



Report for Program 2 - Simulation

COURSE: SIMULATION(20564)

KEIVAN JAMALI 403209743
PROFESSOR SHAFABI

April 17, 2025

Contents

1	Introduction	2
2	Problem Definition	2
3	Simulation Methodology	4
4	Code Structure and Implementation	5
4.1	Tools.py	5
4.2	number_generator.py	5
4.3	Engine.py	5
4.4	main.py	6
5	Experiments and Results	7
5.1	Part a: Simulation for Two Weeks with ($S = 175, s = 25$)	7
5.2	Part b: Comparing Two Policies over a Year	8
5.3	Part c: Discussion and Suggestions	9
6	Test Cases	11
6.1	Individual Test Cases and Results	11

List of Figures

1	Storage level over time for policy ($S = 175, s = 25$) during 2-week simulation	7
2	Storage level for policy ($S = 175, s = 25$) over one year	8
3	Storage level for policy ($S = 100, s = 50$) over one year	9
4	TestCase 1 results	11
5	TestCase 2 results	12
6	TestCase 3 results	12
7	TestCase 4 results	13
8	TestCase 5 results	13
9	TestCase 6 results	14

1 Introduction

Inventory management is a critical component of supply chain and operations planning. Among the various inventory policies, the (S, s) policy is widely used due to its balance between simplicity and efficiency. In this policy, inventory is reviewed periodically, and an order is placed to replenish the stock up to level S whenever the inventory level falls below s .

The purpose of this report is to simulate the behavior of an inventory system governed by an (S, s) policy. Using a custom-built program, the system is analyzed under specific cost parameters and inventory settings to understand the cost implications of holding stock, placing orders, and facing shortages. The simulation provides insights into system performance over both short-term (2-week) and long-term (1-year) periods.

This report not only presents the simulation results but also compares the effectiveness of two different (S, s) configurations over a one-year period. Finally, the study discusses whether the results are reliable and offers suggestions for further improvement of the model or the policy.

2 Problem Definition

The goal of this project is to simulate and evaluate an (S, s) inventory control policy under specific cost conditions. The policy works by monitoring inventory levels weekly. If the inventory falls below a predefined threshold s , an order is placed to replenish the stock up to the upper limit S .

The simulation considers the following cost structure:

- **Fixed cost per order:** \$15
- **Variable cost per order:** \$2 per ton ordered
- **Annual holding cost:** \$5.2 per ton of inventory
- **Annual shortage cost:** \$520 per ton of unsatisfied demand

The program simulates inventory levels and calculates the following performance metrics:

- **Mean yearly holding cost:** Average cost to keep inventory in stock over the year.
- **Mean yearly shortage cost:** Average cost due to inventory shortages.
- **Mean yearly ordering cost:** Includes both fixed and variable ordering costs.
- **Mean total yearly cost:** Sum of all inventory-related costs.

- **Percentage of weeks with shortage:** Fraction of weeks in which at least one shortage occurred.

The performance of two different (S, s) pairs is evaluated: $(175, 25)$ and $(100, 50)$. The goal is to determine which configuration results in lower total cost and better service level.

3 Simulation Methodology

The simulation models a single-item inventory system managed under an (S, s) policy with weekly review periods. The goal is to simulate system behavior over time and estimate the associated costs based on the defined cost structure.

Inventory Dynamics

At the start of each week, the current inventory level is checked. If the level is below the reorder point s , an order is placed to bring the stock level up to the target S . Demand for each week is generated randomly according to a predefined distribution (e.g., uniform, normal, or empirical), and the inventory is updated accordingly. If demand exceeds available inventory, the excess is recorded as a shortage.

Assumptions

The following assumptions are made in the simulation:

- Inventory is reviewed at the beginning of each week.
- Orders are received with a delay.
- Demand is generated stochastically each week.
- There is no maximum capacity constraint in storage or ordering.
- Costs are accrued continuously, but results are reported on a yearly basis.

Performance Metrics

For each simulation run, the following metrics are computed:

- **Holding cost:** Calculated based on the average inventory over time.
- **Shortage cost:** Based on the cumulative unmet demand throughout the simulation.
- **Ordering cost:** Includes both a fixed cost per order and a variable cost proportional to the order size.
- **Total cost:** Sum of holding, shortage, and ordering costs.
- **Shortage frequency:** Percentage of weeks that shows unmet demand at the end of each week.

Simulation Duration

Two durations are considered:

- A short-term run of 2 weeks using $(S = 175, s = 25)$ to visualize system behavior.
- A long-term run of 52 weeks (1 year) using both $(175, 25)$ and $(100, 50)$ for performance comparison.

