MATH4406 - Assignment 2

Sean Watson - 42613512

1 Exercise 3.2

A generalised model of the inventory model.

1.a

Formulation of the MDP.

Decision Epochs As with the original model: $T = \{1, 2, ..., N\} \quad N < \infty$

States The amount of stock in inventory: $S = \{..., -2, -1, 0, 1, 2, ...M\}$. Recall that M is the maximum capacity of the warehouse. A negative state indicates that there are orders waiting to be filled from previous months.

Actions The amount of stock to order in month t, this is the same as the original model. $A_s = \{0, 1, ..., M - s\}$. Note that the maximum amount that can be ordered is equal to the capacity minus the current stock in inventory, so if there are unfilled orders we can still order more than the capacity of the warehouse.

Expected Rewards Expected revenue minus ordering, holding and backlogged costs. Similar to the original model but with the backlogging cost included:

$$r_t(s, a) = F - O(a) - h(s + a) - b(s + a), \qquad t = 1, 2, ..., N - 1$$

with *O, h* and b as defined in the original model. For the revenue we define as:

$$F = \sum_{j=0}^{\infty} p_j f(j)$$

This is needed because the new model no longer has the option of having more demand than inventory to meet it. So the first term of the revenue is the same but the second is no longer necessary. $f(j)$ is the present value of the revenue gained for selling j units. This means that the expected revenue is constant as all orders received can be taken even if there is insufficient stock to fill them.

The value of terminal inventory remains the same: $r_N(s) = g(s) \qquad$ *t=N.*

Transition Probabilities

$$p_t(j|s, a) = \begin{cases} p_{s+a-j} & \text{if } j \le s + a \le M \\ 0 & \text{otherwise} \end{cases}$$

p_j is the probability of j new orders arriving so p_{s+a-j} is the probability of $s + a - j$ orders arriving so essentially the probability of moving to state j given state s and action a. Due to the allowance for backlogging orders the first transition probability simply means that we can move to any state with less stock then we currently have (given the relative probabilities). The second probability encompasses the possibility of having more stock than the maximum capacity and "negative" orders i.e. $j > M$ and $j > s+a$. Clearly neither of these is possible.

1.b

Calculate $r_t(s,a)$ and $p_t(j|s,a)$ for the above model with $b(u) = 3u$ and all other model parameters identical to those in section 3.2.2.

$K = 4, c(u) = 2u, g(u) = 0, h(u) = u, b(u) = 3u, M = 3, N = 3, f(u) = 8u$ and:

$$p_j \begin{cases} \frac{1}{4} & \text{if } j = 0 \\ \frac{1}{2} & \text{if } j = 1 \\ \frac{1}{4} & \text{if } j = 2 \end{cases}$$

Due to fact that we can backlog orders we always expect to the same amount of revenue regardless of our current stock:

u	*F(u)*
0	$0 \times \frac{1}{4} + 8 \times \frac{1}{2} + 16 \times \frac{1}{4} = 8$
1	$0 \times \frac{1}{4} + 8 \times \frac{1}{2} + 16 \times \frac{1}{4} = 8$
2	$0 \times \frac{1}{4} + 8 \times \frac{1}{2} + 16 \times \frac{1}{4} = 8$
3	$0 \times \frac{1}{4} + 8 \times \frac{1}{2} + 16 \times \frac{1}{4} = 8$

The expected rewards and transition probabilities are given by:

$$r_t(s,a) = \begin{cases} 8 - (4 + 2a) + 3s & \text{if } s < 0 \text{ and } s + a \leq 0 \\ 8 - 3s & \text{if } s < 0 \text{ and } a = 0 \\ 8 - (4 + 2a) - (s + a) + 3s & \text{if } s < 0 \text{ and } s + a \geq 0 \\ 8 - (4 + s + 3a) & \text{if } s \geq 0 \text{ and } a > 0 \\ 8 - s & \text{if } s \geq 0 \text{ and } a = 0 \end{cases}$$

	$p_t(j\|s,a)$									
$s+a$ $\quad j$	-∞	-$\infty+1$	-$\infty+2$	$\cdots$	-2	-1	0	1	2	3
-$\infty+2$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
-2	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0	0	0
-1	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0	0
0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0
1	0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0
2	0	0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0
3	0	0	0	0	0	0	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$

1.c

Formulate the model where the payment is given at the end of the month in which the order is filled.

