MATH4406 – HW2

Madeleine Nargar, 41214538

Question 1a)

Let:
x_t = units of stored inventory at start of month t, $x_t \in \{0, 1, \dots, M\}$,
y_t = number of backlogged units at start of month t, $y_t \in \{0, 1, \dots, \}$ (assume no limit on units back-logged),
$s_t = x_t - y_t \equiv$ "net" units of stock at start of month t,
M = warehouse capacity a_t = units ordered in month t,
D_t = demand in month t, random variable distributed according to $p_j = \mathbb{P}(D_t = j),\ j = 0, 1, \dots.$
$h(u)$ = holding cost per unit stored for one month
$b(u)$ = back-log cost per month
$O(u)$ = cost per unit ordered,
$f(u)$ = revenue per unit sold.

Assume timing of events is the same as in Figure 3.2.1(Puterman, Sec 3.2.1), with revenue received for D_t units during the order period (regardless of how many orders filled in t):

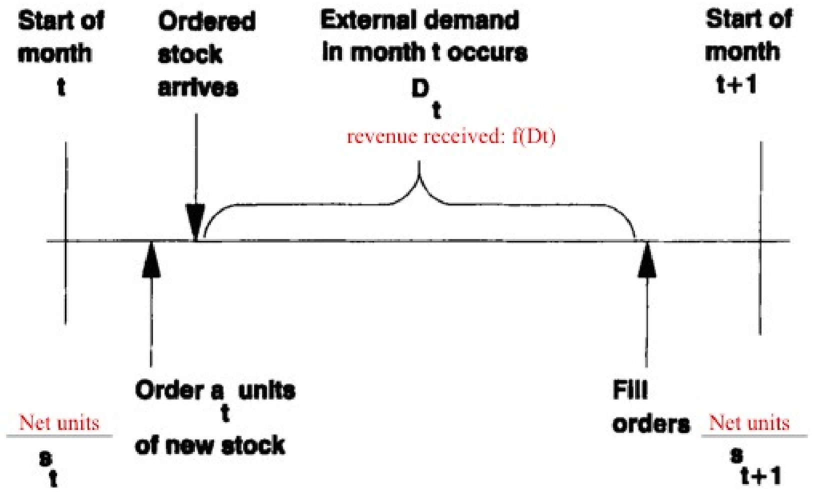


Figure 3.2.1 Timing of events in the inventory model of Sec. 3.2.1.

Assuming that orders are always back-logged whenever demand exceeds available units (i.e. decision maker chooses how many units to order, but does not control back-logging, which occurs automatically if necessary/possible), then the system is governed by the following equation:

$$s_{t+1} = s_t + a_t - D_t$$

Decision epochs: $T = \{1, 2, \ldots, N\},\ N \leq \infty$

State space: (states s $\equiv$ ”net” units of stock at start of month)

$\mathbf{S} = \{\ldots, -1, 0, 1, \ldots, M\}$

Action set:
$\mathbf{A_s} = \{0, 1, \ldots, maxorder_s\}$, where $maxorder_s = min(M, M - s)$

Transition probabilities:
$\mathbb{P}_t(j|s, a) = p_{s+a-j}$

Rewards: The following assumes that customers pay for units at the time they order them (regardless of whether orders are filled immediately or back-logged). Also assumes that order are added to the back-log (if necessary) at the end of month t, with their back-log cost paid from the start of month $t + 1$.

Revenue at time t = $f(s_t + a_t - s_{t+1})$ (i.e. $f(D_t)$) is dependent on s_{t+1}.
Instead use $\mathbb{E}$(revenue at time t) = $\sum_{j=0}^{\infty} f(j)p_j \ \forall t < N$.
In this formulation, expected revenue at t is bounded only by D_t, not by $s_t + a_t$ (as it is in formulation without back-logging).

$\Rightarrow$ **Expected reward**:
$r_t(s, a) = -O(a) - h(max(s + a, a)) - b(-min(0, s)) + \sum_{j=0}^{\infty} f(j)p_j$, for $t = 1, \ldots N - 1$
$r_N(s) = g^{(+)}(max(0, s) - g^{(-)}(max(0, -s)$, for t = N
(where $g^{(+)}$ = salvage value of inventory, $g^{(-)}$ = penalty for back-logged units never filled)

Question 1b)

Use $r_t(s, a) = -O(a) - h(max(s + a, a)) - b(max(0, -s)) + \sum_{j=0}^{\infty} f(j)p_j$ with:

$f(u) = 8u \Rightarrow \qquad \sum_{j=0}^{\infty} f(j)p_j = 0 * 1/4 + 8 * 1/2 + 16 * 1/4 = 16$

$h(s, a) = max(s + a, a), \qquad O(a) = \begin{cases} 0 & a = 0 \\ 4 + 2a & a > 0 \end{cases}$

$b(s) = max(0, -3s) \qquad g^{(+)}(s) = 0$

No $g^{(-)}(s)$ is specified, but some penalty for back-logged units that have been paid for but never delivered seems sensible. e.g. $g^{(-)}(s_N) = max(0, 8s_N) \equiv$ refunding unfilled orders.