4 Code Structure and Implementation

The simulation program is organized into four modular Python files, each responsible for a specific aspect of the inventory simulation. This modular design ensures clarity, reusability, and ease of testing or modification.

4.1 `Tools.py`

This file contains the core `Storage` class, which manages inventory level updates, computes integral metrics such as inventory over time, and tracks cost-related values like total ordering costs and unmet demand penalties.

4.2 `number_generator.py`

This module includes utility classes to generate random variables with controllable seeds. These generators produce:

- Exponentially distributed demand sizes,
- Uniformly distributed demand intervals,
- Uniformly distributed delivery delays.

The specific internal details of the random generation logic are abstracted away and not crucial to the user interface.

4.3 `Engine.py`

This file implements the core of the simulation through the `Storage_Simulator` class. Its main responsibilities include:

- Managing simulation events: demand arrival, order placement, and order fulfillment.
- Applying the (S, s) inventory policy with weekly checks.

- Tracking and computing weekly and yearly performance metrics.
- Generating storage level plots for visualization.

The simulation progresses by selecting the next chronological event among demand, review, and delivery, updating the system accordingly.

4.4 `main.py`

This is the front-end file where all components are assembled and executed. The user provides:

- Time duration (in days),
- Initial storage level,
- (S, s) policy parameters,
- A flag to determine whether to plot results.

The `main` function initializes all generators with fixed seeds and passes them along with the cost structure and policy parameters to the simulation engine. This script provides an easy-to-use interface for running simulations with different policies and durations.

5 Experiments and Results

To evaluate the performance of the (S, s) inventory policy, we conducted several simulation experiments under controlled conditions. The demand is generated based on an exponential distribution, while the intervals between demands and the delay in order arrivals follow uniform distributions. Cost parameters are fixed as defined in the problem.

5.1 Part a: Simulation for Two Weeks with $(S = 175, s = 25)$

In this experiment, the storage policy was set with a capacity $S = 175$ and a reorder point $s = 25$. The simulation was run for a duration of two weeks (14 days). The purpose of this short simulation was to visualize the storage behavior and verify that the simulator behaves as expected.

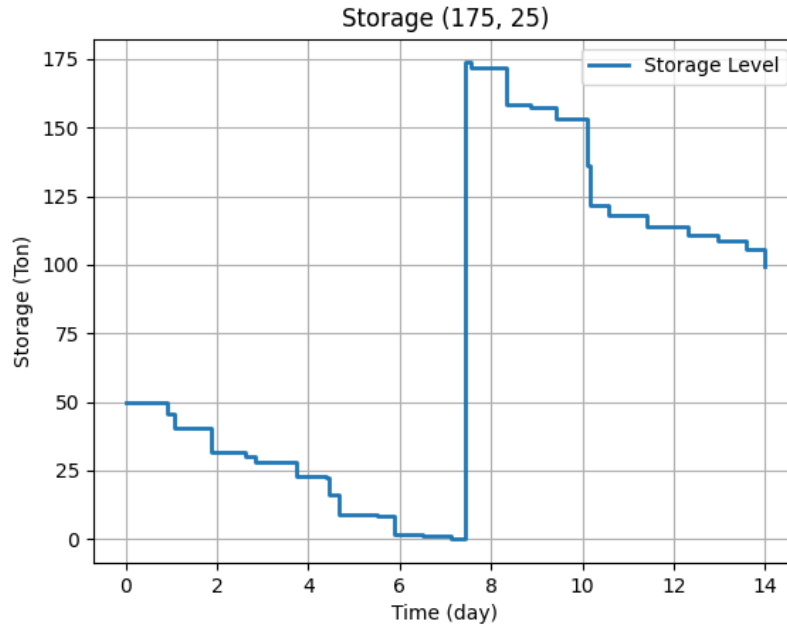


Figure 1: Storage level over time for policy $(S = 175, s = 25)$ during 2-week simulation

The results of the simulation are summarized below:

- **Mean Holding Cost per Year:** 390.2
- **Mean Shortage Cost per Year:** 0.8
- **Mean Ordering Cost per Year:** 9425

- **Mean Total Cost per Year:** 9815.9
- **Percentage of Weeks with Shortages:** 0

5.2 Part b: Comparing Two Policies over a Year

To evaluate long-term performance, we simulated two policies over the course of an entire year (365 days):

1. ($S = 175, s = 25$)
2. ($S = 100, s = 50$)

