MATH4406 - Assignment 3

Sean Watson - 42613512

1 Inventory Control

The problem done in this section has the following parameters: $K = 3$, $c(u) = u$, $g(u) = 0$, $h(u) = u$, $M = 4$, $f(u) = 10u$ and $p_j = (1/5, 1/5, 2/5, 1/5, 0)$. And tables for the expected revenue, transition probabilities and rewards are given below:

Table 1: Revenue

u	$F(u)$
0	0
1	8
2	14
3	16
4	14

Table 2: $r_t(s, a)$

s	a=0	1	2	3	4
0	0	3	7	7	5
1	7	8	8	6	×
2	12	9	7	×	×
3	13	8	×	×	×
4	12	×	×	×	×

Table 3: $p_t(j|s, a)$

s+a	j=0	1	2	3	4
0	1	0	0	0	0
1	4/5	1/5	0	0	0
2	3/5	1/5	1/5	0	0
3	1/5	2/5	1/5	1/5	0
4	0	1/5	2/5	1/5	1/5

Using the same format for the backwards induction as in Puterman:

1. Set $t = 5$ and $u_5^*(s) = r_5(s) = 0, s = 0, 1, 2, 3, 4$.
2. Set $t = 4$ and

$$u_4^*(s) = \max_{a \in A_s} \left\{ r(s, a) + \sum_{j \in S} p(j|s, a) u_5^*(s) \right\}, \qquad \forall s \in S$$

$$= \max_{a \in A_s}\{r(s,a)\}$$

Which gives the values of:

s	$u_4^*(s)$	$A_{s,3}^*$
0	7	3
1	8	2
2	12	0
3	13	0
4	12	0

3. Set $s = 3$ and

$$u_3^*(s) = \max_{a \in A_s}\{u_3^*(s,a)\}$$

Where for example:

$$\begin{aligned} u_3^*(1,3) &= r(1,3) + p(0|1,3)u_4^*(0) + p(1|1,3)u_4^*(1) + p(2|1,3)u_4^*(2) + p(3|1,3)u_4^*(3) + p(4|1,3)u_4^*(4) \\ &= 6 + 0 \times 7 + \frac{1}{5} \times 8 + \frac{2}{5} \times 12 + \frac{1}{5} \times 13 + \frac{1}{5} \times 12 \\ &= 17.4 \end{aligned}$$

The table below gives all of the values for $u_3^*(s,a), u_3^*(s)$ and $A_{s,3}^*$, No further individual calculations will be provided since it is assumed that they are unnecessary as long as each value is given.

Table 4: $u_3^*(s,a)$

s	a=0	a=1	a=2	a=3	a=4	$u_3^*(s)$	$A_{s,3}^*$
0	7	10.2	15.2	16.6	16.4	16.6	3
1	14.2	16.2	17.6	17.4	×	17.6	2
2	20.2	18.6	18.4	×	×	20.2	0
3	22.6	19.4	×	×	×	22.6	0
4	23.4	×	×	×	×	23.4	0

4. Set $t = 2$ and continue as previously:

Table 5: $u_2^*(s,a)$

s	a=0	a=1	a=2	a=3	a=4	$u_2^*(s)$	$A_{s,2}^*$
0	16.6	19.8	24.52	25.92	25.8	25.92	3
1	23.8	25.52	26.92	26.8	×	26.92	2
2	29.52	27.92	27.8	×	×	29.52	0
3	31.92	28.8	×	×	×	31.92	0
4	32.8	×	×	×	×	32.8	0

5. Set $t = 1$ and continue as previously:

Table 6: $u_1^*(s,a)$

s	a=0	a=1	a=2	a=3	a=4	$u_1^*(s)$	$A_{s,1}^*$
0	25.92	29.12	33.84	35.24	35.136	35.24	3
1	33.12	34.84	36.24	36.136	×	36.24	2
2	38.84	37.24	37.136	×	×	38.84	0
3	41.24	38.136	×	×	×	41.24	0
4	42.136	×	×	×	×	42.136	0