Assuming that a maximum of three units can be back-logged at once (to avoid infinite state space as formulated above), gives:

$\mathbf{r_t(s, a)}$ for $t = 1, \ldots, N-1$

		s						
		-3	**-2**	**-1**	**0**	**1**	**2**	**3**
	0	-1	2	5	8	7	6	5
a	**1**	-8	-5	-2	1	0	1	-
	2	-11	-8	-5	-2	-3	-	-
	3	-14	-11	-8	-5	-	-	-

e.g. $\mathbf{r_N(s)} = -max(0, 8s_N)$ for $t = N$.

$\mathbf{p_t(j|s, a)}$

		j						
		-3	**-2**	**-1**	**0**	**1**	**2**	**3**
	-3	1	0	0	0	0	0	0
	-2	3/4	1/4	0	0	0	0	0
s+a	**-1**	1/4	1/2	1/4	0	0	0	0
	0	0	1/4	1/2	1/4	0	0	0
	0	0	0	1/4	1/2	1/4	0	0
	2	0	0	0	1/4	1/2	1/4	0
	3	0	0	0	0	1/4	1/2	1/4

Question 1c)

In this formulation, all is the same as part a) except the rewards. Assuming that payment for an order is received in the time period the order is *filled* (vs. time period in which order is *placed* in part (a)). This means expected revenue at time t now depends on s_t and a_t, as regardless of demand and back-log, orders filled is limited by inventory available.

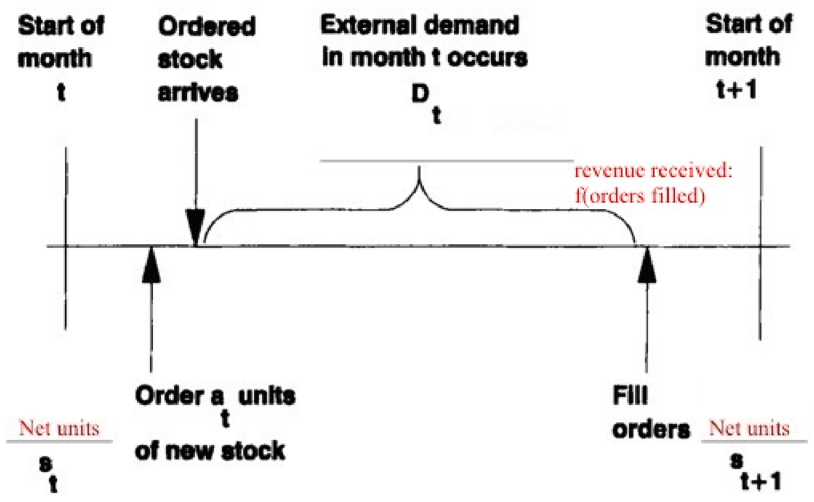


Figure 3.2.1 Timing of events in the inventory model of Sec. 3.2.1.

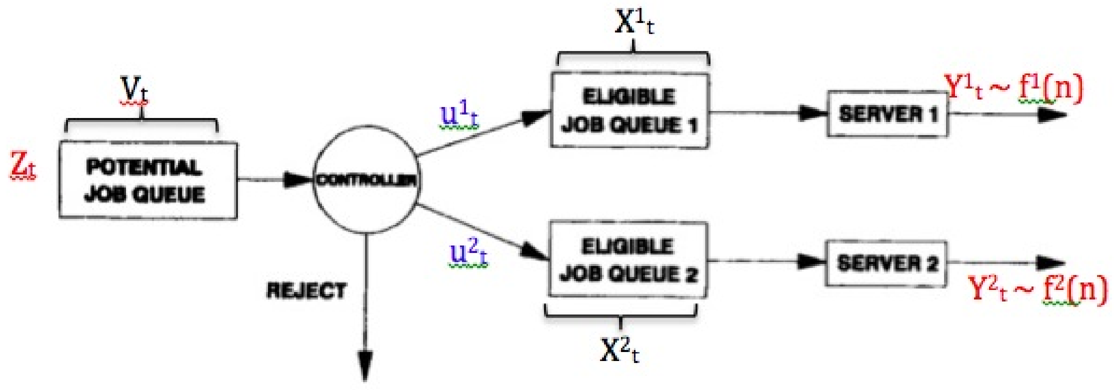
Revenue at time t

$$= f(\text{orders filled in month t}) = \begin{cases} f(D_t - min(0, s_t)) & if \;\; D_t - min(0, s_t) \leq max(0, s_t) + a_t \\ f(max(0, s_t) + a(t)) & if \;\; D_t - min(0, s_t) > max(0, s_t) + a_t \end{cases}$$

