Markov Decision Processes

Kjell Raaijmakers (1244095), Jeroen van Riel (1236068)

May 18, 2022

Question 1a

The possible decisions that we can take are *starting a social media campaign* (S), *starting a discount code campaign* (D) or *not starting any campaign* (N). We explicitly consider the possibility of doing nothing, because it is not entirely clear upfront that profit can be made with the currently available campaign proposals. Formally, we define the actions space $\mathcal{A} = \{S, D, N\}$.

In order to satisfy the Markov property, the states need to capture all the information about the past that influence the remaining dynamics of the decision process. We recognize the following three main components of the current problem setting:

1. Budget and accumulated potential profit.
2. Outcome of previous campaign.
3. Duration of the campaign.

As we will see below, the possible values for budget plus accumulated potential profit must be multiples of 100.000 and range between 100.000 and 700.000, so we choose to model the monetary component as an integer $k \in \{1, \ldots, 7\}$. We represent the outcome (unsuccessful/successful) of the last campaign as a binary flag $l \in \{0, 1\}$.

The representation of the time component is little less straightforward. Suppose that we start with a social media campaign, then we have two more months left to plan. We could model these remaining months m as part of the state (k, l, m), such that the possibility of starting a campaign (S or D) depends on m. However, we observe that the notion of m does not coincide with the number of remaining steps n in the finite horizon setting (m decreases by one or two, n always decreases by exactly one). One approach could be to define states with $m = 0$ to be *absorbing* by only allowing action N transitioning to itself, see Section 3.4 in [2].

Instead, we choose to explicitly model the three moments in time at which we could have to make a decision. We need to avoid the possibility of starting a new campaign when we are still in the middle of another, so we encode an active campaign as a binary flag $m \in \{0, 1\}$. The formal definition of the state space thus becomes $(k, l, m) \in \mathcal{I} = \{1, \ldots, 7\} \times \{0, 1\} \times \{0, 1\}$. The initial state is $(2, u, 0)$.

We now discuss two possible ways to model rewards. When we decide to assign rewards directly to the transition from state (k_1, l_1, m_1) to state (k_2, l_2, m_2) after taking actions a, we need to require $r^a(k_1, k_2) = k_2 - k_1$ to reflect the change in accumulated profits. Precisely this property allows us to assign rewards to the final states in terms of the initial budget plus accumulated profits by defining $q(k, l, m) = k$.

Before we define the transition probabilities, let us consider which actions are allowed in which states. Suppose we are not currently executing a campaign, so $m = 0$. If we have at least 200.000 euro ($k \geq 2$), then all actions are available to us, so $\mathcal{A}_{(k,l,0)} = \{S, D, N\}$. Otherwise, we cannot start a social media campaign, so $\mathcal{A}_{(1,l,0)} = \{D, N\}$. Note that k cannot become zero, because we could only lose 100.000 by choosing S, which requires having $k \geq 2$ in the first place. When we are executing a campaign ($m = 1$), then the only allowed action is $\mathcal{A}_{(k,l,1)} = \{N\}$.

The transition probabilities can now be derived rather systematically. First, let the probability $p(l_1, l_2)$ of the next outcome l_2 given the previous outcome l_1 be given by

$$\begin{aligned} p(0,1) &= 0.4, \\ p(0,0) &= 0.6, \\ p(1,1) &= 0.7, \\ p(1,0) &= 0.3. \end{aligned}$$

When choosing S, the next state must have $m = 1$, so

$$p^S((k,l,0),(k,l,1)) = 1.$$

Taking action N next results in the transitions

$$\begin{aligned} p^N((k,l,1),(k+3,1,0)) &= p(l,1), \\ p^N((k,l,1),(k-1,0,0)) &= p(l,0). \end{aligned}$$

When choosing D, the transitions are

$$\begin{aligned} p^D((k,l,0),(k+1,1,0)) &= p(l,1), \\ p^D((k,l,0),(k,0,0)) &= p(l,0). \end{aligned}$$

All other transition probabilities are zero.

Question 1b

We are now ready to apply dynamic programming to find a policy that satisfies Bellman's optimality condition (which is stated as Theorem 1.3.1 in the lecture notes). In Figure 1, we see all the paths that we can take if we take the optimal action in each step. As an example, consider the following decision paths: we begin in the top state, where we have 200.000 euros available, and the expected reward is 436.720 euros. According to our algorithm, the best action we can take is the discount coupons. Now suppose the campaign failed, which means we go to state $2, \mathrm{D}, 3.672$ (i.e., we still have 200.000 euros, and the best possible action we can take is the discount coupons. Our expected reward has decreased to 367.200 euros).

Now suppose this discount campaign was a succes, so we go to state $3, \mathrm{S}, 4.8$, which we reached after 2 months after starting. Now that we are in this state, our best course of action is to do the social media campaign. Since the social media campaign takes 2 months, we always end up in state $3, \mathrm{N}, 4.8$ in the third month.