6. Since $t = 1$ stop.

The algorithm yields the expected total reward function v_4^* and the optimal policy $\pi^* = (d_1^*(s), d_2^*(s), d_3^*(s), d_4^*(s))$ as shown below. It is important to note that in $d_4^*(s)$ both states 0 and 1 had two policies of equal value though for convenience's sake it is easiest to consider them as 3 and 2 respectively to give a stationary policy.

s	$d_1^*(s)$	$d_2^*(s)$	$d_3^*(s)$	$d_4^*(s)$	$v_4^*(s)$
0	3	3	3	2/3	35.24
1	2	2	2	1/2	36.24
2	0	0	0	0	38.84
3	0	0	0	0	41.24
4	0	0	0	0	42.136

The optimal policy is to order stock up to 3 units if the stock falls below 2 units. It is a stationary (σ, Σ) policy given by:

$$d^*(s) = \begin{cases} 0 & s > 1 \\ 2 & s = 1 \\ 3 & s = 0 \end{cases}$$

2 Threshold Policy in inventory control

The Python code for this question can be found in the appendix, it is fully generalised in that given a probability vector and other parameters it can construct the rewards and transition probabilities before moving on to the induction itself. It was tested first on the example in the textbook (relevant input is induct1(4,3,[0.25,0.5,0.25,0],8,1,2,4)) and then the example given in the previous question to ensure that it was correct.

2.1 10 Threshold Policy Problems

Since it is unclear just how different each problem has to be, each problem varies only by one or two parameters from the last. $N = 15$ and $g(u) = 0$ in all cases.

1. $M = 3$, $p_j = (0.25, 0.5, 0.25, 0)$, $f(u) = 8u$, $h(u) = u$, $c(u) = 2u$, $K = 4$
2. $M = 3$, $p_j = (0.25, 0.5, 0.25, 0)$, $f(u) = 12u$, $h(u) = u$, $c(u) = 2u$, $K = 4$
3. $M = 3$, $p_j = (0.25, 0.5, 0.25, 0)$, $f(u) = 10u$, $h(u) = 2u$, $c(u) = 2u$, $K = 4$
4. $M = 3$, $p_j = (0.25, 0.5, 0.25, 0)$, $f(u) = 15u$, $h(u) = 3u$, $c(u) = 3u$, $K = 4$

5. $M = 4$, $p_j = (1/5, 1/5, 2/5, 1/5, 0)$, $f(u) = 12u$, $h(u) = 2u$, $c(u) = 3u$, $K = 4$
6. $M = 4$, $p_j = (1/5, 1/5, 2/5, 1/5, 0)$, $f(u) = 8u$, $h(u) = 2u$, $c(u) = u$, $K = 6$
7. $M = 4$, $p_j = (1/5, 1/5, 2/5, 1/5, 0)$, $f(u) = 8u$, $h(u) = 2u$, $c(u) = u$, $K = 2$
8. $M = 4$, $p_j = (2/5, 0, 2/5, 1/5, 0)$, $f(u) = 12u$, $h(u) = 2u$, $c(u) = 2u$, $K = 2$
9. $M = 4$, $p_j = (2/5, 0, 2/5, 1/5, 0)$, $f(u) = 6u$, $h(u) = u$, $c(u) = 2u$, $K = 2$
10. $M = 4$, $p_j = (1/5, 2/5, 0, 2/5, 0)$, $f(u) = 8u$, $h(u) = 2u$, $c(u) = u$, $K = 5$

The code was then run for each of these problems, with the additional command to print the optimal policy choices at each time (see the bottom of the appendix). From this output the following decision rules were made for each problem, as well as noting the relevant (σ, Σ) policies where if the inventory level falls below σ we order Σ stock.

