1 Inventory Models

1.1 Classic (s, S) policy model

A company sells a product (items) which it keeps in inventory (a warehouse for example). Customers make requests one at a time according to Poisson process at rate λ. Each request corresponds to the customer wishing to buy a random number of items (generically denoted by a discrete rv $B \sim G$; $P(B = k) = p(k),\ k \geq 1$). If the inventory level (denoted by I) is large enough, then the customer is sold all B items at a price of $\$r$ each. In general the customer is sold $m = \min\{I, B\} \leq B$ and pays the company $\$rm$, and then the inventory level drops to $I - m$. All demand when $I = 0$ is lost. Whenever the inventory I falls below a pre-specified value $s > 0$, the company places an order for $Y = S - I$ new items that will be delivered $L > 0$ (random) units of time later (L is called the *lead* time). (Only one such order at a time is allowed in progress.) The point is that the company wishes to get the inventory level back up to another pre-specified value $S > s$. An order (of size Y), upon delivery costs $\$c(Y)$ where $c(y),\ y > 0$ is a given function. Also, the company incurs a storage cost of $\$h$ per unit item per unit time: If I items are in inventory for t units of time, then the company incurs a cost of $\$Iht$.

We assume that consecutive demands $\{B_n\}$ are iid $\sim G$, and that consecutive lead times $\{L_n\}$ are iid $\sim H$.

Letting $C_o(t)$ denote the total ordering costs up to time t, C_h the total holding costs up to time t, and $R(t)$ the total revenue earned up to time t, suppose our objective is to simulate a copy (or many copies) of the average rate of profit over the first T time units (how much money per unit time):

$$X = X(T) = \frac{R(T) - C_o(T) - C_h(T)}{T}. \tag{1}$$

For example, T might denote 2 years, and we might wish to estimate $E(X)$ by using Monte-Carlo simulation (requiring us to simulate n copies of X and averaging), or we might be interested in the long-run rate $\lim_{T\to\infty} X(T)$, in which case we would simulate one copy of X for a very large T.

Summary of Notation:

- I: Inventory level (number of items)
- r : price per item bought.
- s: Lower threshold value for inventory (if $I < s$ after a sale, then an order is placed to resupply).
- S: Upper value for inventory.
- $B \sim G$: Number of items desired to buy by a customer request.
- $m = \min\{I, B\}$: The number of desired items (out of the B desired) that are in stock.
- $L \sim H$: Lead time for delivery of a resupply order
- $Y = S - I$: Order size to resupply.

- $c(y)$: Delivery cost of ordering y items
- h: Storage cost per unit item per unit time

Simulation algorithm for a copy of X in (1)

There are only two events: e_0 : a customer request occurs, e_1 : an outstanding delivery arrives. t_A = the time at which the next customer request will occur, and t_o = the time at which the next order is delivered, with t_o set to ∞ whenever an order is not in progress. Note that the amount Y ordered is set to 0 whenever an order is not in progress.

INTIALIZE: $t = C_0 = C_h = R = Y = 0$. $I = S$. $t_o = \infty$. $t_A = -\frac{1}{\lambda}\ln(U)$.

1. **Arrival is next event ($t_A = \min\{t_A, t_o\}$):**

   If $t_A \geq T$, then reset $C_h = C_h + (T - t)hI$, give the output $X = (R - C_o - C_h)/T$ and stop. Otherwise:

   Reset $C_h = C_h + (t_A - t)hI$, $t = t_A$, $t_A = t - \frac{1}{\lambda}\ln(U)$, generate $B \sim G$, set $m = \min\{I, B\}$. Reset $R = R + rm$, $I = I - m$. If $I < s$ and $Y = 0$, then reset $Y = S - I$, generate $L \sim H$, reset $t_o = t + L$.

2. **Delivery is next event ($t_o = \min\{t_A, t_o\}$):** If $t_o \geq T$, then reset $C_h = C_h + (T - t)hI$, give the output $X = (R - C_o - C_h)/T$ and stop. Otherwise:
   Reset $C_h = C_h + (t_o - t)hI$, $C_o = C_o + c(Y)$, $I = I + Y$, $t = t_o$, $t_o = \infty$; $Y = 0$.

Clearly, this model can be generalized in many ways to accommodate different situations/needs. For example, the arrivals could be a non-stationary Poisson process, or a renewal process. We could also allow backlogging, that is, allow the inventory to fall below 0 but with the intention of selling this back log when the stock comes in. There may be costs associated with allowing backlogging, and we can incorporate these as well. We could also allow more than one order to be in progress at the same time, and allow for different "priority " types for ordering (fast/slow, for example, in which more cost is incurred for the fast).