The goal was to determine which policy results in lower average total cost and fewer shortages. The figures below show the storage level over time for each policy:

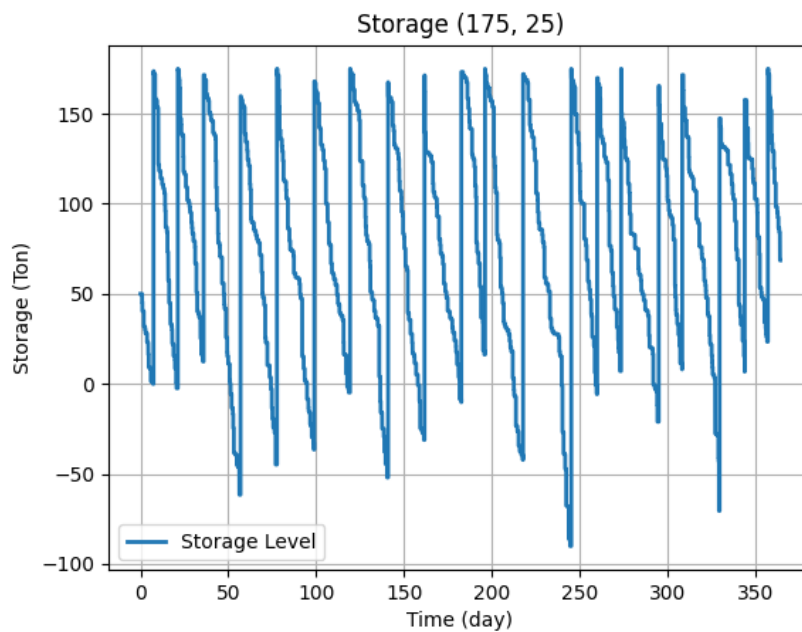


Figure 2: Storage level for policy ($S = 175, s = 25$) over one year

- **Mean Holding Cost per Year:** 401.9
- **Mean Shortage Cost per Year:** 1662.1
- **Mean Ordering Cost per Year:** 7888.4
- **Mean Total Cost per Year:** 9952.4
- **Percentage of Weeks with Shortages:** 25

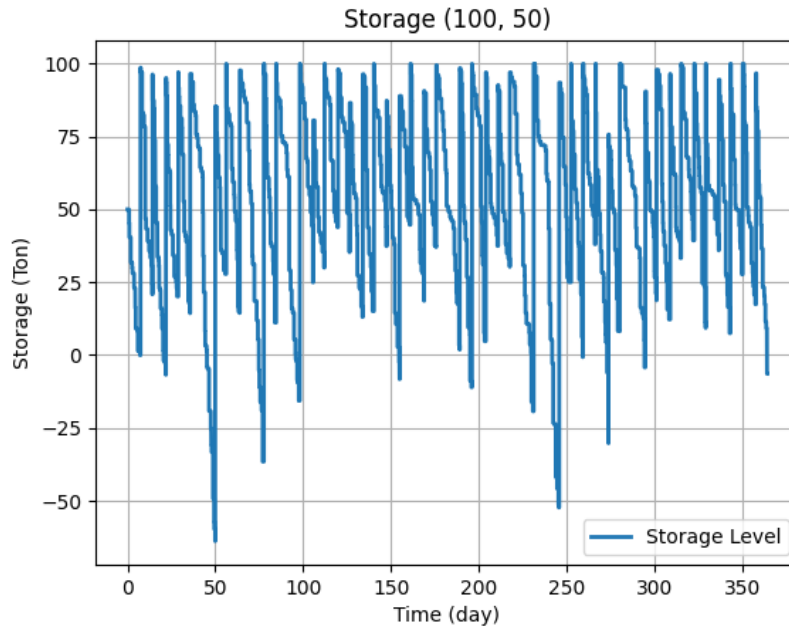


Figure 3: Storage level for policy $(S = 100, s = 50)$ over one year

- **Mean Holding Cost per Year:** 275.1
- **Mean Shortage Cost per Year:** 544.2
- **Mean Ordering Cost per Year:** 8098.4
- **Mean Total Cost per Year:** 8917.8
- **Percentage of Weeks with Shortages:** 19.2

Based on these results, the policy with the lower total cost and/or fewer shortage weeks is considered more efficient. Therefore, The $(100, 50)$ policy is better.

5.3 Part c: Discussion and Suggestions

The simulation provides valuable insights into the inventory system's behavior under different (S, s) policies. The model incorporates stochastic elements such as random demand and delivery delays, which make the outcomes more realistic. Specifically, demand amounts follow an exponential distribution, while the intervals between demands and delivery delays are modeled using uniform distributions.