The original value for $F(s+a)$ was given by (note that the q parameter was replaced with its definition to simplify things):

$$F(s+a) = \sum_{j=0}^{s+a-1} f(j)p_j + f(s+a)\sum_{j=s+a}^{\infty} p_j$$

In the new model we have several cases based on how many orders we are filling. So we have 3 cases: we have no backlogged orders and meet demand for all new orders (with or without surplus), we have backlogged orders but we order enough stock to meet demand, and there remain backlogged orders regardless of new stock. The first of these conditions results in the original definition of the revenue since this is clearly the same problem as in the original model. The second is given by the fact that if inventory exceeds demand then obviously the amount sold is $f(j-s)$ that is the new orders plus the backlogged ones are all filled. The other term, however, is that demand exceeds inventory, this can only occur for the additional stock ordered above s because ordering sufficient stock means we can fill all the s backlogged orders regardless of whether new orders come in. Finally the third obviously means that we can only meet those orders based upon the additional stock we ordered, so only $f(a)$.

$$F(s+a) = \begin{cases} \displaystyle\sum_{j=0}^{s+a-1} f(j)p_j + f(s+a)\sum_{j=s+a}^{\infty} p_j & \text{if } s+a \geq s \geq 0 \\ \displaystyle\sum_{j=0}^{s+a-1} f(j-s)p_j + f(a)\sum_{j=s+a}^{\infty} p_j & \text{if } s+a \geq 0 \geq s \\ f(a) & \text{if } s+a < 0 \end{cases}$$

This gives the expected rewards as:

$$r_t(s,a) = F(s+a) - O(a) - h(s+a) - b(s+a)$$

2 Exercise 3.21

Provide system equations and an MDP formulation for the modified version of 3.7.1 with 2 parallel servers with their own queues. The controller allocates jobs from the potential job queue to either sub-queue or none. There are holding costs, $h_i(x)$, for the number of jobs at each server's queue and a reward, R, for completing a job. Provide a guess regarding the most efficient operating policy.

We will model the problem in much the same way as in section 3.7.1:
Z_t - the number of arrivals to the potential job queue, $Z_t \geq 0$
V_t - the content of the potential job at epoch t
X_{it} - the number of jobs in server i's queue
Y_{it} - possible service completions for server i
u_{it} - number of arrivals admitted from the potential job queue to server i's queue

This gives rise to the system equations:
$X_{it+1} = X_{it} + u_{it} + Y_{it}$, for $i = 1, 2$
$V_{t+1} = Z_t$
$0 \leq u_{1t} + u_{2t} \leq V_t$

The probabilities of completions for each server is given by $f_i(n) = \mathbb{P}(Y_{it} = n)$
The number of arrivals to the potential job queue is given by $g(n) = \mathbb{P}(Z_t = n)$

Decision Epochs $T = \{0, 1, 2, ..., N\}$

States $S = \{0, 1, ...\} \times \{0, 1, ...\} \times \{0, 1, ...\} = S_1 \times S_2 \times S_3$ where S_1 and S_2 are the number of jobs in the queue for servers 1 and 2 and S_3 is the number of jobs in the potential job queue.

Actions The possible actions are how many jobs to admit to each server. $A_{s_1,s_2,s_3} = \{0, 1, ..., s_3\} \times \{0, 1, ..., s_3\}$ where action (a, b) is such that $a + b \leq s_3$.

Rewards $r_t(s_1, s_2, s_3, a) = R\mathbb{E}\{\sum_{i=1,2} min(Y_{it}, s_i + a_i)\} - \sum_{i=1,2} h_i(s_i + a_i)$ where a_i is the number of jobs assigned to server i by action a.

