EEE 598 HW1 Fall 2021

Jacob Sindorf

Jacob Sinderf

Homework 1

Assistant Professor Nicolò Michelusi

Office: GWC 330

In your submission, please include:

- printout of Matlab scripts (pdf) that you created and .m files
- printout of figures
- discussion and steps as requested

Please, be clear, concise and organized, and make sure your hand-writing is readable.

I. Markov chain generator

Implement in Matlab the Markov chain generator discussed in class. The generator should have three inputs: number of states (n), transition probability matrix ($\mathbf{P}$), initial distribution (P_0), number of stages simulated (K). The generator then returns a sequence of states

$$[S_0, S_1, \ldots, S_{K-1}].$$

Code logic

Gumbel dist.

$$U_1, \dots U_n \sim U[0,1] \Rightarrow u_i = \text{rand}$$

$$G_i = -\ln(-\ln(u_i)) \quad \forall i \Rightarrow G_i = -\ln(-\ln(\text{rand}))$$

G_i generated every i

initial
($k=0$)

$$S_0 \sim P_0$$

$$S_0 = \underset{i}{\operatorname{argmax}} \; G_i + \ln(P_0(i))$$

↖ gets index

$i \to 1 \dots n$

check all possible states, we max of that to set S_0

$S_{ans} \Rightarrow$ append S_0

$k \geq 1$

$$S_k = \underset{i}{\operatorname{argmax}} \; G_i + \ln[P(i \mid s_{k-1})]$$

$$S_1 = \underset{i}{\operatorname{argmax}} \; G_i + \ln[\underbrace{P(i \mid S_0)}]$$

($i = 1 \dots n$)

check all $P_{i,1}$ from $\mathbf{P}$

$$S_1 = \max(G_i + \ln[\mathbf{P}_{i,1}])$$

$S_{ans} \Rightarrow$ append $S_1, \dots S_k$

Code:

Note: code given with a specific example to show it is working. Uncomment the user input section to change the MC generated to fit any example

```
%Example:
n = 3; P = [.1 .5 .4;.6 .2 .2; .3 .4 .3];
Po = [.7 .2 .1]; K = 7;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Gather user inputs
% n = number of states, P = tranisiton propbability
% matrix, Po = initial distribution
% K = number of stages
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Publish wont work with user input, uncomment to manually type
% prompt1 = 'Number of states (n): ';
% prompt2 = 'Transition Prob matrix (P): ';
% prompt3 = 'Initial distribution (Po): ';
% prompt4 = 'Number of stages (K): ';
%
% n = input(prompt1);P = input(prompt2);
% Po = input(prompt3);K = input(prompt4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               find chain of states
% Uses gumbel dist and max function to generate
% chain of states
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Sans = zeros(K,1)';%initialize state sequence matrix

for k = 1:K
  vals = zeros(n,1)'; %initialize and zero placeholder for max vals
  for i = 1:n   %i goes from 1 to n
    if k == 1 %assume 1 = 0 as matlab has 1 index
    %initial distribution used
        Gi= -log(-log(rand)); %new G value, as G = G(i)
        vals(i) = (Gi + log(Po(i))); %get val of states using initial
 dist
    else
        j = k-1; %index of previous state
        Gi= -log(-log(rand)); %new G value, as G = G(i)
        vals(i) = (Gi + log(P(i,Sans(j)))); %get val of states
    end
  end
    [maxval,Si] = max(vals); %get max values index
    Sans(k) = Si;    %store index as a state
end
fprintf('Final Markov chain generated: ');
Sans - 1  %print 0 index based state as matlab is 1 index based

Final Markov chain generated:
ans =

     1     0     2     1     0     1     0
```

II. *Lazy* INVENTORY MANAGEMENT

A company needs to manage its inventory levels of a certain product in a warehouse. Assume that the control problem operates at discrete stages $k = 0, 1, 2, \dots$. For instance, each stage may represent a month, or a year (depends on the application).

At stage k, let $S_k \in \{0, \dots, M\}$ be the current inventory level at the warehouse, and M be the maximum amount of products that can be stocked at the warehouse.

The company refills its inventory only when its inventory level reaches 0.

During stage k, D_k items are purchased from customers (demand). It follows a probability distribution $P(\cdot)$, independent and identically distributed (i.i.d.) over stages, so that $P(d)$ is the probability that $D_k = d$ items are purchased by costumers in stage k.

The cost for the company to purchase each unit is c. The revenue to the company for each unit sold is r. However, due to finite inventory, some of the requests may not be met. In this case, each unit of unsatisfied requests incur a penalty of p. Finally, the cost of maintaining the inventory is m per unit per stage.

Assumptions on the sequence of events:

1 - The stock level is S_k at the beginning of stage k

2 - Then, the maintenance cost is incurred

3 - Then, U_k new units are purchased, if necessary (based on the policy outlined earlier)

4 - Then, D_k units of demand occur. We assume that D_k is uniform in $\{0, \dots, M\}$, so that $P(D_k = d) = 1/(M+1),\ \forall d = 0, 1, \dots, M$.

5 - After the demand is processed, stage k terminates and the new one begins.

1) Identify the state and show the state dynamics

State: inventory level

From state space $S : \{0, 1, \dots, M\}$

State Dynamics $s_{k+1} = f(s_k, u_k, D_k)$

$s_{k+1} = f(s_k, D_k)$
$P_k = g(s_k, D_k)$

state $i = 0$

- 0 maintenance incurred
- $U_k = M$ units purchased at cost c
- $D_k = d$ units sold at price r, $P(D_k = d) = \frac{1}{M+1}$
- No chance for penalty as $d > M$ not possible

$-$ $s_{k+1} = f(0, d) = M - d$

state $i \neq 0$

- $M(s_k)$ maintnce incurred
- $U_k = 0$ units purchased
- $D_k = d$ units sold at price r $P(D_k = d) = \frac{1}{M+1}$

$-$ $s_{k+1} = f(i, d) = 0$ $\quad d \geq i$

$d = i$, no penalty

$d > i$, penalty

$-$ $s_{k+1} = f(i, 0, d) = i - d$ $\quad d < i$

2) Write the transition probabilities

$$P(s_{k+1}=j \mid s_k=i) = 0 \qquad \begin{matrix} j>i \\ i\neq 0 \end{matrix}$$

- $$P(s_{k+1}=j \mid s_k=0) = \frac{1}{M+1} \qquad \begin{matrix} j>i \\ i=0 \end{matrix}$$

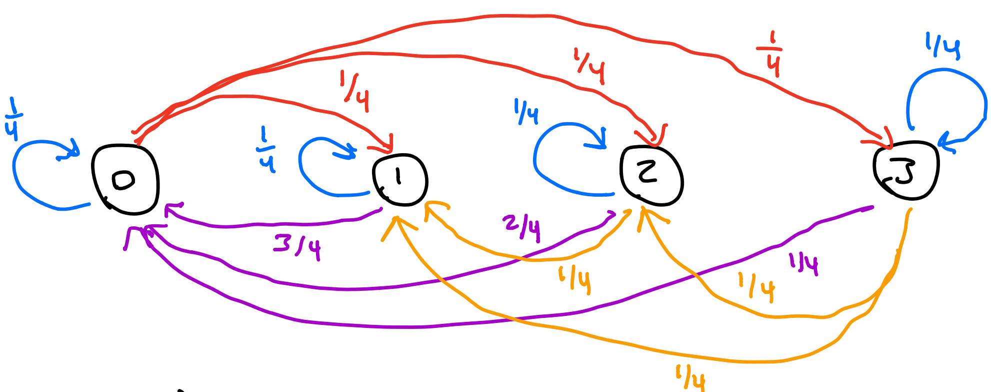
- $$P(s_{k+1}=j \mid s_k=i) = \frac{1}{M+1} \qquad \begin{matrix} j<i \\ j\neq 0 \end{matrix}$$

- Note, these could be combined if $j \leq i, j\neq 0$ unless $i=0$. Split for clarity of cases

- $$P(s_{k+1}=j \mid s_k=i) = \frac{1}{M+1} \qquad j=i \ \forall i \text{ (includes 0)}$$

- $$P(s_{k+1}=0 \mid s_k=i) = \frac{M+1-i}{M+1} \qquad \begin{matrix} j<i \\ i\neq 0 \\ j=0 \end{matrix}$$

3) Draw a graphical representation, for the case $M = 3$



$$P = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \frac{1}{4} & 1/4 & 1/4 & 1/4 \\ 1 & 3/4 & 1/4 & 0 & 0 \\ 2 & 2/4 & 1/4 & 1/4 & 0 \\ 3 & 1/4 & 1/4 & 1/4 & 1/4 \end{array}$$

P_{ij}

4) Compute the steady state distribution in closed form, for the case $M = 3$

(show steps)

For case $M = 3$

(a) $\pi_j = \sum_{l=0}^{M} \pi_l P_{lj}$ + (b) $\sum_{l=0}^{M} \pi_l = 1$

(a)

0) $\pi_0 = \sum_{l=0}^{M} \pi_l P_{lj} \Rightarrow \pi_0 P_{00} + \pi_1 P_{10} + \pi_2 P_{20} + \pi_3 P_{30}$

1) $\pi_1 = \pi_0 P_{01} + \pi_1 P_{11} + \pi_2 P_{21} + \pi_3 P_{31}$

2) $\pi_2 = \pi_0 P_{02} + \cancel{\pi_1 P_{12}}^{0} + \pi_2 P_{22} + \pi_3 P_{32}$

$\pi_2 = \pi_0 P_{02} + \pi_2 P_{22}$

3) $\pi_3 = \pi_0 P_{03} + \cancel{\pi_1 P_{13}}^{0} + \cancel{\pi_2 P_{23}}^{0} + \pi_3 P_{33}$

$\pi_3 = \pi_0 P_{03} + \pi_3 P_{33}$

(b) $\sum_{l=0}^{M} \pi_l = 1$ $\quad \pi_0 + \pi_1 + \pi_2 + \pi_3 = 1$

solve system of equations for $\pi_0 \dots \pi_3$

$$\pi_0 = \pi_0\left(\tfrac{1}{4}\right) + \pi_1\left(\tfrac{3}{4}\right) + \pi_2\left(\tfrac{2}{4}\right) + \pi_3\left(\tfrac{1}{4}\right)$$

$$\pi_1 = \pi_0\left(\tfrac{1}{4}\right) + \pi_1\left(\tfrac{1}{4}\right) + \pi_2\left(\tfrac{1}{4}\right) + \pi_3\left(\tfrac{1}{4}\right)$$

$$\pi_2 = \pi_0\left(\tfrac{1}{4}\right) + \pi_2\left(\tfrac{1}{4}\right) + \pi_3\left(\tfrac{1}{4}\right)$$

$$\pi_3 = \pi_0\left(\tfrac{1}{4}\right) + \pi_3\left(\tfrac{1}{4}\right)$$

$\Rightarrow$

System of eq^n

1. $3\pi_0 = 3\pi_1 + 2\pi_2 + \pi_3$
2. $3\pi_1 = \pi_0 + \pi_2 + \pi_3$
3. $3\pi_2 = \pi_0 + \pi_3$
4. $3\pi_3 = 1\pi_0$
5. $\pi_0 + \pi_1 + \pi_2 + \pi_3 = 1$