$\Rightarrow \mathbb{E}(\text{revenue at time t}) = \sum_{j=0}^{a+s} f(j)p_j + f(s+a)\sum_{j=s+a}^{\infty} p_j$

$\Rightarrow$ **Expected reward**:
$r_t(s,a) = -O(a) - h(max(s+a,a)) - b(-min(0,s)) + \sum_{j=0}^{a+s} f(j)p_j + f(s+a)\sum_{j=s+a}^{\infty} p_j$, for $t = 1, \dots N-1$
$r_N(s) = g^{(+)}(max(0,s) - g^{(-)}(-min(0,s)$, for t = N
(where $g^{(+)}$ = salvage value of inventory, $g^{(-)}$ = penalty for back-logged units never delivered. Note: unlikely to require penalty for unfilled back-log here, as these units will have attracted no revenue.)

Question 2 (Problem 3.21)



For $i = 1, 2, t \in \{1, 2, \dots, \}$:
Let V_t = number of jobs in potential job queue, $V_t^i \in \mathbf{V} = \{0, 1, \dots\}$
Let X_t^i = number of jobs in eligible queue, $X_t^i \in \mathbf{X^i} = \{0, 1, \dots\}$
Let Z_t = random variable, number of arrivals to eligible queue, with $\mathbb{P}(Z_t = n) = g(n)$ for $n \geq 0$
Let Y_t^i = random variable, number of jobs completed by server i, with $\mathbb{P}(Y_t^i = n) = f^i(n)$ for $n \geq 0$.
$h_i(n)$ = cost of holding n jobs in server i.
R = reward per job completed.

State space: $\mathbf{S} = \mathbf{V} \times \mathbf{X^1} \times \mathbf{X^2}$

Decision epochs: $T = \{1, 2, \dots, \}$

Action set: Let $u_t^i \equiv$ number admitted to server i at time t, where $u_t^i \in \mathbf{U^i} = \{0, 1, \dots, \}$. Actions are of the form $u = (u^1, u^2)$.
$\mathbf{A_t} = \mathbf{U^1} \times \mathbf{U^2} - \{u_t : u_t^1 + u_t^2 > V_t\}, \ \forall t \in T$.

The system is governed by the following state equations (giving the transition probabilities below):

$X_{t+1}^i = min(0, X_t^i + u_t^i - Y_t)$, subject to $u_t^1 + u_t^2 \leq V_t \quad \forall t \in T$
$V_t = Z_t$

Transition probabilities:
Considering the servers individually, each have the following transition probabilities:

$$\mathbb{P}(x'|x,u) = \begin{cases} \sum_{n=x+u}^{\infty} f(n) & x' = 0 \qquad \text{i.e. } Y \geq x+u \\ f(x+u-x') & 0 < x' \leq u+x \qquad \text{i.e. } Y < x+u \rightarrow x' = x+u-Y \\ 0 & x' > u+x \end{cases}$$

The number of arrivals to the eligible queue is independent of the jobs in the servers, and of the previous system state (as any jobs not sent to a server are rejected, so nothing left in eligible queue at the end of each time step), i.e. $\mathbb{P}(v'|v, x^1, x^2) = g(v')$.

As the number of jobs completed by the servers are independent of each other, and of the arrival rate (given a particular state), we can use:
$\mathbb{P}((v', x^{1'}, x^{2'})|(v, x^1, x^2), (u^1, u^2)) = \mathbb{P}(v')\mathbb{P}(x^{1'}|x^1, u^1)\mathbb{P}(x^{2'}|x^2, u^2)$

$$= \begin{cases} 0 & x^{i'} > x^i + u^i \text{ for any i}, \ \forall v, v' \\ g(v')(\sum_{n=x^1+u^1}^{\infty} f^1(n))(\sum_{n=x^2+u^2}^{\infty} f^2(n)) & x^{1'} = x^{2'} = 0, \ \forall v, v' \\ g(v')(f^1(x^1+u^1-x^{1'})(\sum_{n=x^2+u^2}^{\infty} f^2(n)) & 0 < x^{1'} \leq x^1 + u^1, \ x^{2'} = 0, \ \forall v, v' \\ g(v')(f^1(x^1+u^1-x^{1'})(f^2(x^2+u^2-x^{2'}) & 0 < x^{1'} \leq x^1 + u^1, \ 0 < x^{2'} \leq x^2 + u^2, \ \forall v, v' \\ g(v')(\sum_{n=x^1+u^1}^{\infty} f^1(n))(f^2(x^2+u^2-x^{2'}) & x^{1'} = 0, \ 0 < x^{2'} \leq x^2 + u^2, \ \forall v, v' \end{cases}$$

Rewards:
$r_t((v, x^1, x^2)|(u^1, u^2)) = R\sum_{i=1,2} \mathbb{E}(min(Y_t^i, x^i + u^1)) - \sum_{i=1,2} h_i(x^i + u^i)$
where $\mathbb{E}(min(Y_t^i, x^i + u^1)) = \sum_{n=1}^{x^i+u^i-1} f^i(n) + (x^i + u^i)\sum_{n=x^i+u^i}^{\infty} f^i(n)$