Transition Probabilities

$$p_t(s_1', s_2', s_3'|s_1, s_2, s_3, a) = \begin{cases} 0 & \text{if } s_1' > a_1 + s_1 \geq 0 \text{ or } s_2' > a_2 + s_2 \geq 0 \\ \prod_{i=1,2} f_i(s_i + a_i - s_i')g(s_3') & \text{if } a_i + s_i > s_i' > 0 \text{ for } i = 1, 2 \\ g(s_3') \prod_{i=1,2} \sum_{j=s_i+a_i}^{\infty} f_i(j) & \text{if } s_i' = 0, a_i + s_i > 0 \text{ for } i = 1, 2 \\ g(s_3') & s_i' = a_i + s_i = 0 \text{ for } i = 1, 2 \\ \\ f_i(s_i + a_i - s_i') \sum_{k=s_j+a_j}^{\infty} f_j(k)g(s_3') & \text{if } a_i + s_i > s_i' > 0, \text{ and } s_j' = 0, a_j + s_j > 0, i \neq j \\ f_i(s_i + a_i - s_i')g(s_3') & \text{if } a_i + s_i > s_i' > 0 \text{ and } s_j' = a_j + s_j = 0, i \neq j \\ g(s_3') \sum_{k=s_i+a_i}^{\infty} f_i(k) & \text{if } s_i' = 0, a_i + s_i > 0 \text{ and } s_j' = a_j + s_j = 0, i \neq j \\ 0 & \text{otherwise} \end{cases}$$

The first case is fairly obvious in that there is no way that we can move to a state with more jobs given the current number in the queue after taking action a.

The second case is that both of the servers serve $s_i + a_i - s_i'$. Since the number of jobs completed by each server is independent the probabilities can be multiplied together.

The third case is that both servers complete all of the jobs in each of their individual queues.

The fourth case is that there are no jobs allocated to either queue and both were already empty. So obviously the resulting state is that there are no jobs in either queue still. The probability of moving to this state is dependent only on the number of jobs arriving.

The remaining cases are combinations of the above: i.e. one server completes all of their jobs and the other does not, one server has an empty queue and is not allocated any new jobs and the other either completes a portion or all of the jobs in their queue.

Optimal Policy The optimal policy would be to assign each server the number of jobs they are expected to complete at each time. Obviously this depends on how the arrivals are distributed as if there are only periodic arrivals it may be better to assign more/all of them.

3 Exercise 3.26

The problem deals with the the amount of food that an individual lion consumes based on when it should hunt and the size of the hunting pack that the lion should be a part of. The trade-off comes from the fact that a larger pack is more likely to result in a successful hunt but will leave less food to go around to each individual member. There are given probabilities of success for varying pack size: p(l) = 0.15, p (2) = 0.33, p(3) = 0.37, p(4) = 0.40, p (5) = 0.42, and p (≥6) = 0.43. If a hunt is successful then the pack gains 164kg of meat to share evenly with each pack member. Each day the lion loses 6kg if is sedentary and an additional 0.5kg if it hunts. The objective is to survive N days.

Decision Epochs $T = \{1, 2, ..., N\} \quad N < \infty$

States $S = \{0, 1, ..., 30\}$ the state relates directly to the amount of energy (meat) that each lion has in its stomach. If it has 0 then the lion is dead, if has 30 then it is at full capacity.

Actions $A_s = \{0, 1, ..., 6\}$. These are the number of lions in the pack, so between 0 and 6 lions. Note that it is possible to have more lions in the pack but they would be detrimental to the group's efforts as the probability of success would not increase whilst eating more meat. Additionally if the current state is less than 0.5 then the actions are limited to $A_s = \{0\}$ since the lion has insufficient energy to hunt.

Rewards The aim is to ensure that the lion survives for the entire time horizon. But we do not particularly care what state it is in at that point so $r_N(s) = 1$ if $s > 0$ and 0 otherwise i.e. at time T we must be alive and it doesn't matter how full we are or were in previous time steps.

Transition Probabilities

$$p_t(j|s,a) = \begin{cases} p(a) & \text{if } a > 0 \text{ and } j = min(s - 6.5 + \frac{164}{a}, 30) \\ 1 - p(a) & \text{if } a > 0 \text{ and } j = max(s - 6.5, 0) \\ 1 & \text{if } a = 0 \text{ and } j = max(s - 6, 0) \\ 0 & \text{otherwise} \end{cases}$$

The first of these probabilities results from whether there is a successful hunt or not, if so then we have the probability given by the number of animals in the pack (a). If we do hunt then we lose 6.5 energy, this is the energy for the day plus the energy required to hunt but if we succeed then we increase our energy by 164 divided by the number of lions in the pack. Limited of course by the capacity of our stomach.