4. $\pi_0 = 3\pi_3$

4. $\rightarrow$ 3.

$$3\pi_2 = 3\pi_3 + \pi_3$$

$$\pi_2 = \frac{4}{3}\pi_3$$

$3 \rightarrow 2$

$$3\pi_1 = 3\pi_3 + \frac{4}{3}\pi_3 + \pi_3$$

$$\pi_1 = \frac{1}{3}\left(\frac{9}{3} + \frac{4}{3} + \frac{3}{3}\right)\pi_3$$

$$\pi_1 = \frac{16}{9}\pi_3$$

into 5.

$$3\pi_3 + \frac{16}{9}\pi_3 + \frac{4}{3}\pi_3 + \pi_3 = 1$$

$\pi_3 = 9/64$

$\pi_0 = 27/64$

$\pi_1 = 1/4$

$\pi_2 = 3/16$

$P_k = g(S_k, D_k)$ in i, $\bar{P}(i) = \mathbb{E}\left(g(S_k, D_k) \mid S = i\right)$

$= \sum_j \mathbb{P}(D=j) \cdot g(i,j)$

5) Compute the expected profit in each state, for the case $M = 3$ (show steps)

a) Solve all $R(i,d)$, general

$R(i,d) = R_m + R_p + R_d$

$R_{\text{maintenance}}$: $-m(S_k)$

$R_{\text{purchase}}(S_k = 0)$: $-Mc$ (zero $\hat{S}_k$)

R_{demand}: $\begin{cases} d \cdot r & \text{if } d \leq \hat{S}_k \text{ or } S_k \\ S_k r - p(d - S_k) & \end{cases}$

$d \leq M$

$R(0,d) = -m(0) - Mc + dr - p(0)$

$= -3c + dr \qquad d = (0,1,2,3)$

$R(1,d) = -m + \begin{cases} dr & d = (1,0) \\ 1(r) - p(d-1) & d = (2,3) \end{cases}$

$R(2,d) = -2m + \begin{cases} dr & d = (0,1,2) \\ 2(r) - p(d-2) & d = 3 \end{cases}$

$R(3,d) = -3m + dr \qquad d = (0,1,2,3)$

$0 \underline{\quad} M$

$1 \rightarrow M+1$

b) solve $\bar{R}(i)$

$$\bar{R}(i) = \mathbb{E}[R(i,D_k)] = \sum_{d=0}^{M} \underbrace{\mathbb{P}(D_k = d)}_{\frac{1}{M+1},\ M=3} \cdot R(i,d)$$

$$\bar{R}(i) = \frac{1}{4} \sum_{d=0}^{M} R(i,d)$$

$$\bar{R}(0) = \frac{1}{4}\Big[\overbrace{\overset{0}{-3c+0r}}^{R(0,0)} \overbrace{- \overset{1}{3c+r} - \overset{2}{3c+2r} - \overset{3}{3c+3r}}^{R(0,(1,2,3))}\Big]$$

$$\bar{R}(0) = -3c + \frac{6}{4}r$$

$$\bar{R}(1) = \frac{1}{4}\Big[\overset{0}{-m+0r} \overset{1}{-m+r} \underbrace{\overset{2}{-m+(r-p)}}_{} \overset{3}{-m+(r-2p)}\Big]$$

$$\bar{R}(1) = -m + \frac{3}{4}r - \frac{3}{4}p$$

$$\bar{R}(2) = \frac{1}{4}\Big[-2m+0r - 2m+r - 2m+2r - 2m+2r-p\Big]$$

$$\bar{R}(2) = -2m + \frac{5}{4}r - \frac{1}{4}p$$

$$\bar{R}(3) = \frac{1}{4}\Big[-3m+0r - 3m+r - 3m+2r - 3m+3r\Big]$$

$$\bar{R}(3) = -3m + \frac{6}{4}r$$

6) Compute the average long-term profit per stage, for the case $M = 3$, $c = 1$, $r = 2$, $p = 1$, $m = 0.5$ (show steps)

$$\lim_{k \to \infty} \mathbb{E}\left[\frac{1}{k} \sum_{t=0}^{k-1} R(s_t) \,\middle|\, s_0 \sim \rho_0 \right] = \sum_{j=0}^{M} \underbrace{\pi_j}_{\#4} \underbrace{\bar{R}(j)}_{\#5} \quad \text{—} \; a$$

from 4

$\pi_3 = 9/64$

$\pi_0 = 27/64$

$\pi_1 = 1/4$

$\pi_2 = 3/16$

from 5

$\bar{R}(0) = -3c + \frac{6}{4} r = 0$

$-3(1) + \frac{6}{4}(2)$

$\bar{R}(1) = -m + \frac{3}{4} r - \frac{3}{4} p = \frac{1}{4}$

$-.5 + \frac{3}{4}(2) - \frac{3}{4}(1)$

$\bar{R}(2) = -2m + \frac{5}{4} r - \frac{1}{4} p = \frac{5}{4}$

$-2(.5) + 5/4(2) - \frac{1}{4}(1)$

$\bar{R}(3) = -3m + \frac{6}{4} r = 3/2$

$-3(.5) + \frac{6}{4}(2)$

using a

$$\sum_{0}^{3} = \pi_0 \bar{R}(0) + \pi_1 \bar{R}(1) + \pi_2 \bar{R}(2) + \pi_3 \bar{R}(3)$$

$$= \frac{27}{64}(0) + \frac{1}{4}\left(\frac{1}{4}\right) + 3/16\left(\frac{5}{4}\right) + \frac{9}{64}\left(\frac{3}{2}\right) = .5078125$$

$$\approx .5 \text{ or } \frac{1}{2}$$

And now do the following with Matlab, for a scenario with $M = 20$, $c = 1$, $r = 2$, $p = 1$, $m = 0.5$ (I suggest to keep these as input parameters, since it helps later on):

1) Implement an algorithm to compute the expected profit per stage over K stages, starting from an empty warehouse:

$$\bar{P}_K \triangleq \frac{1}{K}\mathbb{E}\left[\sum_{k=0}^{K-1} p_k + p_K^{\text{term}} | S_0 = 0\right],$$

where p_k is the profit at the kth stage, and p_K^{term} is a terminal profit for the unsold units, assumed as $p_K^{\text{term}} = r \cdot S_K/2$ (in other words, all unsold units at stage K are sold at half price). Note: as an intermediary step, it may help to compute the expected profit in state $S_k = s$ as $\bar{p}(s) = \mathbb{E}[p_k | S_k = s]$.

EQN set up:

$$\bar{P}_K \triangleq \frac{1}{K}\mathbb{E}\left[\sum_{k=0}^{K-1} p_k + p_K^{\text{term}} | S_0 = 0\right] = \frac{1}{K}\left[\left[\sum_{k=0}^{K-1}\sum_{j=0}^{M} \mathbf{P}_{0j}^{K}\, \bar{p}(j)\right] + \sum_{j=0}^{M} \mathbf{P}_{0j}^{K}\, p_{K}^{\text{tern}}(j)\right]$$

where $\mathbf{P}$ = transition Prob. Matrix

$$\bar{p}(j) = \sum_{d=0}^{M} \mathbb{P}(D_k = d)\cdot \text{cost}(j,d)$$

$$= \frac{1}{M+1}\sum_{d=0}^{M} \text{cost}(j,d)$$

$$p_{K}^{\text{term}}(j) = \frac{r(j)}{2}$$

Assume at $K = 0$

$$\bar{P}_0 = 0$$

2) Simulate the system: generate a single sequence of states $\{S_0, \dots, S_{1000}\}$ and profits $\{p_0, \dots, p_{1000}\}$, starting from $S_0 = 0$, using the state dynamics defined earlier, and compute

$$\hat{P}_K = \frac{1}{K}\sum_{k=0}^{K-1} p_k + p_K^{\text{term}}, \ \forall K = 0, \dots, 1000.$$

Note: p_k should NOT be confused with the expected profit in state s, $\bar{p}(s)$, defined above! In fact, p_k is a *realization*, whereas $\bar{p}(s)$ is an expectation with respect to the demand.

EQN set up:

- generate sequence of demand $U = \text{randi}[0, M]$ for $\underline{K}$ stages
- generate sequence of states based on demand & state dynamics
- Generate sequence of costs based on state & demand
- Calculate $\hat{P}_K$ using $\underline{K}$ sequence of costs
- $P_K^{\text{term}} = \frac{r(S_K)}{2}$ (state at $\underline{K}$ from sequence of states)

3) Compute the average long-term profit per stage,

$$\bar{P}_\infty \triangleq \lim_{K\to\infty} \frac{1}{K}\mathbb{E}\left[\sum_{k=0}^{K-1} p_k | S_0 = 0\right],$$

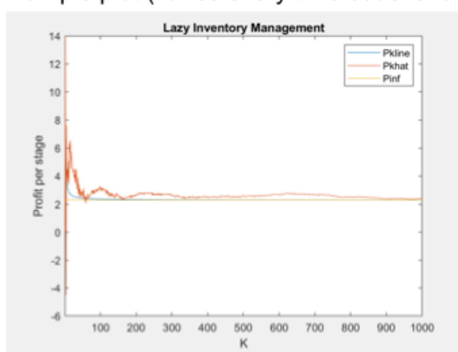
EQN set up:

- from SS of $\bar{P}_\infty$, we simplify to $\sum_{i=0}^{M} \pi(i)\bar{P}(i)$
- $\bar{P}(i)$ calculated in 1
- $\pi(i) = \mathbf{1}^T \cdot [W \cdot W^T]^{-1}$, where $W = [\mathbf{I} - \mathbf{P}; \mathbf{1}]$

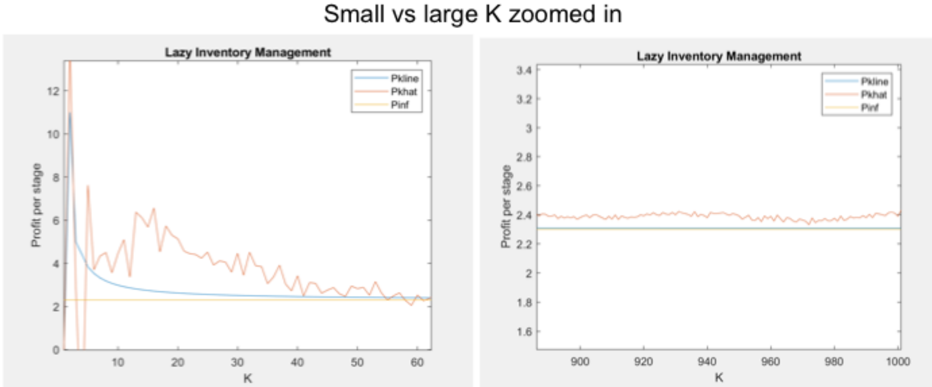
4) Plot $\bar{P}_K$ and $\hat{P}_K$ versus $K = 0, 1, \ldots, 1000$, and $\bar{P}_\infty$ (as a line that spans the entire range of K). Discuss on what you observe.