Finally, suppose the social media campaign was also a succes, which means we finally end up in state 6, which means we end up with 600.000 euros. Note that in the last state, we did neither put the best decision we have to take, nor the expected reward, however, since we are already at month 4, these do not matter.

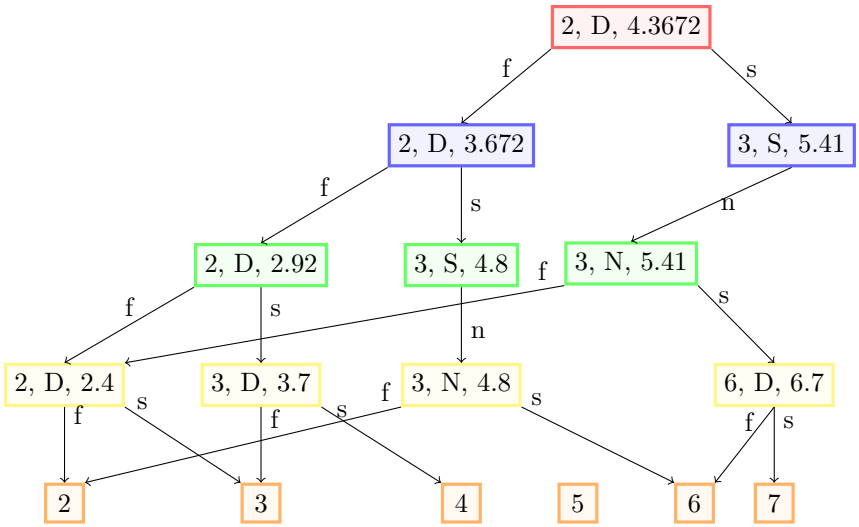

Figure 1: Graphical representation of the optimal policy. Each state $i = (k, l, m)$ is labeled with its k value, the next optimal action and the state value $V_n^*(i)$. Each horizontal level (marked with the same color) corresponds to the same number of remaining time steps n. Furthermore, the letters for each arrow denotes if the campaign was either a succes (denoted by s), a failure (denoted by f), or a "neutral" step (denoted by n). The latter is taken when it is decided to do a social media campaign.

Question 2a

Note that X_t is obtained by *clamping* Y_t between -5 and 5, so we have $X_t = Y_t$ as long as $Y_t \in [-5, 5]$. We assume that the process Y_t takes integer values and that it starts in $Y_0 \in [-5, 5]$, so that the reflecting boundaries make sure that $Y_t \in [-5, 5]$ for all $t \in \mathcal{T}$. This property ensures that X_t itself is a Markov process, which allows us to apply the framework of Markov Decision Processes. Without this property, the transitions of X_t would depend on Y_t in a non-trivial way, e.g. we have $P(X_{t+1} = 5 \,|\, X_t = 5) = 1$ when $Y_t = 6$, but we have $P(X_{t+1} = 5 \,|\, X_t = 5) = 0.5$ when $Y_t = 5$. Under these assumptions, we conclude that it is safe to ignore the distinction between the two processes, so we will refer to X_t in what follows.

Let us now formalize the given problem statement. When we are still a shareholder of the company, the state can be modeled by the value of X_t. Furthermore, let state $\mathcal{L}$ encode the fact that we *left* the company as a shareholder. Later, we will construct the transition probabilities such that $\mathcal{L}$ is an absorbing state (i.e., we will stay in $\mathcal{L}$). The state space is defined as

$$\mathcal{I} = \{-5, -4, \ldots, 4, 5, \mathcal{L}\}.$$

The two actions defined in the problem statement suffice for our analysis, so $\mathcal{A} = \{0, 1\}$. The direct rewards given in the problem statement can be stated as

$$r^0(i) = \begin{cases} i & \text{if } i \neq \mathcal{L} \\ 0 & \text{if } i = \mathcal{L} \end{cases}$$
$$r^1(i) = 0$$

for all $i \in \mathcal{I}$, reflecting the fact that we immediately stop receiving rewards when deciding to leave the company.

Suppose we take action 0. If we were in state $i \notin \{-5, 5, \mathcal{L}\}$, the probability of moving up $(i + 1)$ or down $(i - 1)$ is 0.5. By taking the reflecting boundaries into account, we see that from state $i \in \{-5, 5\}$, we can either stay in that state with probability 0.5, or move (up or down, respectively) with probability 0.5. If we

were in state $\mathcal{L}$, we stay there with probability 1. With a little abuse of notation, we can summarize this as

$$p^0(i,j) = \begin{cases} 0.5 & \text{if } i \in [-4,\, 4],\ j = i \pm 1, \\ 0.5 & \text{if } i = \pm 5,\ j \in \{\pm 4, \pm 5\}, \\ 1 & \text{if } i = \mathcal{L},\ j = \mathcal{L}, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, we always end up in state $\mathcal{L}$ after taking action 1, so $p^1(i,\mathcal{L}) = 1$ for all $i \in \mathcal{I}$ and zero otherwise. For the sake of completeness, we have added the full matrix representations of p^0 and p^1 in Appendix A.