Problem 1

$$d^*_{15}(s) = \begin{cases} 0 & s \in S \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{14}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 3$.

$$d^*_{t \neq 15,14}(s) = \begin{cases} 0 & s > 0 \\ 3 & s = 0 \end{cases}$$

Problem 2

$$d^*_{15,14}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 3$.

$$d^*_{t \neq 15,14}(s) = \begin{cases} 0 & s > 0 \\ 3 & s = 0 \end{cases}$$

Problem 3

$$d^*_{15}(s) = \begin{cases} 0 & s \in S \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{t \neq 15}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

Problem 4

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 1$.

$$d^*_{15}(s) = \begin{cases} 0 & s > 0 \\ 1 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{t\neq 15}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

Problem 5
The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{15}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 2, \Sigma = 3$.

$$d^*_{t\neq 15}(s) = \begin{cases} 0 & s > 1 \\ 2 & s = 1 \\ 3 & s = 0 \end{cases}$$

Problem 6

$$d^*_{15}(s) = \begin{cases} 0 & s \in S \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 3$.

$$d^*_{t\neq 15}(s) = \begin{cases} 0 & s > 0 \\ 3 & s = 0 \end{cases}$$

Problem 7
The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{15}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 2, \Sigma = 2$.

$$d^*_{t\neq 15}(s) = \begin{cases} 0 & s > 1 \\ 1 & s = 1 \\ 2 & s = 0 \end{cases}$$

Problem 8
The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{15}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 2, \Sigma = 3$.

$$d^*_{t\neq 15}(s) = \begin{cases} 0 & s > 1 \\ 1 & s = 1 \\ 2 & s = 0 \end{cases}$$

Problem 9

$$d^*_{15}(s) = \begin{cases} 0 & s \in S \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 2$.

$$d^*_{14}(s) = \begin{cases} 0 & s > 0 \\ 2 & s = 0 \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 2, \Sigma = 2$.

$$d^*_{t \neq 15,14}(s) = \begin{cases} 0 & s > 1 \\ 1 & s = 1 \\ 2 & s = 0 \end{cases}$$

Problem 10

$$d^*_{15}(s) = \begin{cases} 0 & s \in S \end{cases}$$

The following is a (σ, Σ) policy with $\sigma = 1, \Sigma = 3$.

$$d^*_{t \neq 15}(s) = \begin{cases} 0 & s > 0 \\ 3 & s = 0 \end{cases}$$

Clearly each of these are threshold policies as expected.

2.2 Non-Threshold Policy

A suitable ordering cost for a non-threshold policy would be:

$$O[a] = K + c^a$$

This revision was added to the code and tested for several different problems to verify that it does result in non-threshold policies. The resulting optimal decisions are displayed for Problem 8 from the previous question (parameters were $M = 4$, $p_j = (2/5, 0, 2/5, 1/5, 0)$, $f(u) = 12u$, $h(u) = 2u$, $c(u) = u$, $K = 2$).

$$d^*_{15,14,13}(s) = \begin{cases} 0 & s > 3 \\ 1 & \text{otherwise} \end{cases}$$

$$d^*_{t \neq 15,14,13}(s) = \begin{cases} 0 & s = 4 \\ 1 & \text{otherwise} \end{cases}$$

Ordering of stock is fairly consistent regardless of the current level, and this is no surprise given the large increase required to order additional stock. So obviously for any t this is not a threshold policy as there is no particular level Σ that we order up to.