Despite the usefulness of this setup, there are still some limitations that affect the generalizability of the results:

- The parameters of the exponential and uniform distributions are fixed throughout the simulation and do not capture potential changes over time or seasonality.
- The (S, s) policy thresholds are chosen manually, which may not lead to optimal or cost-efficient outcomes.
- The model does not account for variability in lead time due to external disruptions or supply chain bottlenecks beyond simple uniform randomness.

Suggestions for Improvement:

1. Consider using historical or real-world data to fit more accurate demand and delay distributions.
2. Incorporate time-varying or state-dependent distributions to reflect seasonality or market shifts.
3. Apply optimization algorithms (e.g., grid search, reinforcement learning, or metaheuristics) to determine the best (S, s) values for minimizing total cost.
4. Run the simulation multiple times under each configuration to compute average costs and confidence intervals for more robust decision-making.

Implementing these improvements can significantly increase the model's realism and make it more applicable to actual inventory management scenarios.

6 Test Cases

To evaluate the performance and correctness of the inventory simulation model, six test cases were designed and run. The results below summarize the outputs under various initial conditions and storage policies.

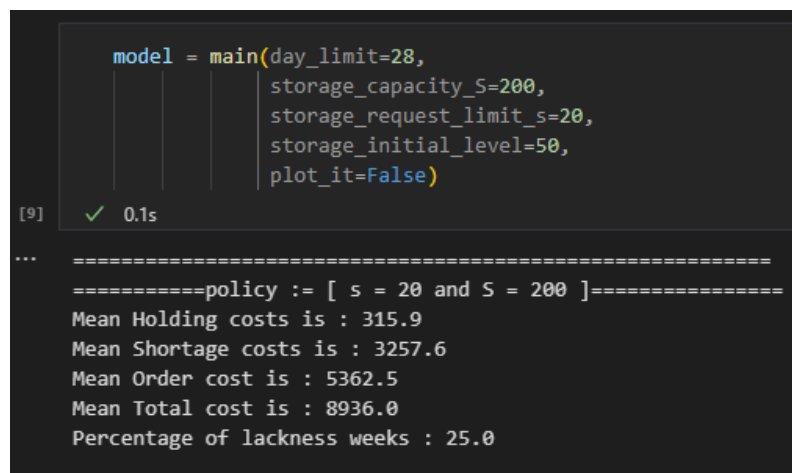
6.1 Individual Test Cases and Results

Each test case is also visualized through storage level plots to confirm behavior over time.

Test Case 1

Inputs:

- Days: 28, Initial Level: 50
- Policy: ($S = 200, s = 20$)



```

model = main(day_limit=28,
              storage_capacity_S=200,
              storage_request_limit_s=20,
              storage_initial_level=50,
              plot_it=False)
[9] ✓ 0.1s
...
=====policy := [ s = 20 and S = 200 ]=====
Mean Holding costs is : 315.9
Mean Shortage costs is : 3257.6
Mean Order cost is : 5362.5
Mean Total cost is : 8936.0
Percentage of lackness weeks : 25.0

```

Figure 4: TestCase 1 results

Test Case 2

Inputs:

- Days: 28, Initial Level: 50
- Policy: ($S = 200, s = 50$)

```

model = main(day_limit=28,
              storage_capacity_S=200,
              storage_request_limit_s=50,
              storage_initial_level=50,
              plot_it=False)
[10] ✓ 0.1s
...
=====policy := [ s = 50 and S = 200 ]=====
Mean Holding costs is : 505.5
Mean Shortage costs is : 0.4
Mean Order cost is : 10170.9
Mean Total cost is : 10676.7
Percentage of lackness weeks : 0.0

```

Figure 5: TestCase 2 results

Test Case 3**Inputs:**

- Days: 70, Initial Level: 50
- Policy: ($S = 100, s = 30$)

```

model = main(day_limit=70,
              storage_capacity_S=100,
              storage_request_limit_s=30,
              storage_initial_level=50,
              plot_it=False)
[11] ✓ 0.1s
...
=====policy := [ s = 30 and S = 100 ]=====
Mean Holding costs is : 253.2
Mean Shortage costs is : 1003.7
Mean Order cost is : 8320.4
Mean Total cost is : 9577.2
Percentage of lackness weeks : 20.0