A good policy for the system (in my opinion) would be to allocate incoming jobs to the faster/cheaper server (by some ratio of average service speed to holding cost) whenever its queue was under a certain threshold (based on its holding cost); otherwise allocate jobs to slower/dearer server whenever its queue was under a second threshold (based on the slow server's holding cost); otherwise reject the jobs.
e.g. $(u^{fast}, u^{slow}) =$

$$\begin{cases} (max(0, Z - thr^{fast}), max(0, Z - thr^{fast} - thr^{slow})) & for\ X^{fast} \leq thr^{fast}, X^{slow} \leq thr^{slow} \\ (0, max(0, Z - thresh^{slow})) & for\ X^{fast} > thr^{fast}, X^{slow} \leq thr^{slow} \\ (0, 0) & for\ X^{fast} > thr^{fast}, X^{slow} > thr^{slow} \end{cases}$$

Question 3 (Problem 3.26)

Assume that the lion starts the day with s kg of meat in its gut, then may hunt and eat, then at the end of the day the amount of meat in its gut descreases by 6 kg. Assume that a lion can start a day with no meat in its gut, but starting day t with a negative amount of meat in its gut means the lion dies at time t. Assume the lion can always find a group of the desired size to hunt with, and that the lion never chooses to hunt in a group of more than 6, as this will not increase its probability of a sucessful kill, but will descrease the amount of meat gained.

State space: $S = \{-1, 0, 1, \dots, 30\}$,
where state $s \equiv \begin{cases} \text{starvation/death}, & s = -1 \\ \text{kg of meat in lion's gut}, & s \geq 0 \end{cases}$

Decision epochs: $\{1, 2, \dots, T\}$

Action set: $A = \{0, 1, \dots, 6\}$, where action $n \equiv \begin{cases} \text{don't hunt} & a = 0, \\ \text{hunt in a group of } a \text{ lions} & a \geq 1 \end{cases}$

Rewards: $r(t = T) = min\{s(T), 0\}$, i.e. MDP has value of -1 if lion has starved by time T, or 0 if the lion has a non-negative amount of food in its gut in time T (i.e. still alive).

This maximises the chance the lion will still be alive at time T (without regard to how much food reserve it has left). It may be desirable for the lion to always have at a decent amount of food in its gut (a risk minimisation strategy), in which case something of the form $r(t) = min\{s(t), \text{minimum target}\}$ for $t \in T$

Transition probabilities:

$$\mathbb{P}(j|s,a) = \begin{cases} 1 & j = -1,\ s = -1,\ a \in A \quad \text{(once dead, lion stays dead)} \\ 0 & j\ else,\ s = -1,\ a \in A \\ 1 & j = min\{-1, s - 6\},\ s \geq 0,\ a = 0 \quad \text{(action: don't hunt)} \\ 0 & j\ else,\ s \geq 0,\ a = 0 \\ p_a & j = min\{30, \frac{164}{a} - 6.5 + s\},\ s \geq 0,\ a \geq 1 \quad \text{(successful hunt in group of size a)} \\ 1 - p_a & j = min\{-1, s - 6.5\},\ s \geq 0,\ a \geq 1 \quad \text{(unsuccessful hunt)} \\ 0 & j\ else,\ s \geq 0,\ a \geq 1 \end{cases}$$

These assume: successful hunt in group of n gains $164/n$ kg of meat, each hunt consumes 0.5 kg of meat regardless of success, and that lions require 6 kg of meat daily regardless of action taken.

Question 4

a) Let $l \equiv$ level, $\tau \equiv$ years in level, $e \equiv$ eligibility for promotion (0 = can't apply, 1 = can apply).

State space: States are of the form s $= (l, \tau, e)$, where $l \in L = \{0, 1, 2, 3\}, \tau \in \Gamma = \{0, \dots, 9\}, e \in E = \{0, 1\}$. The state space is a subset of $L \times \Gamma \times E$, consisting states accessible from $(0, 0, 0)$ by the following rules:

$(l, \tau, 1) \rightarrow (l + 1, 0, 0), for\ \tau < \tau max(l), l < 3$ where $\tau max(l) = 9 - 2l$

$(l, \tau, 1) \rightarrow (l, \tau + 1, 0), for\ \tau < \tau max(l), l < 3$
$(l, \tau, e) \rightarrow (l, \tau + 1, 1), for\ \tau < \tau max(l) - 1, l < 3$
$(l, \tau, e) \rightarrow (l, \tau + 1, 0), for\ \tau = \tau max(l) - 1, or\ l = 3, \tau < \tau max(l)$

This leads to 43 states.

Action set: $A = \{W, A\}$, where $W \equiv$ don't apply, $A \equiv$ apply.