After plotting all three values, we see that eventually, they all converge to around the same value of around 2.3 shown by the steady-state line. Given the randomness of the simulated costs, the final plot for Pkhat changes each time the code is ran. However, it still converges to around the same point. It is noticeable that the simulated data is noisier in comparison to the expected value. The expected data has a large spike in the beginning but settles very quickly to the final value. Overall the profit value being so low could be caused by the lazy inventory refill, which we will try to optimize in part III.

Example plot (varies every time code is ran)



Small vs large K zoomed in



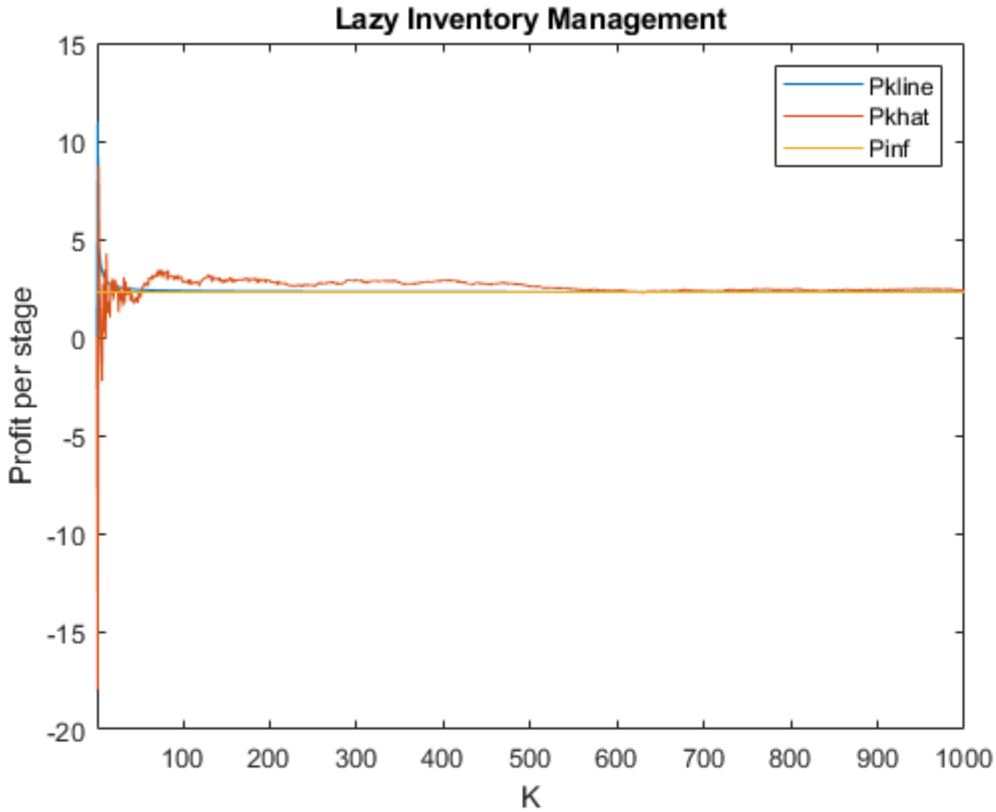
Code for part II

```
clear all; clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%constants
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M = 20; c = 1; r = 2; p = 1; m = .5;
So = 0;
Prob = 1/(M+1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%gather number of stages, K
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Publish does not work with input, uncommet for manual entry
% prompt1 = 'Number of stages (K): ';
% K = input(prompt1);
K = 1000;
%Note:Matlab is 1 index, so I go 1 to K
%This is same as 0 to K-1 in sum terms
Kmat = [1:K+1]; %needed for plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate transition prob matrix, P
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Probmatrix = zeros(M+1,M+1); %initalize
%Note matlab is 1 index, so M+1 needed
%treat state 1 as state 0
for i = 1:M+1     %from i (row), to j (column)
    for j = 1:M+1
        if (j > i)
            if i ~= 1
                Probmatrix(i,j) = 0;
            else
                Probmatrix(i,j) = (1/(M+1));
            end
        elseif  (j <= i)
            if j == 1 && i ~=1
                Probmatrix(i,j) = ((M+2-i)/(M+1)); %need +2 per 1
 index notation
            else
               Probmatrix(i,j) = (1/(M+1));
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%compute expected profit per state, p(line)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
costpk = zeros(M+1,M+1); %initialize cost matrix to hold values
```

```
for j = 0:M
    for d = 0:M    %find cost based on demand d and state j
        if j == 0
            costpk(j+1,d+1) = -M*c + d*r;
        else
            if d > j
                costpk(j+1,d+1) = -j*m + (j*r - p*(d - j));

            else
                costpk(j+1,d+1) = -j*m + d*r;
            end
        end
    end
end
costpksum = sum(costpk,2); %sum each row and multiply by prob
pline = (1/(M+1))*costpksum';
%pline is expected profit per state, presented as a vector

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Pk(line)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Assume Po = 0 as 1/K when K = 0 is invalid
Pkline = zeros(1,K+1);
ksumvals =  zeros(1,K+1);
for k = 2:K+1   %1 index, so 1 to K+1 instead of 0 to K
    sumvals = zeros(1,M+1);
    sumvalsterm = zeros(1,M+1);
    for j = 1:M+1
        P1 = (Probmatrix)^(k-1);%k step prob
        ProbK = P1(1,j);     %get matrix value, its from 0 to j (index
 1)
        sumvals(j) = ProbK*pline(j); %multiply the matrix value with
 pline and save it

        sumvalsterm(j) = ProbK*(r*j/2);
    end
    ksumvals(k) = sum(sumvals);

    Pkline(k) = (1/(k-1))*((sum(ksumvals(1:(k-1))) +
 sum(sumvalsterm)));
%note, due to index = 1 in matlab, we keep all index = k, as this
%coresponds to the proper index, but have to use k-1 for calculation
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate sequence of demands and states
%demands are random integer between 0 and M, iid
%Generate sequence of states based on demand and state dynamics
%keep track of d>s for penalty needed in cost
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Sseq = zeros(1,K+1);  %initialize vector for state sequence
dseq = zeros(1,K+1);  %initialize vector for demand sequence
penalty = zeros(1,K+1); %create penalty placeholder

%initial state = 0,and demand
Sseq(1) = So;
dseq(1) = randi([0 M]);

for i  = 2:K+1
    if Sseq(i-1) == 0
        Skh = M;
        Sseq(i) = Skh - dseq(i-1); %calculate new state based on
 previous
        dseq(i) = randi([0 M]);    %generate new demand
    elseif dseq(i-1) > Sseq(i-1)
        Sseq(i) = 0;               %demand > sequence goes to zero
        dseq(i) = randi([0 M]);    %generate next demand
        penalty(i-1) = 1;          %penalty is incured
    else
        Sseq(i) = Sseq(i-1) - dseq(i-1); %new state
        dseq(i) = randi([0 M]);    %generate new demand
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate sequence of costs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cseq = zeros(1,K+1);  %initialize vector for cost sequence
for i = 1:K+1
%     if Sseq(i) == M
%         pkseq(i) = -m(Sseq(i)) + dseq(i)*r;
    if Sseq(i) == 0
        cseq(i) = -M*c + dseq(i)*r;
    else
       if penalty(i) == 1 %need a penalty based on unmet demand
           cseq(i) = -m*(Sseq(i)) + Sseq(i)*r - p*(dseq(i) - Sseq(i));
       else
           cseq(i) = -m*(Sseq(i)) + dseq(i)*r;
       end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Pk(hat)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sum_j P(D=j)cost(i,j) in state i
%P(Dk = d) = 1/M+1 = Prob
Pkhat = zeros(1,K+1); %1/K cannot do K = 0, so assum P0hat = 0
for k = 2:K+1
   pk1sum = sum(cseq(1:(k-1)));  %costs are based on state at time k
   %see generate sequence of costs. cost based on sequence at k
   pkterm = r*(Sseq(k-1))/2;
   Pkhat(k) = (1/(k-1))*(pk1sum + pkterm);
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Pinf(line)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%uses transition prob matrix and pline calculated earlier
%sum from i=1 to M+1
onematrix = ones(M+1,1);
w = [(eye(M+1,M+1) - Probmatrix),onematrix];
pivals = (onematrix')*inv(w*w');

Pinf = sum(pivals.*pline);
Pinfgraph = Pinf.*(ones(1,K+1));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 plot values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

plot(Kmat,Pkline);hold on; plot(Kmat,Pkhat); hold on;
 plot(Kmat,Pinfgraph);
xlabel('K');
ylabel('Profit per stage');
legend('Pkline','Pkhat','Pinf');
xlim([1,K]);
title('Lazy Inventory Management');
```



Published with MATLAB® R2020b

Note:
due to randomness graph is different than one seen in discussion for $\hat{P}_k$

Part III

1) Identify the states, actions and show the state dynamics

States: $\{0, 1, \ldots, M\}$

Action: $U(i) = \{0 \ldots M-i\} \; \forall i$ ~ action cannot exceed M

$S_{k+1} = f(S_k, U_k, D_k) = \cdot\; S_k + U_k - D_k$

$\cdot$ if $D_k > S_k + U_k$, $S_{k+1} = 0$

2) Write the transition probabilities

i: current state $\quad a$: action

j: next state $\quad d$: demand

$$j > i \begin{cases} \mathbb{P}(j \mid i, a) = 0 & j > (i+a) \\ \mathbb{P}(j \mid i, a) = \dfrac{1}{M+1} & j \leq (i+a) \end{cases}$$

$$j \leq i \begin{cases} \mathbb{P}(j \mid i, a) = \dfrac{1}{M+1} & j \neq 0 \\ \mathbb{P}(j \mid i, a) = \dfrac{M+1-i-a}{M+1} & j = 0 \end{cases}$$

3) Compute the expected profit in a single stage, under each action/state pair

$\bar{C}$ ⇒ expected profit in a single stage under each action/state pair

$$\bar{C}(i,u) = \frac{1}{M+1}\sum_{d=0}^{M} \text{cost}(i,u,d)$$

$$\bar{C}(i, u \in (0, M-i))$$

ex: $\bar{C}(0, u=(0,1,\cdots M))$

Cost ⇒ 1) $S_k = i$ at beginning

2) maintenance $\quad -M(i)$

3) purchase $\mu_k(i) = u \quad -c(u)$

4) demand

new state $=(i+u)$ $\quad d > i+u$ goes to 0

$$+r\underbrace{(i+u)}_{\text{available}} - p(d-\underbrace{(i+u)}_{\text{unmet}})$$

$d \le i+u$

$+r(d)$

See Clinevals.C struct in Matlab for answer.