The second probability is that the lion hunts, but fails. Thus taking the hit to the energy but not increasing it. Obviously in this case the limiting energy level is 0 since the lion dies at this point.

The third probability is that the lion does not hunt at all and loses energy for a day passing. If the given action is $a = 0$ this is guaranteed since it means the lion does not hunt.

The final probability is trivial - any other event cannot happen so the probability of it occurring is 0.

4 Promotion MDP

4.a

Decision Epochs $T = \{1, 2, ..., 10\}$ There are 10 time steps for a 10 year career.

States The state is the current year of the career - 1-10, the level - 0,1,2,3, how long we've been at the current level and whether we can apply or not. So $S = \{1, 2, ..., 10\} \times \{0, 1, 2, 3\} \times \{0, 1, ..., 9\} \times \{0, 1\}$. Note of course that depending on where we start our 10 year career some of these states are not actually possible.

Actions $A_s = \{0, 1\}$ where 0 denotes not applying for promotion and 1 denotes applying.

Transition Probabilities

$$p_t(j|s,a) = \begin{cases} q^{\tau_s} & \text{if } a = 1 \text{ and } j = (L_s, \tau_s + 1, 0) \\ 1 - q^{\tau_s} & \text{if } a = 1 \text{ and } j = (L_s + 1, 0, 0) \\ 1 & \text{if } a = 0 \text{ and } j = (L_s, \tau_s + 1, 1) \\ 1 & \text{if } L_s = 3 \text{ and } j = (L_s, \tau_s + 1, 1) \\ 0 & \text{otherwise} \end{cases}$$

Rewards The reward for being in our current state is simply the level we are at, there is no discount factor so the reward for being in a level is the same at any time: $r_t(s,a) = L_s$

The value function comprises of the rewards plus the expectation of future rewards given our current state, this is not written in detail since it seems that it was not called for.

4.b

The number of different policies is simply given by the number of decisions we can make raised to the power of the number of times that we can make a decision. Given that we always apply in the final year if we are able to (so no choices here), then we have 15 (there are 18 states where we can choose to apply or not) possible time options and 2 choices at each of them. So $2^{15} = 32768$ possible policies.

4.c

The code for this question can be found in the appendix. It was run using Python. It found results of approximately $v = 7.758$ for the policy of applying as early as possible and $v = 5.668$ for only applying after three years in the current level. So applying whenever we can is clearly a better policy, this is logical since if we were to apply as soon as possible in each level then even if we failed then we would still have the option to apply when $\tau = 3$, which is the time when we first apply for the other policy.

4.d

The Python code for this question can be found in the appendix. It found results of 7.78159 for the first policy of applying as often as possible and 5.59775 for only applying after three years in each level. These are very similar values to those in the previous question as expected.

4.e

The Python code can, again, be found in the appendix. This code produces all possible policies (possible states to apply in) and evaluates each individually. The code can be checked to be correct by inputting every state as being able to apply (when eligible to apply), this gives the original result given in the above subsection. In fact this code, without the loop, could be used in place of that in the original evaluation.

By looping and evaluating all of the policies we find that applying whenever we can is an optimal policy, and further that there are in fact 64 policies in total with the same expected reward. This means that any of these policies would in fact be optimal.

A Monte Carlo Code for the Promotion MDP

```
import $random$
#policy = 0 means apply as much as possible, anything else for $> 3$
def monte($policy, runs$) :
    $results = []$
    for r in range($runs$) :
        $q = 0.75$
        $state = (0, 0, 0)$
        $history = []$
#note the range is only 9 as although we have 10 years we only have 9 chances to act.
        for t in range(9) :
            $x = random.random()$
            if $state[2] == 0$ or $state[0] == 3$ :
                $state = (state[0], state[1] + 1, 1)$
#we can apply otherwise
            elif policy $== 0$ :
                if $x <= 1 - 0.75 * *(state[1])$ :
                    $state = (state[0] + 1, 0, 0)$
                else :
                    $state = (state[0], state[1] + 1, 0)$
            else :
                if $state[1] >= 3$ :
                    if $x <= 1 - 0.75 * *(state[1])$ :
                        $state = (state[0] + 1, 0, 0)$
                    else :
                        $state = (state[0], state[1] + 1, 0)$
                else :
                    $state = (state[0], state[1] + 1, 1)$
            history.append(state[0])
        results.append(sum(history))
    return sum(results)/float(runs)
```