Decision epochs = {0,. . . ,9}

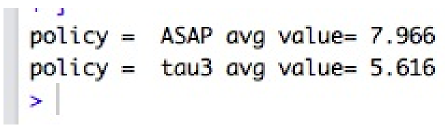
Rewards: $r((l, \tau, e)) = l$

Transition probabilities:
$\mathbb{P}((l, 0, 0)|(l, \tau, 1), A) = q^\tau$ (successful application)
$\mathbb{P}((l, \tau + 1, 0)|(l, \tau, 1), A) = 1 - q^\tau$ (unsuccessful application)
$\mathbb{P}((l, \tau + 1, 1)|(l, \tau, e), W) = 1, for\ \tau < \tau max(l) - 1, l < 3$
$\mathbb{P}((l, \tau + 1, 0)|(l, \tau, e), W) = 1, for\ \tau = \tau max(l) - 1, or\ l = 3, \tau < \tau max(l)$
else, $\mathbb{P}(j|s, a) = 0$

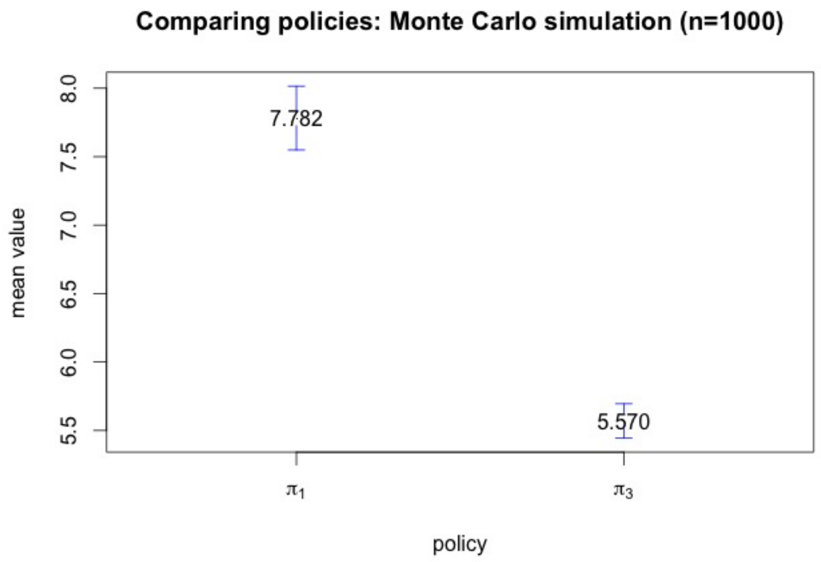
b) There are 18 states with $e = 1$ (i.e. where you are eligible to apply for promotion), meaning there are only 18 states where a decision is necessary, with two options, wait or apply, available in each. For other states only the wait action is possible. Thus, there are 2^{18} stationary deterministic decision rules possible. (There will be infinitely many stationary randomised policies, consisting of $q_d(s)$ for each of the 18 states.)

$$\Pi^{SD} = D_1 \times D_2 \cdots \times D_{43}, \quad \text{where } D_s = \begin{cases} \{W\} & \forall s = (l, \tau, e) : e = 0 \\ \{W, A\} & \forall s = (l, \tau, e) : e = 1 \end{cases}$$

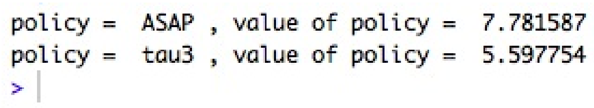
c) Simulating these policies (see `promotionMDP.R`), gives the following average rewards:


```
policy =  ASAP avg value= 7.966
policy =  tau3 avg value= 5.616
> |
```

This suggests the 'apply as soon as you can' policy is better.



d) Implementing the policy evaluation algorithm gives comparable results with the simulation above:



```
policy =  ASAP , value of policy =  7.781587
policy =  tau3 , value of policy =  5.597754
>
```

See `promotionMDP.R` for details.

e) Testing all the stationary deterministic policies (2^{18}) shows that the 'apply as soon as you can' policy is in fact optimal. There are actually 512 degenerate optimal policies, with only 9 of the 'eligible to apply' states being relevant to the optimal policy, i.e. for all states $\notin \{(0,1,1),(0,3,1),(0,5,1),(0,7,1),(1,1,1),(1,3,1),(1,5,1),(2,1,1),(2,3,1)\}$, the action chosen does not affect the optimal policy.

Question 5

For each individual baby: state space: $S_i = \{1,2,3\}$, where 1 = sleeping, 2 = awake, 3 = screaming; action set: $A_i = \{1,2\}$, where 1 = make baby i passive, 2 = feed/make active baby i. Each baby has the same transition probabilities, P^1 when passive and P^2 when active. The states and actions of the babies are independent of each other, subject to the constraint that no more than two of the babies can be active at any given time. This leads to the following formulation:

State space: $S = S_1 \times S_2 \times S_3$, i.e. of form $s = (s_1, s_2, s_3)$

Action set: $A = A_1 \times A_2 \times A_3 - \{(2,2,2)\}$, i.e. of form $a = (a_1, a_2, a_3)$

$\mathbf{T}$ = {1,2,...} (treating this as a discrete time process, where for each time step baby i starts in state s_i, is made passive or active according to action a_i then transitions to state j_i according to its state and active/passive status).