Organized as follows

Clinevals(i).C(u)

So ex:

state 0, all actions 0...M-i or (0 ... 20)

Clinevals(0).C(u) gives

From Matlab for M=20

c(0,(0...20))
-10.0000 -8.1429 -6.4286 -4.8571 -3.4286 -2.1429 -1.0000 0 0.8571 1.5714
2.1429 2.5714 2.8571 3.0000 3.0000 2.8571 2.5714 2.1429 1.5714 0.8571
0
c(0,(0...19))
-7.6429 -5.9286 -4.3571 -2.9286 -1.6429 -0.5000 0.5000 1.3571 2.0714 2.6429
3.0714 3.3571 3.5000 3.5000 3.3571 3.0714 2.6429 2.0714 1.3571 0.5000
c(0,(0...18))
-5.4286 -3.8571 -2.4286 -1.1429 0 1.0000 1.8571 2.5714 3.1429 3.5714
3.8571 4.0000 4.0000 3.8571 3.5714 3.1429 2.5714 1.8571 1.0000
c(0,(0...17))
-3.3571 -1.9286 -0.6429 0.5000 1.5000 2.3571 3.0714 3.6429 4.0714 4.3571
4.5000 4.5000 4.3571 4.0714 3.6429 3.0714 2.3571 1.5000
c(0,(0...16))
-1.4286 -0.1429 1.0000 2.0000 2.8571 3.5714 4.1429 4.5714 4.8571 5.0000
5.0000 4.8571 4.5714 4.1429 3.5714 2.8571 2.0000
c(0,(0...15))
0.3571 1.5000 2.5000 3.3571 4.0714 4.6429 5.0714 5.3571 5.5000 5.5000
5.3571 5.0714 4.6429 4.0714 3.3571 2.5000
c(0,(0...14))
2.0000 3.0000 3.8571 4.5714 5.1429 5.5714 5.8571 6.0000 6.0000 5.8571
5.5714 5.1429 4.5714 3.8571 3.0000
c(0,(0...13))
3.5000 4.3571 5.0714 5.6429 6.0714 6.3571 6.5000 6.5000 6.3571 6.0714
5.6429 5.0714 4.3571 3.5000
c(0,(0...12))
4.8571 5.5714 6.1429 6.5714 6.8571 7.0000 7.0000 6.8571 6.5714 6.1429
5.5714 4.8571 4.0000
c(0,(0...11))
6.0714 6.6429 7.0714 7.3571 7.5000 7.5000 7.3571 7.0714 6.6429 6.0714
5.3571 4.5000
c(0,(0...10))
7.1429 7.5714 7.8571 8.0000 8.0000 7.8571 7.5714 7.1429 6.5714 5.8571
5.0000
c(0,(0...9))
8.0714 8.3571 8.5000 8.5000 8.3571 8.0714 7.6429 7.0714 6.3571 5.5000
c(0,(0...8))
8.8571 9.0000 9.0000 8.8571 8.5714 8.1429 7.5714 6.8571 6.0000
c(0,(0...7))
9.5000 9.5000 9.3571 9.0714 8.6429 8.0714 7.3571 6.5000
c(0,(0...6))
10.0000 9.8571 9.5714 9.1429 8.5714 7.8571 7.0000

c(0,(0...5))

10.3571 10.0714 9.6429 9.0714 8.3571 7.5000
c(0,(0...4))
10.5714 10.1429 9.5714 8.8571 8.0000
c(0,(0...3))
10.6429 10.0714 9.3571 8.5000
c(0,(0...2))
10.5714 9.8571 9.0000
c(0,(0,1))
10.3571 9.5000
c(0,(0))
10

And now do the following with Matlab, for a scenario with $M = 20$, $c = 1$, $r = 2$, $p = 1$, $m = 0.5$ (I suggest to keep these as input parameters, since it helps later on):

1) Implement a dynamic programming algorithm to maximize the expected profit per stage over K stages, starting from an empty warehouse:

$$\bar{P}_K^* \triangleq \max_{\mu} \frac{1}{K} \mathbb{E}_{\mu} \left[\sum_{k=0}^{K-1} p_k + p_K^{\text{term}} | S_0 = 0 \right],$$

where the expectation is with respect to the dynamics generated under policy μ. Note that, in this case, the optimal policy is non-stationary, i.e. μ_k^* is a function of k.

code logic

Use DP algorithm to solve

1) initialize

$V_0^*(i) = p_K^{term} = \frac{rS_K}{2} = \frac{r}{2}(i)$ gives column vector $V_0^* = \begin{bmatrix} \vdots \end{bmatrix}_{(M+1,1)}$

2) $V_{N-K}^*(i) = \max_{u \in U(i)} c_k(i,u) + \mathbb{E}\left[V_{N-K-1}^*(s_{k+1}) \mid s_k = i, U_k = u\right]$

$= \max_{u \in U(i)} \left[c_k(i,u) + \sum_{j \in S} \mathbb{P}_{j|i,u} V_{N-K-1}^*(j)\right]$

- where $C_k(i,u)$ simplifies to $\bar{C}$ from p111, #3
- $\mathbb{P}_{j|i,u}$ is Prob. matrix
- V_{N-K-1} is previous V val, V_{N-K} is new

get $\mu_k^*(i)$ argmax value, obtained after K iterations. Need a value of K to compute.

Using $V^*(i)$, we want from state i, so $V^*(0)$ for $k = 0 \to 1000$ gets us the $\bar{\rho}^*$ value if $\frac{1}{k} V_k^*(0)$ is done.

Code for getting $\overline{P}_k^*$

```
clear all;
M = 20;
c = 1; r = 2; p = 1; m = .5;
K = 1000;
delta = .00001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate transition prob matrix, P
%generated for all actions and stored in a struct
%Action (u+i) cannot exceed M, so matricies are limited
%giving size of M-a,M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pmat = struct('m',{}); %structre to hold possible Prob matricies
%matrix based on action, so go from 0 to M for action (1 to M+1 index)

for a = 1:M+1 %(0 to M), go through all possble actions to make M prob
 matricies

    Probmat = zeros(M+1-(a-1),M+1); %initalize based on action
%      %note, i goes from 0 to M, but action cannot exceed next state.
%      %thus limit i based on action as we technically look at i+a to
 get j
%      %so this excludes impossible actions
%      %(ex: M=3, a=1, i~=3 as i + a > M, so exlude row M (3),
%      %giving an (3+1-1,3+1) or (3,4) matrix
    for i = 1:size(Probmat,1)
        for j = 1:M+1
            if j > i
               if (j-1) > (i+a-2)%matlab is 1 index so scale
                   Probmat(i,j) = 0;
               else %j <= (i-1) + (a-1)
                   Probmat(i,j) = (1/(M+1));
               end

            else %j<=i
                 if j ~= 1
                    Probmat(i,j) = (1/(M+1));
                else %j == 1
                    Probmat(i,j) = ((M+1-(i-1)-(a-1))/(M+1));
                 end
            end
        end
    end
      if abs(sum(Probmat(i,:)) - 1) > delta %make sure row sums to 1
            i
            fprintf('error'); %if not show where it messed up
      end
    pmat(a).m = Probmat;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected cost cline(i,u)
%store in a struct
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clinevals = struct('c',{});

for j=0:M
    clinetemp = zeros(1,(M+1-j));
    for a = 0:M-j
        costpk = zeros(1,a+1);

        for d = 0:M    %find cost based on demand d, state j, and
 action a
            if d > (j+a)
                costpk(1,d+1) = -(j)*m  -c*a + (j+a)*r - p*(d - (j
+a));

            else
                costpk(1,d+1) = -(j)*m - c*a + d*r;
            end
        end
        costpksum = sum(costpk); %sum each row and multiply by prob
        cline = (1/(M+1))*costpksum;
        clinetemp(1,a+1) = cline;
    end
    clinevals(j+1).c = clinetemp;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected Vstar0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Vstar = zeros(M+1,K+1);
Mustar = zeros(M+1,K+1);
for i = 0:M
Vstar(i+1,1) = (r*i)/2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected Vstar and Mustar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k = 1:K

    for i = 0:M
        Vu = zeros(1,M+1-i); %initialize vector to take max from
        %u loop to loop across actions
        for u = 0:(M-i)
            PV = pmat(u+1).m(i+1,:) * Vstar(:,k); %matrix mult between
 row and column vector
            % this yields sum for Prob mat and previous V star val
            Vu(u+1) = clinevals(i+1).c(u+1) + PV;
        end
        [Vstar(i+1,k+1), Mustar(i+1,k+1)] = max(Vu);
    end

end
Mustar = Mustar - 1;  %reduce by 1 as actions are 0 to M
```

```
Mustar(:,1) = 0;

%want starting from 0, so look at Vstar(0,all k)
%must divide by K
Pstarline = Vstar(1,:);
for k = 2: size(Pstarline,2)
    Pstarline(1,k) = Pstarline(1,k)/(k-1);
end
```

Published with MATLAB® R2020b

function of n.

2) Implement a policy iteration algorithm to maximize the average long-term profit per stage,

$$\bar{P}_{\infty}^{*} \triangleq \max_{\mu} \lim_{K\to\infty} \frac{1}{K}\mathbb{E}_{\mu}\left[\sum_{k=0}^{K-1} p_k | S_0 = 0\right],$$

and let the optimal stationary policy $\mu^*(s)$.