B Policy Evaluation Algorithm Code for the Promotion MDP

```
$q = 0.75$
$known = \{\}$

#Policy 1: apply as soon as possible
def $V(t, s, tau, app)$ :
    if $(t, s, tau, app)$ not in $known$ :
        if $t == 10$ :
            $known[(t, s, tau, app)] = s$
        elif $s == 3$ :
            $known[(t, s, tau, app)] = (s) + V(t + 1, s, tau + 1, 1)$
        elif $app == 1$ :
            $known[(t, s, tau, app)] = (s) + (1 - q * *(tau)) * V(t + 1, s + 1, 0, 0)+$
                                   $q * *(tau) * V(t + 1, s, tau + 1, 0)$
        else :
            $known[(t, s, tau, app)] = (s) + V(t + 1, s, tau + 1, 1)$
    return $known[(t, s, tau, app)]$


#Policy 2: apply after tau >=3
$known2 = \{\}$
def $V2(t, s, tau, app)$ :
    if $(t, s, tau, app)$not in $known2$ :
        if $t == 10$ :
            $known2[(t, s, tau, app)] = s$
        elif $s == 3$ :
            $known[(t, s, tau, app)] = (s) + V(t + 1, s, tau + 1, 1)$
        elif $app == 1$ :
            if $tau >= 3$ :
                $known2[(t, s, tau, app)] = (s) + (1 - q * *(tau)) * V2(t + 1, s + 1, 0, 0)+$
                                        $q * *(tau) * V2(t + 1, s, tau + 1, 0)$
            else :
                $known2[(t, s, tau, app)] = (s) + V2(t + 1, s, tau + 1, 1)$
        else :
            $known2[(t, s, tau, app)] = (s) + V2(t + 1, s, tau + 1, 1)$
    return $known2[(t, s, tau, app)]$
```

C All Policy Evaluations

```
import itertools

$q = 0.75$

$states = [(0, t, 1)$ for t in range$(1, 8)] + [(1, t, 1)$ for $t$ in range$(1, 6)] + [(2, t, 1)$ for $t$ in range$(1, 4)]$
$x = $ list(itertools.product($states, [0, 1]$))
$y = []$
for $i$ in range(len($x$))[:: 2] :
    $y$.append($[x[i], x[i + 1]]$)
pols = list(itertools.product($*y$))

def $V(t, s, tau, app)$ :
    $known = \{\}$
    if $(t, s, tau, app)$ not in known:
        if $t == 10$ :
            $known[(t, s, tau, app)] = s$
        elif $s == 3$ :
            $known[(t, s, tau, app)] = (s) + V(t + 1, s, tau + 1, 1)$
        elif $((s, tau, app), 1)$ in $p$ :
            $known[(t, s, tau, app)] = (s) + (1 - q * *(tau)) * V(t + 1, s + 1, 0, 0)+$
                                        $q * *(tau) * V(t + 1, s, tau + 1, 0)$
        elif $(s, tau, app) == (0, 8, 1)$ or $(s, tau, app) == (1, 6, 1)$ or $(s, tau, app) == (2, 4, 1)$ :
            $known[(t, s, tau, app)] = (s) + (1 - q * *(tau)) * V(t + 1, s + 1, 0, 0)+$
                                        $q * *(tau) * V(t + 1, s, tau + 1, 0)$
        else:
            $known[(t, s, tau, app)] = (s) + V(t + 1, s, tau + 1, 1)$
    return $known[(t, s, tau, app)]$

result = {}
for p in pols :
    result[p] = $V(1, 0, 0, 0)$
#to find the policy which gives the highest value
p = max(result, key=result.get)
print result[$p$]
```