```

Figure 6: TestCase 3 results

Test Case 4**Inputs:**

- Days: 70, Initial Level: 50

- Policy: ($S = 150, s = 30$)

```

model = main(day_limit=70,
             storage_capacity_S=150,
             storage_request_limit_s=30,
             storage_initial_level=50,
             plot_it=False)
[12] ✓ 0.1s

...
=====policy := [ s = 30 and S = 150 ]=====
Mean Holding costs is : 366.3
Mean Shortage costs is : 525.5
Mean Order cost is : 8606.4
Mean Total cost is : 9498.2
Percentage of lackness weeks : 30.0

```

Figure 7: TestCase 4 results

Test Case 5

Inputs:

- Days: 1400, Initial Level: 100
- Policy: ($S = 100, s = 10$)

```

model = main(day_limit=1400,
             storage_capacity_S=100,
             storage_request_limit_s=10,
             storage_initial_level=100,
             plot_it=False)
[13] ✓ 0.1s

...
=====policy := [ s = 10 and S = 100 ]=====
Mean Holding costs is : 193.8
Mean Shortage costs is : 3531.7
Mean Order cost is : 7625.9
Mean Total cost is : 11351.3
Percentage of lackness weeks : 46.0

```

Figure 8: TestCase 5 results

Test Case 6

Inputs:

Report

Report for Program 2 - Simulation

13

- Days: 1400, Initial Level: 100
- Policy: ($S = 100, s = 20$)

```

model = main(day_limit=1400,
             storage_capacity_S=100,
             storage_request_limit_s=20,
             storage_initial_level=100,
             plot_it=False)
[14] ✓ 0.1s
...
=====policy := [ s = 20 and S = 100 ]=====
Mean Holding costs is : 207.8
Mean Shortage costs is : 2791.9
Mean Order cost is : 7661.0
Mean Total cost is : 10660.6
Percentage of lackness weeks : 42.0

```

Figure 9: TestCase 6 results

References

- Prof.Shafashi, *Simulation slides*, Spring, 2025.
- ChatGPT-o4, *OpenAi Model of o4(free account)*, Spring, 2025.

Appendix

A. Code for the Simulation

The following is the Python code that was used to run the simulation for the Storage system based on the policy chosen (S, s).

```
from modular.number_generator import Exponential_Generator, Uniform_Generator
from modular.Tools import Storage
import matplotlib.pyplot as plt

class Storage_Simulator:
    """Storage Simulator"""

    def __init__(self, demand_generator: Exponential_Generator,
                 demand_interval_generator: Uniform_Generator,
                 delay_generator: Uniform_Generator,
                 fixed_cost: float,
                 variable_cost: float,
                 pos_cost: float,
                 neg_cost: float,
                 initial_level: int) -> None:
        """This function will simulate the storage for you.

        Args:
            demand_interval_generator (Uniform_Generator): Interval between demand.
            delay_generator (Uniform_Generator): The delay after ordering till the goods arrives.
            fixed_cost (float): Fix cost per order.
            variable_cost (float): Variable cost per ton.
            pos_cost (float): Holding cost per ton per year.
            neg_cost (float): Unmet penalty per ton per year.
            initial_level (int): The initial goods that we have in storage.
        """
        self.storage = Storage(initial_level=initial_level)
        self.demand_generator = demand_generator
        self.demand_interval_generator = demand_interval_generator
        self.delay_generator = delay_generator
        self.fixed_cost = fixed_cost
        self.variable_cost = variable_cost
        # convert the yearly cost to dayly cost.
        self.pos_cost = pos_cost / (52 * 7)
        # convert the yearly cost to dayly cost.
        self.neg_cost = neg_cost / (52 * 7)
        # next send request time, next demand time, next arrive goods time, Stop time.
        self.events = [[], [], [], []]
        self.time = 0
        self.negative_ordering_weeks = 0

    def fit(self, time_limit: int, policies: list) -> None:
        """This function will fit the data and run the simulator.

        Args:
            time_limit (int): Time limit in day.
            policies (list): [[S1, s1], [S2, s2], ...]
        """
        self.time_limit = time_limit # This should be days.
        for policy in policies:
            self._reset_generators()
            self.storage.reset()
            # We check the storage every 7 days.
            self._simulate(S=policy[0], s=policy[1], N=7)
            self._log(policy)
```

B. Sample Output

Below is an example output generated by the simulation.

```
=====
=====policy := [ s = 20 and S = 200 ]=====
Mean Holding costs is : 315.9
Mean Shortage costs is : 3257.6
Mean Order cost is : 5362.5
Mean Total cost is : 8936.0
Percentage of lackness weeks : 25.0
```