Code logic:

use Policy Iteration

1) Start at a $\mu^{(0)}$ where μ is from lazy inventory problem, where $\mu(0) = M$, $\mu(i) = 0 \ \forall i \neq 0$.

2) under $\mu^{(k)}$ find h_k, v_k

$$\begin{cases} h_k(i) + v_k = c(i, \mu^k(i)) + \sum_{j=1}^{n} \mathbb{P}_{j|i,\mu^k(i)} h_k(j) \quad \forall i \\ h_k(\bar{s}) = 0 \quad ; \quad \bar{s} = 0 \text{ as this is a frequent state} \end{cases}$$

put into Matrix form to solve

$$\begin{bmatrix} \mathbf{h}_\mu \\ v_\mu \end{bmatrix} = Z \cdot W^{-1} \mathbf{c}_\mu$$

where $Z = \begin{bmatrix} \tilde{I}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$, $\tilde{I}^T$ = identity w/ $\bar{s}$ row removed

$W = [I - \mathbf{P}_\mu ; \mathbf{1}] \cdot Z$ $\quad \mathbf{P}_\mu$ is transition prob. matrix built from action given by policy

$\mathbf{c}_\mu$ is $\bar{c}$ values

3) Improve: $\mu^{(k+1)}(i) = \underset{u \in U(i)}{\text{argmax}} \ c(i,u) + \sum_{j=0}^{M} P_{j|i,u} h_k(j)$

$c \Rightarrow \bar{c}$, $P \Rightarrow$ transition. prob. matrix, $h \Rightarrow$ obtained in 2)

4) check $\max_i |h_{k+1}(i) - h_k(i)| < \varepsilon$ & $|v_{k+1} - v_k| < \varepsilon$ $\quad (\varepsilon \approx .00001)$

if true, converged & done, $\mu^*(s)$ obtained

Code for P_∞^* & getting $\mu^*(s)$ through policy iteration

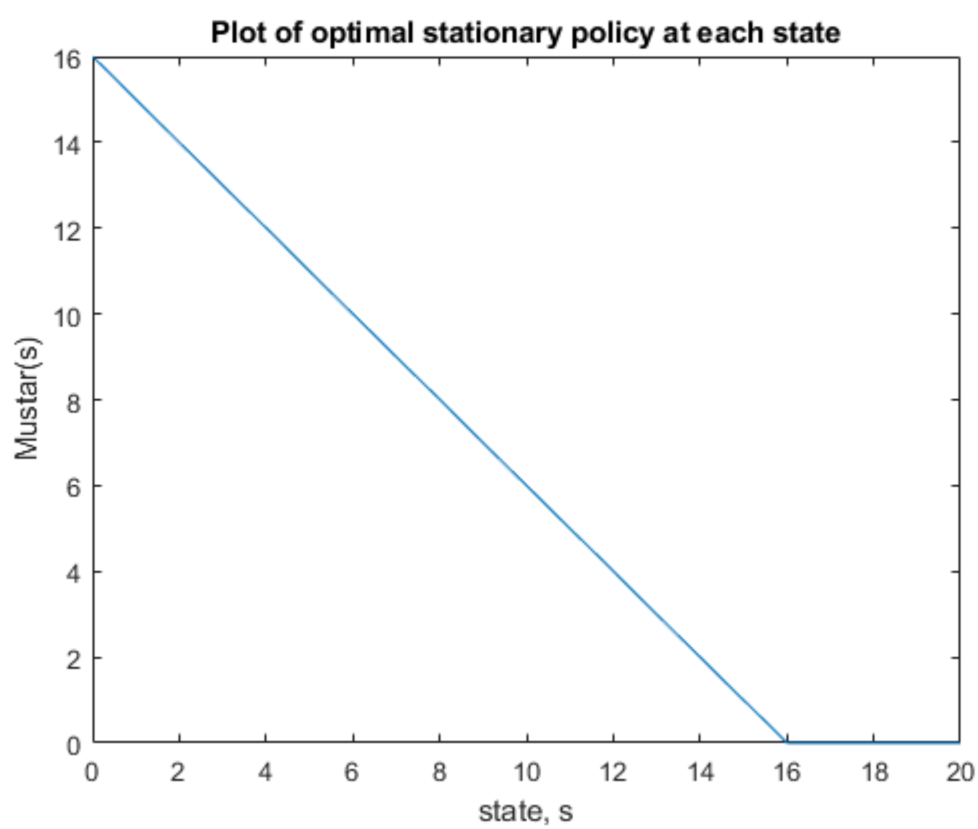
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
M = 20;
c = 1; r = 2; p = 1; m = .5;
K = 1000;
delta = .00001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Pinf star
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate transition prob matrix, P
%generated for all actions and stored in a struct
%Action (u+i) cannot exceed M, so matricies are limited
%giving size of M-a,M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pmat = struct('m',{}); %structre to hold possible Prob matricies
%matrix based on action, so go from 0 to M for action (1 to M+1 index)

for a = 1:M+1 %(0 to M), go through all possble actions to make M prob
 matricies

    Probmat = zeros(M+1-(a-1),M+1); %initalize based on action
%      %note, i goes from 0 to M, but action cannot exceed next state.
%      %thus limit i based on action as we technically look at i+a to
 get j
%      %so this excludes impossible actions
%      %(ex: M=3, a=1, i~=3 as i + a > M, so exlude row M (3),
%      %giving an (3+1-1,3+1) or (3,4) matrix
    for i = 1:size(Probmat,1)
        for j = 1:M+1
            if j > i
               if (j-1) > (i+a-2)%matlab is 1 index so scale
                   Probmat(i,j) = 0;
               else %j <= (i-1) + (a-1)
                   Probmat(i,j) = (1/(M+1));
               end

            else %j<=i
                 if j ~= 1
                    Probmat(i,j) = (1/(M+1));
                else %j == 1
                    Probmat(i,j) = ((M+1-(i-1)-(a-1))/(M+1));
                 end
            end
        end
    end
      if abs(sum(Probmat(i,:)) - 1) > delta %make sure row sums to 1
            i
            fprintf('error'); %if not show where it messed up
      end
    pmat(a).m = Probmat;
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected cost cline(i,u)
%store in a struct
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clinevals = struct('c',{});

for j=0:M
    clinetemp = zeros(1,(M+1-j));
    for a = 0:M-j
        costpk = zeros(1,a+1);

        for d = 0:M    %find cost based on demand d, state j, and
 action a
            if d > (j+a)
                costpk(1,d+1) = -(j)*m  -c*a + (j+a)*r - p*(d - (j
+a));

            else
                costpk(1,d+1) = -(j)*m - c*a + d*r;
            end
        end
        costpksum = sum(costpk); %sum each row and multiply by prob
        cline = (1/(M+1))*costpksum;
        clinetemp(1,a+1) = cline;
    end
    clinevals(j+1).c = clinetemp;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected Vstar0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Vstar = zeros(M+1,K+1);
Mustar = zeros(M+1,K+1);
for i = 0:M
Vstar(i+1,1) = (r*i)/2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate u initial with lazy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%per lazy policy, only buy in state 0, so u(0) = M, else
% u(i) = 0
%Vinf = zeros(M+1,K+1); %value and policy holding arrays
%Muinf = zeros(M+1,K+1);

Mu0 = zeros(M+1,1);
Mu0(1) = M;

Muinf(:,1) = Mu0;           %initialize holders for opt mu and V
Vinf(:,1) = zeros(M+1,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Generate clinemu, Pmu, sbar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% CONSTANTS needed for linear algebra step
sbar = 1; %state 0 is most visited, matlab index is 1
%create [It' 0
%        0   1]
%call it Z
It = eye(M+1);
It(sbar,:) = [];
zerocol = zeros(M+1,1);
zerocol = [zerocol;1];
zerorow = zeros(1,M); %It transpose is n-1,n
Z = [It'; zerorow];
Z = [Z zerocol];
onerow = ones(M+1,1);

%count = iterations taken until converge
count = 1; %initialize count for a while loop with break condition
converge = 0;
while converge == 0
%initilaize cline mu and Prob matrix mu values
clinemu = zeros(M+1,1);
Pmu = zeros(M+1,M+1);

for i = 1:M+1    %get clinemu and Pmu based on policy
action = Muinf(i,count) + 1;
clinemu(i) = clinevals(i).c(action);
Pmu(i,:) = pmat(action).m(i,:);
end

%Create lin alg

W = [(eye(size(Pmu,1)) - Pmu)  onerow] * Z;
%solve for hmu and Vmu
temp = Z*inv(W)*clinemu;
hmu = temp(1:M+1);
vmu = temp(M+2);

Hmu(:,count) = hmu; %store Hmu and Vmu values
Vmu(count) = vmu;

%policy improvment
%i (state) loop
for i = 0:M

Vumu = zeros(1,M+1-i); %initialize vector to take max from
        %u loop to loop across actions
for u = 0:(M-i)
%action loop

    Vumu(u+1) = clinevals(i+1).c(u+1) + pmat(u+1).m(i
+1,:)*Hmu(:,count);
end
[Vinf(i+1,count+1), Muinf(i+1,count+1)] = max(Vumu);
```