Rewards: $r((s_1, s_2, s_3), (a_1, a_2, a_3) = \sum_i (r(s_i) + r(a_i))$,
where $r(s_i) = -\mathbf{1}(s_i = 3)$, $\quad r(a_i) = -\mathbf{1}(a_i = 2)$

Transition probabilities: $\mathbb{P}((j_1, j_2, j_3)|(s_1, s_2, s_3), (a_1, a_2, a_3)) = \prod_i \mathbb{P}(j_i|s_i, a_i) = \prod_i P^{a_i}_{j_i, s_i}$ (as babies are independent of each other, can use product rule).

Some possible policies:

Never feed $\quad \pi_{never} : d = (1,1,1)\ \forall s$

Feed when screaming $\quad \pi_{scream} : d = \begin{cases} (a_1, a_2, a_3) : a_i = \begin{cases} 1 & s_i \neq 3 \\ 2 & s_i = 3 \end{cases} & s \neq (3,3,3) \\ (2,2,1) & s = (3,3,3) \end{cases}$

Alternate feeding $\pi_{alt} = ((2,1,1), (1,2,1), (1,1,2), \dots)$ i.e. feed one baby each time, swapping each time.

Running these over 10,000 steps (see `bandit.R`) gives the following average costs per step: $cost(\pi_{never}) = 1.6874$, $cost(\pi_{scream}) = 1.9934$, $cost(\pi_{alt}) = 2.4343$.