3 Optimal Markov Deterministic Policies

Given that S is finite or countable and A_s is finite for each $s \in S$ then we can show that there exists an a' such that:

$$r_t(s,a') + \sum_{j \in S} p_t(j|s_t,a') u^*_{t+1}(h_t,a',j) = sup_{a \in A_{s_t}} \{r_t(s,a) + \sum_{j \in S} p_t(j|s_t,a) u^*_{t+1}(h_t,a,j)\}$$

with $u^*_t(h_t,a',j)$ satisfying the optimality equations:

$$u^*_t(h_t) = sup_{a \in A_s} \{r_t(s_t,a) + \sum_{j \in S} p_t(j|s_t,a) u_{t+1}(h_t,a,j)\}$$

for $t = 1, ..., N-1$ and $h_t = (h_{t-1}, a_{t-1}, s_t) \in H_t$. With boundary conditions given by:

$$u_N(h_N) = r_N(s_N)$$

Now we can show that for each t, $u^*_t(h_t)$ depends on h_t only through s_t through induction. Since $u^*_N(h_N) = u^*_N(h_{N-1}, a_{N-1}, s) = r_N(s)$ for all $h_{N-1} \in H_{N-1}$ and $a_{N-1} \in A_{s_{N-1}}$, $u^*_N(h_N) = u^*_N(s_N)$. Assume that the claim is valid for $n = t+1, ..., N$ Then

$$u^*_t(h_t) = sup_{a \in A_{s_t}} \{r_t(s,a) + \sum_{j \in S} p_t(j|s_t,a) u^*_{t+1}(h_t,a,j)\}$$

Which by the induction hypothesis gives

$$u^*_t(h_t) = sup_{a \in A_{s_t}} \{r_t(s,a) + \sum_{j \in S} p_t(j|s_t,a) u^*_{t+1}(j)\}$$

Since the terms inside the brackets depend on h_t only through s_t, we have proved that $u^*_t(h_t)$ depends on h_t only through s_t holds for all t.

Now we can prove that for any $\epsilon > 0$ there exists an ϵ-optimal policy which is deterministic and Markov: First we choose some $\epsilon > 0$, and let $\pi^\epsilon = (d^\epsilon_1, d^\epsilon_2, ..., d^\epsilon_{N-1})$ be any policy in Π^{MD} satisfying

$$r_t(s_t, d^\epsilon_t(s_t)) + \sum_{j \in S} p_t(j|s_t, d^\epsilon_t(s_t)) u^*_{t+1}(j) + \frac{\epsilon}{N-1} \geq sup_{a \in A_{s_t}} \{r_t(s,a) + \sum_{j \in S} p_t(j|s_t,a) u^*_{t+1}(j)\}$$

Then by the claim proved previously, π^ϵ satisfies the requirements necessary to be ϵ-optimal.

Finally we can prove the claim that there must now exist an optimal Markov deterministic policy: first note that according to Theorem 4.3.3 in the textbook there exists some $\pi^* = (d^*_1, d^*_2, ..., d^*_{N-1}) \in \Pi^{MD}$, which satisfies

$$r_t(s_t, d^*_t(s_t)) + \sum_{j \in S} p_t(j|s_t, d^*_t(s_t)) u^*_{t+1}(j) \geq max_{a \in A_{s_t}} \{r_t(s,a) + \sum_{j \in S} p_t(j|s_t,a) u^*_{t+1}(j)\}$$

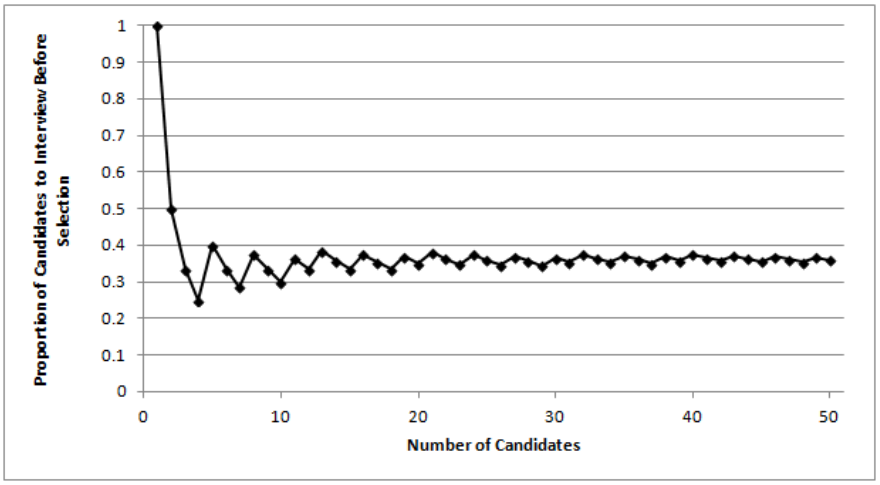
Then π^* is optimal.