```
end
Muinf(:,count+1) = Muinf(:,count+1) - 1;

if count > 1
%can check for convergence
if max(abs(Hmu(:,count) - Hmu(:,count-1))) < delta && abs(Vmu(count) -
 Vmu(count-1)) < delta
    fprintf('converged to opt policy in %i iterations\n', count);
    converge = 1;
end

end
count = count + 1; %add one to iteration count
end

Pinfgraph = Vmu(count-1)*ones(1,1001);

states = (0:1:M);
plot(states,Muinf(:,count));
xlabel('state, s');
ylabel('Mustar(s)');
title('Plot of optimal stationary policy at each state');
```

converged to opt policy in 4 iterations



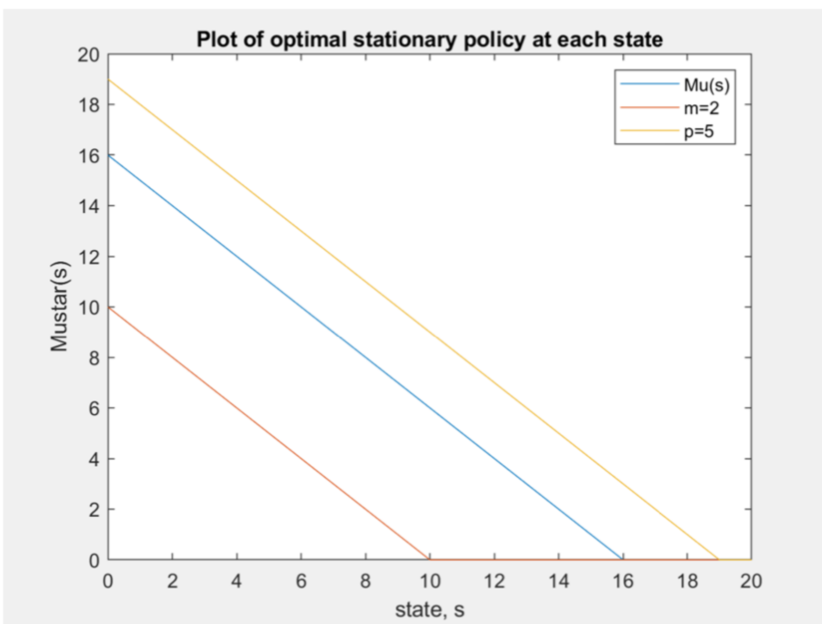
Published with MATLAB® R2020b

3) Plot the optimal stationary policy $\mu^*(s)$ from the previous step as a function of s. Discuss on what you observe: does the policy make intuitive sense?

Looking at the optimal stationary policy, it has a linear relationship until it reaches zero and stays at zero. This makes sense that it eventually reaches zero as the system cannot exceed the maximum inventory M, thus no purchases will be made. In order to avoid a penalty, the value is kept around the max inventory but not quite at the max. This could be as it is better to not over-purchase in case demand decreases.

However, to really explore the trend, the system is very sensitive to the m,r,c,p values, as these determine the cost per stage.

For example, if you increase the value of m from .5, to 2, it causes the system to purchases less as it is more cost-effective to keep less inventory. A linear relationship spanning most of the states can be achieved by adding a high value for p, as the system would then attempt to always have inventory to match demand.



4) Simulate the system under both the *optimal* stationary control policy $\mu^*(s)$ defined above and under the lazy inventory management policy defined earlier: generate a single sequence of states $\{S_0, \ldots, S_{1000}\}$, optimal and lazy actions $\{U_0, \ldots, U_{1000}\}$ and profits $\{p_0, \ldots, p_{1000}\}$, starting from $S_0 = 0$, using the state dynamics defined earlier, and compute

$$\hat{P}_K = \frac{1}{K}\sum_{k=0}^{K-1} p_k + p_K^{\text{term}}, \quad \forall K = 0, \overset{5}{\ldots}, 1000.$$

for both the optimal and lazy inventory management policies. Note: to make the two systems comparable, it is essential that you evaluate the two policies under the *same* realization of the sequence of demands. To do so, it might help to first generate the sequence of demands, and then use the state dynamics to generate the state sequence.

logic:

Using code from p11, we can generate random demands from rand[0 M] as it is iid.

lazy we can use demand & set states with $\mu(0)=M, \mu(i)=0$

Using same demand we can do non lazy.

action being $\mu^*(s)$ from previous part.

So get demand, initial state = 0, get action, get cost based on demand, state, and action.

States: Sseq & Sseqstar

Action: Useq + Useqstar

profit: Cseq + Cseqstar

demand: dseq (same for both)

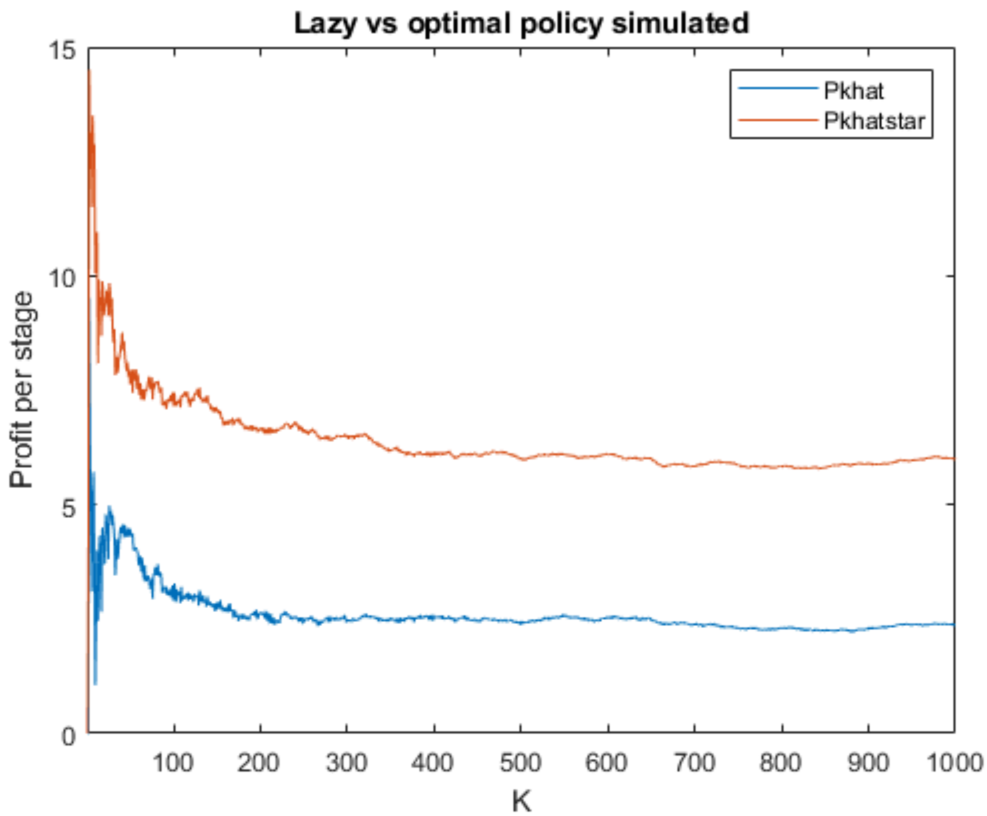
Simulated $\hat{P}_k^*$ & $\hat{P}_k$ same demand

```
clear all;
%policy results from code

M = 20; c = 1; r = 2; p = 1; m = .5;
So = 0;
Prob = 1/(M+1);
K= 1000;
delta = .00001;

Muinf = [ 16
    15
    14
    13
    12
    11
    10
     9
     8
     7
     6
     5
     4
     3
     2
     1
     0
     0
     0
     0
     0 ];
Kmat = [1:K+1]; %needed for plot
%Simulate values for Pkhat, using lazy policy
%code from part 2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate sequence of demands and states
%demands are random integer between 0 and M, iid
%Generate sequence of states based on demand and state dynamics
%keep track of d>s for penalty needed in cost
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Sseq = zeros(1,K+1);  %initialize vector for state sequence
dseq = zeros(1,K+1);  %initialize vector for demand sequence
Useq = zeros(1,K+1);
penalty = zeros(1,K+1); %create penalty placeholder

%initial state = 0,and demand
Sseq(1) = So;
dseq(1) = randi([0 M]); %random gumbel

for i  = 2:K+1
    if Sseq(i-1) == 0
```

```
        Skh = M;
        Sseq(i) = Skh - dseq(i-1); %calculate new state based on
 previous
        dseq(i) = randi([0 M]);    %generate new demand
        Useq(i) = 1;
    elseif dseq(i-1) > Sseq(i-1)
        Sseq(i) = 0;               %demand > sequence goes to zero
        dseq(i) = randi([0 M]);    %generate next demand
        penalty(i-1) = 1;          %penalty is incured
        Useq(i) = 1;
    else
        Sseq(i) = Sseq(i-1) - dseq(i-1); %new state
        dseq(i) = randi([0 M]);    %generate new demand
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate sequence of costs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cseq = zeros(1,K+1);  %initialize vector for cost sequence
for i = 1:K+1
%     if Sseq(i) == M
%         pkseq(i) = -m(Sseq(i)) + dseq(i)*r;
    if Sseq(i) == 0
        cseq(i) = -M*c + dseq(i)*r;
    else
       if penalty(i) == 1 %need a penalty based on unmet demand
           cseq(i) = -m*(Sseq(i)) + Sseq(i)*r - p*(dseq(i) - Sseq(i));
       else
           cseq(i) = -m*(Sseq(i)) + dseq(i)*r;
       end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Pk(hat)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sum_j P(D=j)cost(i,j) in state i
%P(Dk = d) = 1/M+1 = Prob
Pkhat = zeros(1,K+1); %1/K cannot do K = 0, so assum P0hat = 0
for k = 2:K+1
   pklsum = sum(cseq(1:(k-1)));  %costs are based on state at time k
   %see generate sequence of costs. cost based on sequence at k
   pkterm = r*(Sseq(k-1))/2;
   Pkhat(k) = (1/(k-1))*(pklsum + pkterm);
end


%%%%%%Non lazy values%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Generate state, action = Mustar(state), using same sequence of
 demands
%generated earlier
```

```
Sseqstar = zeros(1,K+1);  %initialize vector for state sequence
Useqstar = zeros(1,K+1);
%initial state = 0,and demand
Sseqstar(1) = So;
Useqstar(1) = Muinf(Sseqstar(1) + 1);
for i = 2:K+1
    %becomes value less than 0, gets a penatly, new state must be 0
    if dseq(i-1) > (Sseqstar(i-1) + Useqstar(i-1))
        Sseqstar(i) = 0;
        Useqstar(i) = Muinf(Sseqstar(i) + 1);
    else %new state is action + current - demand
        Sseqstar(i) = (Sseqstar(i-1) + Useqstar(i-1)) - dseq(i-1);
        Useqstar(i) = Muinf(Sseqstar(i) + 1);
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate sequence of costs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%must follow maintenance on current state, buy based on action
% sold demand based on current state + action if too much demand
%otherwise just sell based on demand
cseqstar = zeros(1,K+1);  %initialize vector for cost sequence
for i = 1:K+1
    if dseq(i) > (Sseqstar(i) + Useqstar(i))
        cseqstar(i) = -Sseqstar(i)*m  -c*Useqstar(i) + ...
            (Sseqstar(i)+Useqstar(i))*r - ...
            p*(dseq(i) - (Sseqstar(i)+Useqstar(i)));

    else
        cseqstar(i) = -Sseqstar(i)*m - c*Useqstar(i) +...
            dseq(i)*r;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Pkstar(hat)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sum_j P(D=j)cost(i,j) in state i
%P(Dk = d) = 1/M+1 = Prob
Pkhatstar = zeros(1,K+1); %1/K cannot do K = 0, so assum P0hat = 0
for k = 2:K+1
   pklsums = sum(cseqstar(1:(k-1)));  %costs are based on state at
 time k
   %see generate sequence of costs. cost based on sequence at k
   pkterms = r*(Sseqstar(k-1))/2;
   Pkhatstar(k) = (1/(k-1))*(pklsums + pkterms);
end

plot(Kmat,Pkhat,Kmat,Pkhatstar);
xlabel('K');
ylabel('Profit per stage');
legend('Pkhat','Pkhatstar');
```