Solving the MDP (with the value iteration algorithm in the R package `MDPtoolbox`) shows that π_{never} is actually the optimal policy. By lowering the penalty for feeding relative to the screaming penalty, more interesting policies become optimal. For example, for cost = number screaming + 0.6× number feeding, the *feed when scream* policy is optimal, while for cost = number screaming + 0.6× number feeding, the optimal policy is to feed a bandit whenever it isn't sleeping (subject to only two at a time feeding, prioritising the screaming bandits).

```
rm(list =ls ())
library("MDPtoolbox")
library("combinat")

###############SET UP#################################
q=0.75
T <- 9
L <- c(0:3)   #levels
taumax <- c(9-L*2)  #max tau possible for each level (9 - 2*#promotions needed)

##enumerating possible states##
S <- data.frame()
for (l in 0:2){
  for (t in c(0,2:taumax[l+1])){
    S <- rbind(S,c(l,t,0))
  }
  for (t in 1:(taumax[l+1]-1)){
    S <- rbind(S,c(l,t,1))
}}
for (t in 0:(taumax[4])) {
  S <- rbind(S,c(3,t,0))
}
names(S) <- c("lev","tau","elig")
N <- nrow(S)   #number of states
S$stnum <- 1:N
S$stlab <- paste0(S$lev,S$tau,S$elig)
attach(S)

##set up transition probs - mostly will be zeroes##
P <- array(data=0,dim=c(N,N,2),dimnames=list(S$stlab,S$stlab,c("wait","apply")))
#wait action, set all eligible  P to 1
#non-terminals
temp1 <- S[tau<taumax[lev+1]-1 & lev!=3,]
ind1 <-
array(c(temp1$stlab,paste0(temp1$lev,temp1$tau+1,1),rep("wait",nrow(temp1))),dim=c(nrow(temp1),3))
#one before terminals (can't move to l,tmax,1)
temp2 <- S[(lev==3& tau<taumax[lev+1])|(lev!=3 & tau==taumax[lev+1]-1),]
ind2 <-
array(c(temp2$stlab,paste0(temp2$lev,temp2$tau+1,0),rep("wait",nrow(temp2))),dim=c(nrow(temp2),3))
#only self loops for ends
tempT <- S[tau==taumax[lev+1],]
ind3 <- array(c(tempT$stlab,tempT$stlab,rep("wait",nrow(tempT))),dim=c(nrow(tempT),3))
P[rbind(ind1,ind2,ind3)] <- 1
#apply action
temp3 <- S[elig==1,]   #can apply
indS <- array(c(temp3$stlab,paste0(temp3$lev+1,0,0),rep("apply",nrow(temp3))),dim=c(nrow(temp3),3))
indF <-
array(c(temp3$stlab,paste0(temp3$lev,temp3$tau+1,0),rep("apply",nrow(temp3))),dim=c(nrow(temp3),3))
P[indS] <- 1-q^temp3$tau
P[indF] <- q^temp3$tau

#check row sums add up to 1
apply(P[,,1],FUN=sum,MARGIN=1)
apply(P[,,2],FUN=sum,MARGIN=1)

##reward matrix, r((lev,tau,elig))=lev for all states (lev,taue,elig), all actions
R <- array(S$lev,dim=c(N,2),dimnames=list(S$stlab,c("wait","apply")))

##check MDP##
mdp_check(P,R)  #woohoo OK

#######find optimal policy with MDPtoolbox#####
print(MDP.prom <- mdp_finite_horizon(P, R, 1, 10))
print(cbind(S,MDP.prom$policy))

###############generate policies##########
#policies <- list(ifelse(S$elig==1,"apply","wait"), ifelse(S$tau >= 3 & S$elig==1,"apply","wait"),
MDP.prom$policy)
#names(policies) <- c("ASAP","tau3","best")
policies <- list(ifelse(S$elig==1,"apply","wait"), ifelse(S$tau >= 3 & S$elig==1,"apply","wait"))
names(policies) <- c("ASAP","tau3")

##### PART C: simulate MC performance ##########
trials <- 1000
r <- array(0,dim=c(length(policies),trials))
for (pol in 1:length(policies)) {
policy <- policies[[pol]]
for (i in 1:trials) {
  s <- 1
  #cat("init state=",S[s,]$stlab,"\n")
  for (t in 0:(T-1)) {
    a <- policy[s]
    r[pol,i] <- r[pol,i] + R[s,a]
    s <- sample(1:N,1,prob=P[s,,a])
    #cat("t=",t," a=",a," r=",r[i],"new s=",S[s,]$stlab,"\n")
  }
  r[pol,i] <- r[pol,i] + R[s,a]
  #cat("t=9, total value = ",r[i],"\n")
}
cat("policy = ",names(policies)[pol], "avg value=",mean(r[pol,]),"\n")
}
#plot means and conf intervals
plotmeans(c(r[1,],r[2,]) ~ c(rep(1,trials),rep(2,trials)),connect=FALSE,xlab="policy",ylab="mean
value",main="Comparing policies: Monte Carlo simulation
(n=1000)",mean.labels=TRUE,legend=c(expression(pi[1]),expression(pi[3])),pch=".",n.label=FALSE,gap=0.5)


############PART D: policy evaluation algorithm##########
for (pol in 1:length(policies)) {
  policy <- policies[[pol]]

  U <- array(data=0,dim=c(N,10),dimnames=list(S$stlab,c(1:10)))

  for (s in 1:N) {   #t=N
    U[s,10] <- R[s,1]
  }
  for (t in 9:1) {
    for (s in 1:N) {
      U[s,t] <- (R[s,policy[s]] + sum(P[s,,policy[s]]*U[,t+1]))
    }
  }
  cat("policy = ",names(policies)[pol],", value of policy = ",U[1,1],"\n")
}


###########PART E: all stationary policies######
###set up policies

#make list of decision nodes (states with elig=1)
dec.nodes <- S[S$elig==1,]$stnum
#all 2^18 combinations of wait/apply for 18 decision states
dec.pols <- hcube(rep(2,length(dec.nodes)),1)
colnames(dec.pols) <- as.numeric(dec.nodes)
#set action for all others (elig=0) to 1 (wait)
other.nodes <- array(data=1,dim=c(nrow(dec.pols),N-ncol(dec.pols)))
colnames(other.nodes) <- as.numeric(S[-dec.nodes,]$stnum)
