for i in range(NUMBER_OF_SIMULATIONS):
    env = simpy.Environment()
    store = simpy.Container(env, CAPACITY)
    store.put(INITIAL_INVENTORY)
    env.process(customer_arrivals(env, store))
    env.process(delivery(env, store))
    env.process(inventory_array_generator(env, store))
    env.run(until=SIM_TIME)
    cost_tracker+=cost_checker(env, store)
```

```
        backlog_cost_tracker+=backlog_cost(env, store)
        shipping_cost_tracker+=shipping_cost(env, store)
        holding_cost_tracker+=holding_cost(env, store)
        negative_counter=0
        NUMBER_OF_SHIPMENTS=0
        ITEMS_IN_STORAGE=0
        DELIVERY=0
        Matrix[i]=inventory_array
        inventory_array=[]

    final.append([DESIRED_AMOUNT, cost_tracker/NUMBER_OF_SIMULATIONS,
      backlog_cost_tracker/NUMBER_OF_SIMULATIONS, shipping_cost_tracker/NUMBER_OF_SIMULATIONS,
      holding_cost_tracker/NUMBER_OF_SIMULATIONS])

    print('Expected cost for initial inventory of 100 with delivery threshold '
      +str(DESIRED_AMOUNT)+' and 24 hours between delivery is '
      +str(cost_tracker/NUMBER_OF_SIMULATIONS))
    print('Expected backlog cost for initial inventory of 100 with delivery threshold '
      +str(DESIRED_AMOUNT)+' and 24 hours between delivery is '
      +str(backlog_cost_tracker/NUMBER_OF_SIMULATIONS))
    print('Expected shipping cost for initial inventory of 100 with delivery threshold '
      +str(DESIRED_AMOUNT)+' and 24 hours between delivery is '
      +str(shipping_cost_tracker/NUMBER_OF_SIMULATIONS))
    print('Expected holding cost for initial inventory of 100 with delivery threshold '
      +str(DESIRED_AMOUNT)+' and 24 hours between delivery is '
      +str(holding_cost_tracker/NUMBER_OF_SIMULATIONS))
    print('Expected number of customers is '+str(customer_counter/(4*NUMBER_OF_SIMULATIONS)))

    print(final)

    placeholder=0
    for hour in range(SIM_TIME):
        for simulation in range(NUMBER_OF_SIMULATIONS):
            placeholder+=Matrix[simulation][hour]
        placeholder=placeholder/NUMBER_OF_SIMULATIONS
        big_array.append(placeholder)
        placeholder=0
    print(big_array)

    plt.plot(big_array)
    plt.xlabel('time in hours')
    plt.ylabel('store level')
    plt.show()
```

```
labels = 'Shipping Cost', 'Holding Cost', 'Backorder Cost'
sizes = [shipping_cost_tracker/NUMBER_OF_SIMULATIONS,
   holding_cost_tracker/NUMBER_OF_SIMULATIONS, backlog_cost_tracker/NUMBER_OF_SIMULATIONS]
colors = ['gold', 'yellowgreen', 'lightcoral']
explode = (0, 0, 0)  # explode 1st slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()

#ourresult = pd.DataFrame(final)
#ourresult.to_csv('0inventoryfinal.csv', index=True, header=False)
```