```
xlim([1,K]);
title('Lazy vs optimal policy simulated');
```



Published with MATLAB® R2020b

5) Plot the following for both the lazy and optimal inventory management policies: $\bar{P}_K$ and $\hat{P}^*_K$ versus $K = 0, 1, \ldots, 1000$, and $\bar{P}^*_\infty$ (as a line that spans the entire range of K). Discuss on what you observe. How much do you gain by optimizing the inventory management policy? If you look at the realizations $\hat{P}^*_K$ and $\hat{P}_K$ of the optimal and lazy policies, is $\hat{P}^*_K > \hat{P}_K$ always? If not, why?

where

$\bar{P}_K$ from part II

$\bar{P}^*_K$ from #1 pIII

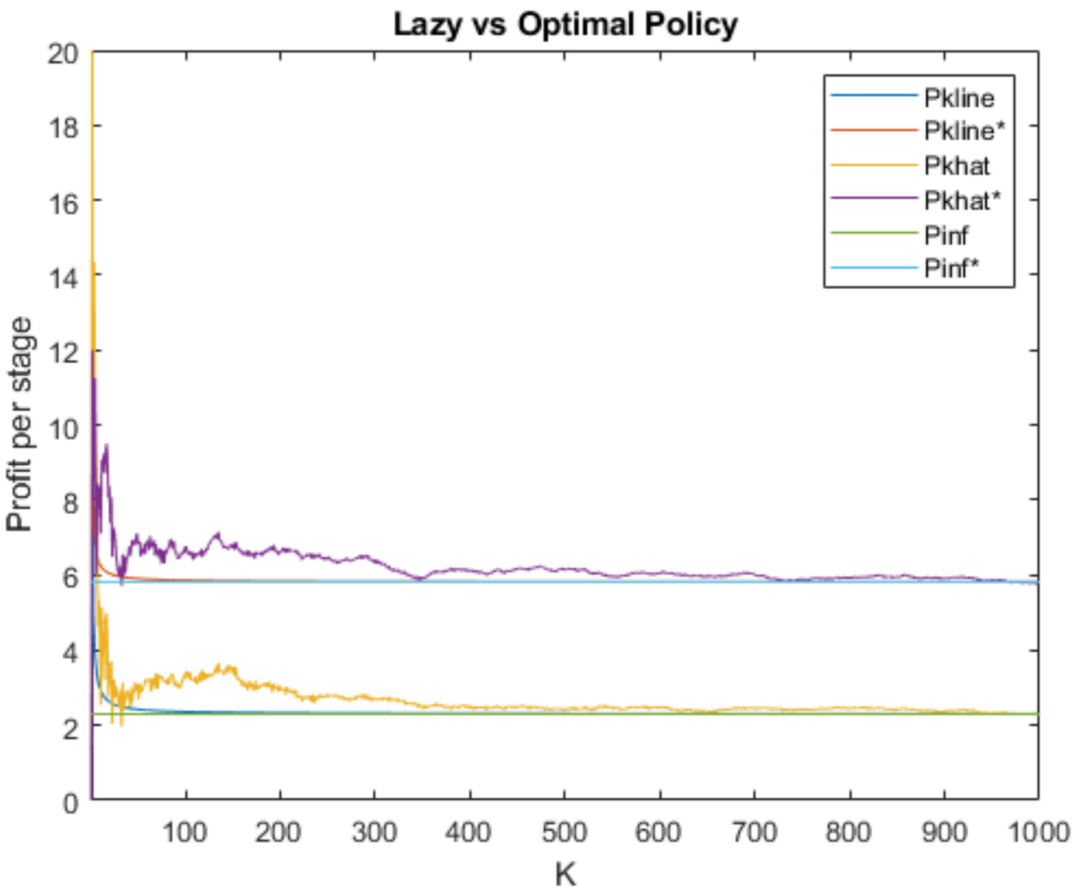
$\hat{P}_K$, $\hat{P}^*_K$ caclulatd from same demand in 4

$\bar{P}_\infty$ from pII

$\bar{P}^*_\infty$ from $V^*(s)$ (calculated when setting $\mu^*(s)$)

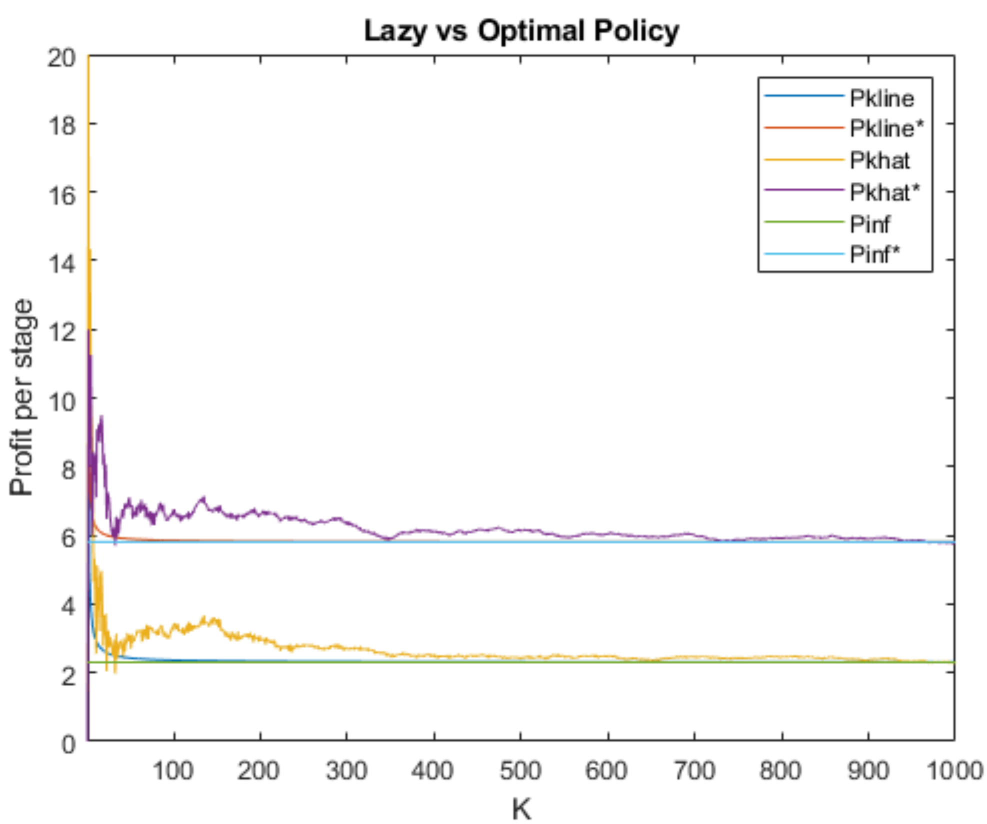
Optimizing the policy has about a +3 effect on the system. This is a good increase and helps us visualize how a simple policy optimization is better than a lazy one. Change in the m,r,p,c values could also be explored to see how the optimal policy reacts, and if the lazy one could end up being superior in some instances.

After running the simulated data a bunch of times to generate new demand sequences, it seems that the optimal policy is better than the lazy one almost always. This is mainly due to the parameters m,c,p, and r defined in the problem. Changing those values around would have an effect on the optimal policy by making it more expensive to store or more expensive to purchase. But as of now it is pretty cheap to store and pretty cheap to buy when compared to selling for profit. Both pkhats converge on a profit, with Pkhat settling around 2, and Pkhat star settling around 5. There may be a slight dip in the beginning, but other than that brief moment, Pkhat* far outperforms the Pkhat.



Code to graph all profits

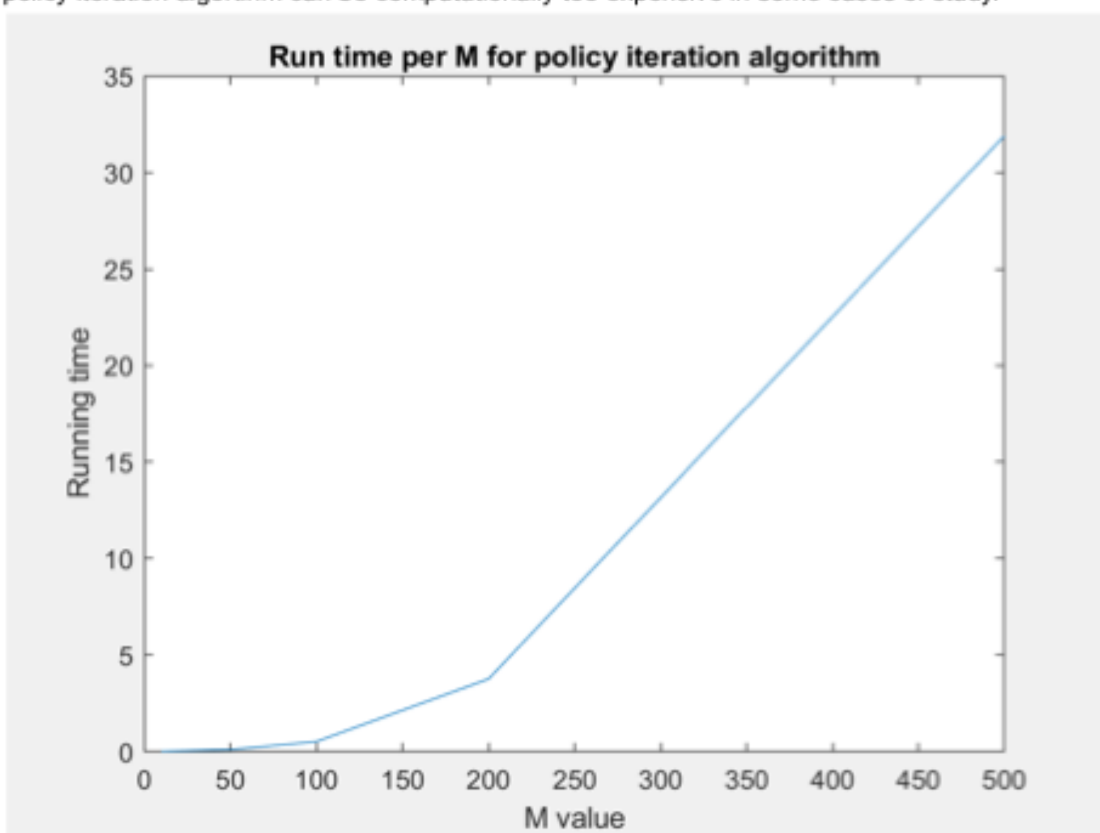
```
%code to graph all values.
%Note, data has been presaved and uploaded to simply just graph.
%data taken from the repective matlab code it was solved in.
%Pkhat and Pkhatstar are generated using same demand. Other graphs
 using
%these may look different as a different demand sequence may be used.
 4
K = 1000;
load('PklinepII.mat');load('PstarlinepIII.mat');
load('Pkhat1.mat');load('Pkhatstar1.mat');
load('PinfgraphpII.mat');load('PinfgraphpIII.mat');
Kmat = [1:K+1]; %needed for plot