#combine and sort, now have 2^18 policies, with decision for each state
all.policies <- cbind(dec.pols,other.nodes)
all.policies <- all.policies[,order(as.numeric(colnames(all.policies)))]

#run policy eval alg
V <- rep(0,nrow(all.policies))
names(V) <- c(1:nrow(all.policies))
for (pol in 1:nrow(all.policies)) {
  policy <- all.policies[pol,]

  U <- array(data=0,dim=c(N,10),dimnames=list(S$stlab,c(1:10)))

  for (s in 1:N) {   #t=N
    U[s,10] <- R[s,1]
  }
  for (t in 9:1) {
    for (s in 1:N) {
      U[s,t] <- (R[s,policy[s]] + sum(P[s,,policy[s]]*U[,t+1]))
    }
  }
  #cat("policy = ",names(policies)[pol],", value of policy = ",U[1,1],"\n")
  V[pol] <- U[1,1]
}
max(V)   #value of optimal policy
length(which(V==max(V)))  #how many policies are optimal?
opt.pol <- all.policies[which(V==max(V)),]   #collect all
colnames(opt.pol) <- S$stlab
apply(opt.pol,2,FUN=mean)   #check which states have same decision in all optimal policies (ones that
vary don't matter)
apply(opt.pol,2,FUN=mean)[S$elig==1 & apply(opt.pol,2,FUN=mean) != 1.5] #only 9 of the 18 'decision
states' have actions relevant to opt. policy, all have action=apply
```

```
cat("\014")
rm(list =ls ())
library("MDPtoolbox")

feedcost = -.5
screamcost = -1

#transition probs for each individual baby
p.act <- matrix(c(0.6, 0.1, 0.3, 0.5, 0.2, 0.3, 0.3, 0.6, 0.1), 3, 3, byrow=TRUE)
p.pas <- matrix(c(0.6, 0.1, 0.3, 0.1, 0.3, 0.6, 0.2, 0.1, 0.7), 3, 3, byrow=TRUE)
p <- list(p.pas,p.act)

#actions: of form (a1,a2,a3), where ai=1:baby i is passive, ai=2:baby i is active)
A <- expand.grid(a1=1:2,a2=1:2,a3=1:2)[1:7,]     #don't include (2,2,2) as only 2 babies feeding at
once permitted
A$reward <- apply(A,1,function(x) feedcost*sum(A[x,]==2))  #cost for taking this action (from any
state) i.e. # babies feeding

#states: of form (s1,s2,s3) where si=1:baby i is sleeping, si=2:baby i is content, si=3:baby i is agro
S <- expand.grid(s1=1:3,s2=1:3,s3=1:3)
S$reward <- apply(S,1,function(x) screamcost*sum(S[x,]==3))  #cost for being in this state (for any
action) i.e. #babies in state 3
N <- nrow(S)

#reward matrix R[s,a] - rewards not destination dependent
R <- outer(S$reward,A$reward,function(x,y) x+y)     #cost of feeding + cost of screaming for each s,a
combination

#transition matrix for combined MC, P[s_i,s_j,a]
P <- array(0, dim=c(N,N,nrow(A)))
for (s in 1:nrow(A)){
  for (i in 1:N){
    for (j in 1:N){
      #as babies indep, P[(s1_i,s2_i,s3_i),(s1_j,s2_j,s3_j)|(a1,a2,a3)] =
p(s1_i,s1_j|a1)*p(s2_i,s2_j|a2)*p(s3_i,s3_j|a3)
      P[i,j,s] <- p[[A[s,1]]][S[i,1],S[j,1]]*p[[A[s,2]]][S[i,2],S[j,2]]*p[[A[s,3]]][S[i,3],S[j,3]]
}}}

#some policies to test:
policy.neverfeed <- rep(1,N)  #don't feed ever
policy.alternatefeed <- rep(c(2,3,5),9)  #feed one baby each time, alternating
policy.feedwhenscream <- c(1,1,2,1,1,2,3,3,4,1,1,2,1,1,2,3,3,4,5,5,6,5,5,6,7,7,7)  #feed as many
screaming babies as possible

######simulate cost of MC given policy
SIMMDP <- function(policy,policyname) {
steps <- 10000
s <- 1 #starting state
r <- 0   #reward at start
#cat("s_0 =", s)
for (t in 1:steps){
  a <- policy[s]
  r <- r + R[s,a]
  s <- sample(1:N,1,prob=P[s,,a])
  #cat("t=",t," a=",a," r=",r, "s=",s)
}
r <- r + R[s]
cat("r_n = ",r)
cat("policy: ",policyname,",", "avg cost = ",r/steps)
}

SIMMDP(policy.neverfeed,"never feed")
SIMMDP(policy.alternatefeed,"alternate feed")
SIMMDP(policy.feedwhenscream,"feed when scream")

###plot means and conf intervals
plotmeans(c(r[1,],r[2,]) ~ c(rep(1,trials),rep(2,trials)),connect=FALSE,xlab="policy",ylab="mean
value",main="Comparing policies: Monte Carlo simulation
(n=1000)",mean.labels=TRUE,legend=c(expression(pi[1]),expression(pi[3])),pch=".",n.label=FALSE,gap=0.5)


#####for comparison, using MDPtoolbox:
#finding optimal policy:
print(MDPresult <- mdp_value_iteration(P,R,discount=1,epsilon=0) )
policy.best <- MDPresult$policy

#evaluate various policies (gives much the same results as sim above):
cat("best policy, avg cost:",mean(mdp_eval_policy_iterative(P, R,
discount=1,policy=policy.best,max_iter=10000,V0=rep(0,N),epsilon=0))/10000)
cat("never feed, avg cost:",mean(mdp_eval_policy_iterative(P, R,
discount=1,policy=policy.neverfeed,max_iter=10000,V0=rep(0,N),epsilon=0))/10000) #opt for feedcost=-1
cat("alternate feed, avg cost:",mean(mdp_eval_policy_iterative(P, R,
discount=1,policy=policy.alternatefeed,max_iter=10000,V0=rep(0,N),epsilon=0))/10000)
cat("feed when scream, avg cost:",mean(mdp_eval_policy_iterative(P, R,
discount=1,policy=policy.feedwhenscream,max_iter=10000,V0=rep(0,N),epsilon=0))/10000 ) #opt for
feedcost=-0.6
#for feedcost=-0.5, feed whenever awake or scream is opt

#show optimal policy in terms of action for each baby in each state
showpol <- cbind(S[1:3],A[policy.best,1:3])
rownames(showpol) <- rownames(S)
showpol
```