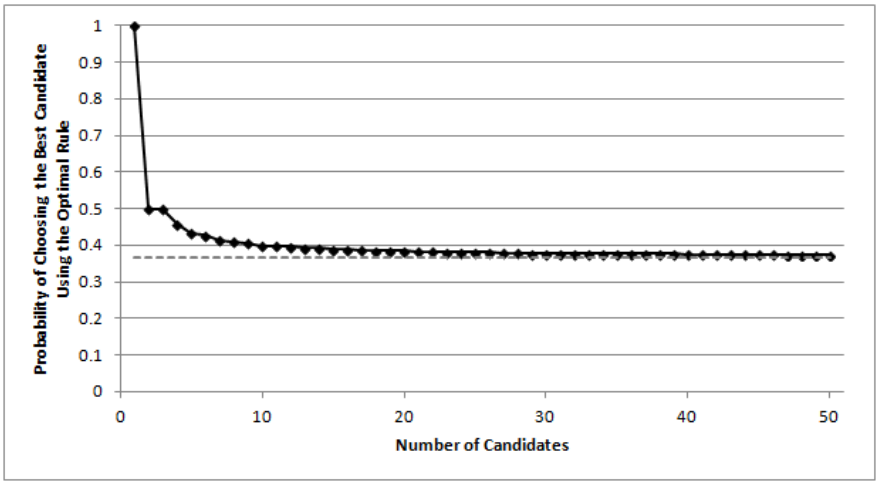
Thus we have proven that when S is finite or countable and A_s are all finite then there exists a deterministic Markovian policy which is optimal.

4 The Secretary Problem

4.1 Figure 4.6.2

The Python code for this problem can be found in the appendix.





4.2 Page 102 Analysis

The first step of the analysis is to note that as $N \to \infty$:

$$\frac{1}{\tau(N)} + \frac{1}{\tau(N)} + \cdots + \frac{1}{N-1} \approx 1 \tag{1}$$

This makes logical sense since we pick the largest τ such that equation 1 is greater than 1 so as N becomes larger the equation goes closer to 1. This can then be simplified to:

$$\sum_{x=\tau(N)}^{N} \frac{1}{x} = \frac{1}{\tau(N)} + \frac{1}{\tau(N)} + \cdots + \frac{1}{N-1}$$

$$
\begin{aligned}
&\approx \int_{\tau(N)}^{N} \frac{1}{x} dx \\
&\approx [log(x)]_{\tau(N)}^{N} \\
&\approx log\,(N) - log\,(\tau(N)) \\
&\approx log\left(\frac{N}{\tau(N)}\right) \\
&\approx 1
\end{aligned}
$$

So now we can take the *log* to the other side (note that this only holds for large N):

$$
\begin{aligned}
log\left(\frac{N}{\tau(N)}\right) &\approx 1 \\
\frac{N}{\tau(N)} &\approx e^1 \\
\frac{\tau(N)}{N} &\approx e^{-1} \\
\tau(N) &\approx Ne^{-1}
\end{aligned}
$$

4.3 Modified Secretary Problem

An interesting modification of the secretary problem would be to consider the need to hire multiple candidates (k) and maximise the sum of their value. Obviously this is a much harder problem than the original and more applicable as it also has uses in online auctions with regards to when to accept a bid. Kleinberg (2005) proposed a simple algorithm for this problem that worked by taking a random sample m from the binomial distribution $Bin(n, 1/2)$ and recursively applying the original algorithm for the secretary problem to select $l = \lfloor k/2 \rfloor$ elements from the first m samples. Then the m candidates are ordered according to their rank with y_1 being the highest. Then resuming sampling, we hire every candidate with a rank higher than y_l. This algorithm was proven to have an expected value of at least $(1 - \frac{5}{\sqrt{k}})v$ where v is the sum of the k highest ranked candidates.