plot(Kmat,Pkline,Kmat,Pstarline,...
    Kmat,Pkhat,Kmat,Pkhatstar,...
    Kmat,Pinfgraph,Kmat,Pinfgraphstar);
xlabel('K');
ylabel('Profit per stage');
legend('Pkline','Pkline*','Pkhat','Pkhat*',...
     'Pinf','Pinf*');
xlim([1,K]);
title('Lazy vs Optimal Policy');
```



Published with MATLAB® R2020b

6) Now, let's explore the running time of the algorithm vs problem size M. Run the policy iteration algorithm that you implemented above to optimize the average long term profit, for the following values of M, until convergence: $M \in \{10, 20, 50, 100, 200, 500\}$. For each value of M, evaluate the total running time of the algorithm (tic toc can be used in Matlab to compute elapsed time). Plot running time vs M and comment on what you observe. Does the running time grow linearly or exponentially (or else) with respect to M?

We can see that the running time jumps between each M value. This makes sense as we are evaluating matrices of size M and some of size MxM, which leads to very taxing computation times. It would be expected to have a relatively exponential trend in this instance, which is confirmed by plotting the running time vs the M value. This can help show that very quickly the policy iteration algortihm can be computationally too expensive in some cases of study.



Code to capture time.

```
clear all;
Mval = [10,20,50,100,200,500];
times = zeros(1,length(Mval));
c = 1; r = 2; p = 1; m = .5;
K = 5;
delta = .00001;
tic
t = 1;
for M = Mval
    M
    Hmu = zeros(M+1,1);
    Vmu = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Pinf star
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate transition prob matrix, P
%generated for all actions and stored in a struct
%Action (u+i) cannot exceed M, so matricies are limited
%giving size of M-a,M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pmat = struct('m',{}); %structre to hold possible Prob matricies
%matrix based on action, so go from 0 to M for action (1 to M+1 index)

for a = 1:M+1 %(0 to M), go through all possble actions to make M prob
 matricies

    Probmat = zeros(M+1-(a-1),M+1); %initalize based on action
%      %note, i goes from 0 to M, but action cannot exceed next state.
%      %thus limit i based on action as we technically look at i+a to
 get j
%      %so this excludes impossible actions
%      %(ex: M=3, a=1, i~=3 as i + a > M, so exlude row M (3),
%      %giving an (3+1-1,3+1) or (3,4) matrix
    for i = 1:size(Probmat,1)
        for j = 1:M+1
            if j > i
               if (j-1) > (i+a-2)%matlab is 1 index so scale
                   Probmat(i,j) = 0;
               else %j <= (i-1) + (a-1)
                   Probmat(i,j) = (1/(M+1));
               end

            else %j<=i
                 if j ~= 1
                    Probmat(i,j) = (1/(M+1));
                else %j == 1
                    Probmat(i,j) = ((M+1-(i-1)-(a-1))/(M+1));
                 end
            end
        end
    end
```

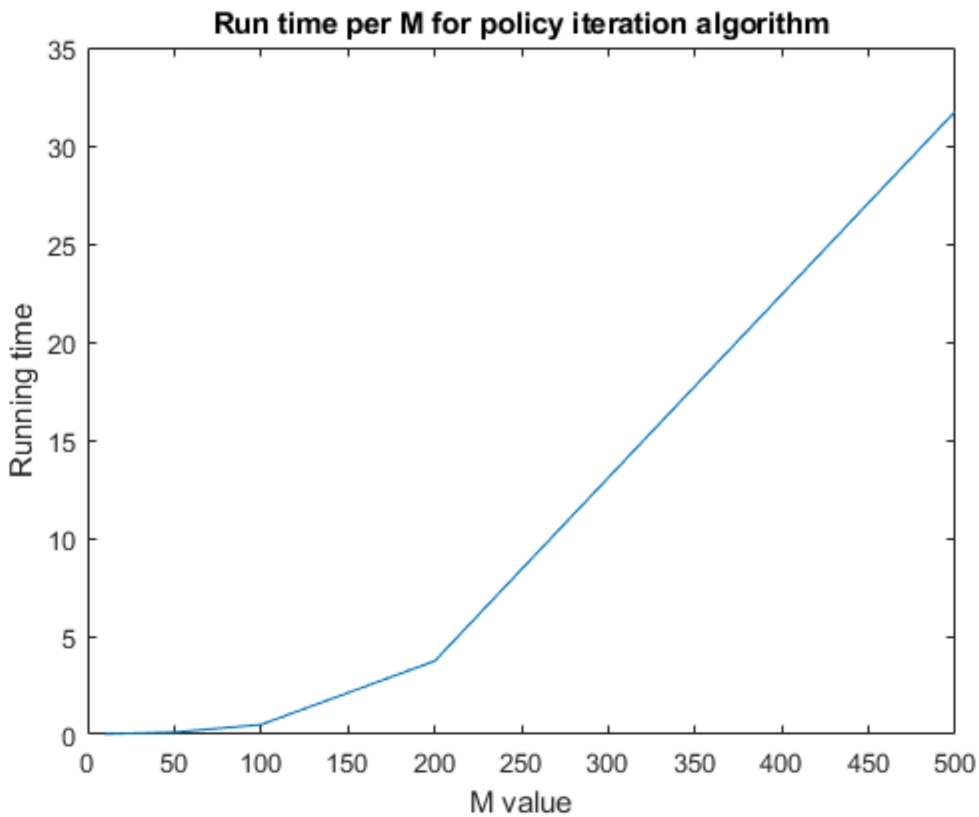
```
      if abs(sum(Probmat(i,:)) - 1) > delta %make sure row sums to 1
            i
            fprintf('error'); %if not show where it messed up
      end
    pmat(a).m = Probmat;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected cost cline(i,u)
%store in a struct
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clinevals = struct('c',{});

for j=0:M
    clinetemp = zeros(1,(M+1-j));
    for a = 0:M-j
        costpk = zeros(1,a+1);

        for d = 0:M    %find cost based on demand d, state j, and
 action a
            if d > (j+a)
                costpk(1,d+1) = -(j)*m  -c*a + (j+a)*r - p*(d - (j
+a));

            else
                costpk(1,d+1) = -(j)*m - c*a + d*r;
            end
        end
        costpksum = sum(costpk); %sum each row and multiply by prob
        cline = (1/(M+1))*costpksum;
        clinetemp(1,a+1) = cline;
    end
    clinevals(j+1).c = clinetemp;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected Vstar0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Vstar = zeros(M+1,K+1);
Mustar = zeros(M+1,K+1);
for i = 0:M
Vstar(i+1,1) = (r*i)/2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate u initial with lazy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%per lazy policy, only buy in state 0, so u(0) = M, else
% u(i) = 0
%Vinf = zeros(M+1,K+1); %value and policy holding arrays
%Muinf = zeros(M+1,K+1);
Muinf = zeros(M+1,1);
Vinf = zeros(M+1,1);
```

```
Mu0 = zeros(M+1,1);
Mu0(1) = M;

Muinf(:,1) = Mu0;            %initialize holders for opt mu and V
Vinf(:,1) = zeros(M+1,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate clinemu, Pmu, sbar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% CONSTANTS needed for linear algebra step
sbar = 1; %state 0 is most visited, matlab index is 1
%create [It' 0
%        0   1]
%call it Z
It = eye(M+1);
It(sbar,:) = [];
zerocol = zeros(M+1,1);
zerocol = [zerocol;1];
zerorow = zeros(1,M); %It transpose is n-1,n
Z = [It'; zerorow];
Z = [Z zerocol];
onerow = ones(M+1,1);

%count = iterations taken until converge
count = 1; %initialize count for a while loop with break condition
converge = 0;
while converge == 0
%initilaize cline mu and Prob matrix mu values
clinemu = zeros(M+1,1);
Pmu = zeros(M+1,M+1);

for i = 1:M+1    %get clinemu and Pmu based on policy
action = Muinf(i,count) + 1;
clinemu(i) = clinevals(i).c(action);
Pmu(i,:) = pmat(action).m(i,:);
end

%Create lin alg

W = [(eye(size(Pmu,1)) - Pmu)  onerow] * Z;
%solve for hmu and Vmu
temp = Z*inv(W)*clinemu;
hmu = temp(1:M+1);
vmu = temp(M+2);

Hmu(:,count) = hmu; %store Hmu and Vmu values
Vmu(count) = vmu;

%policy improvment
%i (state) loop
for i = 0:M

Vumu = zeros(1,M+1-i); %initialize vector to take max from
        %u loop to loop across actions
for u = 0:(M-i)
```

```
%action loop

    Vumu(u+1) = clinevals(i+1).c(u+1) + pmat(u+1).m(i
+1,:)*Hmu(:,count);
end
[Vinf(i+1,count+1), Muinf(i+1,count+1)] = max(Vumu);
end
Muinf(:,count+1) = Muinf(:,count+1) - 1;

if count > 1
%can check for convergence
if max(abs(Hmu(:,count) - Hmu(:,count-1))) < delta && abs(Vmu(count) -
 Vmu(count-1)) < delta
    fprintf('converged to opt policy in %i iterations\n', count);
    converge = 1;
end

end
count = count + 1; %add one to iteration count
end


times(t) = toc;
t = t+1;
end
plot(Mval,times);
xlabel('M value');
ylabel('Running time');
title('Run time per M for policy iteration algorithm');
```

```
M =

    10

converged to opt policy in 3 iterations

M =

    20

converged to opt policy in 4 iterations

M =

    50

converged to opt policy in 4 iterations

M =

   100

converged to opt policy in 4 iterations
```

```
M =

   200

converged to opt policy in 4 iterations

M =

   500

converged to opt policy in 4 iterations
```



Published with MATLAB® R2020b