Robert Kleinberg, A multiple-choice secretary algorithm with applications to online auctions, Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, January 23-25, 2005, Vancouver, British Columbia

A Backward Induction Code

```
def rt(s,a, O, h, F):
        if a == 0:
                return F[s] - h*s
        else:
                return F[s+a] - O[a] - h*(s+a)

def induct1(N,M,p_s,f,h,c,k):
        nM = range(M+1)
        x = values(M,p_s,f,h,c,k)
        r = x[1]
        p = x[0]
        # assumed g(u) = 0
        u = {}; policies = {}
        u_n_star = {}; pol_n = {}
        for s in nM:
                x = {}
                for a in nM:
                        x[r[s,a]] = (s,a)
                u_n_star[s] = max(x)
                pol_n[s] = x[max(x)]
        u[N-1] = u_n_star
        policies[N-1] = pol_n
        for n in range(N-1)[:0:-1]:
                u_n = {}
                for i in nM:
                        for a in nM:
                                u_n[i,a] = r[i,a] + sum(u[n+1][b]*p[i+a,b] for b in nM if (i+a)<(M+1))
                u_n_star = {}; pol_n = {}
                for s in nM:
                        x = {}
                        for a in nM:
                                x[u_n[s,a]] = (s,a)
                        u_n_star[s] = max(x)
                        pol_n[s] = x[max(x)]
                u[n] = u_n_star
                policies[n] = pol_n
        return u, policies

def values(M,p_s,f,h,c,k):
```

```
    nM = range(M+1)
    O = {}; p_t = {}; F = {}

    for a in nM[1:]:
        O[a] = k + c*a

    for s in nM:
        for j in nM:
            if j > s:
                p_t[s,j] = 0
            else:
                p_t[s,j] = p_s[s-j]
        if sum(p_t[s,j] for j in nM) < 1:
            p_t[s,0] = 1 - sum(p_t[s,j] for j in nM[1:])

    for u in nM:
        if u == 0:
            F[u] = 0
        else:
            F[u] = sum(p_t[u,u-i]*f*i for i in range(u+1))
    r_t = {}
    for s in nM:
        for a in nM:
            if s+a > M:
                r_t[s,a] = 0
            else:
                r_t[s,a] = rt(s,a, O, h, F)
    return p_t,r_t


a = induct1(15,4,[0.4, 0, 0.4, 0.2, 0],12,2,2,4)
for n in range(14):
    print a[1][n+1]
```

B Secretary Code

This code was used to calculate the actual values for the plots in the Secretary problem, however the plots themselves were made by copying the results across to excel as I'm currently having issues with Python's plotting package.

```
def sec(N):
    r = {}
```

```
        for t in range(N)[1:]:
                a = t
                s = []
                while a < N:
                        s += [1.0/a]
                        a += 1
                r[t] = sum(s)
        best = (5,0)            # arbitrary number > 1
        for t in range(N)[1:]:
                if r[t] >= 1 and r[t] < best[0]:
                        best = (r[t],t)
        return best

res = []
for n in range(51)[1:]:
        res.append(sec(n)[1]/float(n))

def probs(N):
        u = {}
        u[N,1] = 1
        u[N,0] = 0

        for t in range(N)[:0:-1]:
                u[t,0] = 1/float(t+1)*u[t+1,1] + t/float(t+1)*u[t+1,0]
                u[t,1] = max(t/float(N), u[t,0])
        return u[1,0]

res2 = []
for n in range(51)[1:]:
        res2.append(probs(n))
```