We have now all the necessary components to further specify the functional equation for the optimal value

$$(T_\alpha^* v)(i) = \max_{a \in \mathcal{A}} \left\{ r^a(i) + \alpha \sum_{j \in \mathcal{I}} p^a(i,j) v(j) \right\}.$$

For $a = 1$ in the maximum, the expression reduces to

$$\underbrace{r^1(i)}_{=0} + \alpha \left(\underbrace{p^1(i,\mathcal{L})}_{=1} v(\mathcal{L}) + \sum_{j \in \mathcal{I} \setminus \{\mathcal{L}\}} \underbrace{p^1(i,j)}_{=0} v(j) \right) = \alpha v(\mathcal{L}). \tag{1}$$

Consider $i = \mathcal{L}$, then the expression for $a = 0$ is also

$$\underbrace{r^0(\mathcal{L})}_{=0} + \alpha \left(\underbrace{p^0(\mathcal{L},\mathcal{L})}_{=1} v(\mathcal{L}) + \sum_{j \in \mathcal{I} \setminus \{\mathcal{L}\}} \underbrace{p^0(\mathcal{L},j)}_{=0} v(j) \right) = \alpha v(\mathcal{L}),$$

so that we have

$$(T_\alpha^* v)(\mathcal{L}) = \alpha v(\mathcal{L}).$$

Now from this we see that $\lim_{n\to\infty}(T_{\alpha,n}^* v)(\mathcal{L}) = 0$, which shows that we can also safely take $v(\mathcal{L}) = 0$ from the start. This makes perfect sense, because starting in state $\mathcal{L}$ will not lead to any rewards. From this assumption and (1), we arrive at the form that was given in the problem description

$$(T_\alpha^* v)(i) = \max \left\{ 0,\ r^0(i) + \alpha \sum_{j \in \mathcal{I}} p^0(i,j) v(j) \right\},$$

or with concrete values substituted

$$(T_{0.95}^* v)(i) = \begin{cases} \max\{0, \quad i + 0.95 \cdot (0.5 \cdot v(i-1) + 0.5 \cdot v(i+1))\} & \text{if } i \in [-4,4], \\ \max\{0, \quad 5 + 0.95 \cdot (0.5 \cdot v(4) + 0.5 \cdot v(5))\} & \text{if } i = 5, \\ \max\{0, \ -5 + 0.95 \cdot (0.5 \cdot v(-4) + 0.5 \cdot v(-5))\} & \text{if } i = -5, \\ 0 & \text{if } i = \mathcal{L}. \end{cases}$$

Question 2b

In light of the results presented in Chapter 3 of the lecture notes, we will only consider stationary decision rules $\mathbf{f} = (f, f, f, \dots)$, so we do not have to distinguish between $\mathbf{f}$ and f and we choose to use the latter notation. The policy f_0 of stopping just before we would obtain negative reward is formally given by

$$f_0(i) = \begin{cases} 0 & \text{if } i \in \{0,\ 1,\ \dots,\ 5\}, \\ 1 & \text{if } i \in \{-5, -4, \dots, -1, \mathcal{L}\}, \end{cases}$$

or written as a vector

$$f_0 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where each entry corresponds to $i \in \mathcal{I}$ in the order in which we defined the state space above.

Given a policy f, we want to *evaluate* its performance by determining the *total discounted expected reward* $V_\alpha^f(i)$, for each initial state i. This can be done by solving the functional equation $T_\alpha^f v = v$, where

$$(T_\alpha^f v)(i) := r^f(i) + \alpha \sum_{j \in \mathcal{I}} p^f(i,j) v(j), \quad i \in \mathcal{I}. \tag{2}$$

One possible method is to use iterative substitution, for which convergence is guaranteed by Lemma 3.1.1 in the lecture notes. Note that we may also write (2) in matrix form as

$$T_\alpha^f v = r^f + \alpha p^f v,$$

when regarding r^f and v as a column vectors and defining

$$p^f(i,j) := \begin{cases} p^0(i,j) & \text{if } f(i) = 0, \\ p^1(i,j) & \text{if } f(i) = 1, \end{cases}$$

to be the transition matrix induced by f. This allows us to solve the system of linear equations

$$v = r^f + \alpha p^f v. \tag{3}$$

Using GNU Octave [1] (a free and open-source Matlab alternative), we find that

$$V_\alpha^{f_0} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 9.783065 & 20.595926 & 31.471516 & 41.449371 & 49.474528 & 54.286478 & 0 \end{pmatrix}.$$

Next, we can perform a *policy improvement step* by computing

$$f_1(i) := \arg\max_{a \in \mathcal{A}} \left\{ r^a(i) + \alpha \sum_{j \in \mathcal{I}} p^a(i,j) V_\alpha^{f_0}(j) \right\},$$

from which we find

$$f_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Note that by definition of f_1, we always have $T_\alpha^{f_1} V_\alpha^{f_0} \geq T_\alpha^{f_0} V_\alpha^{f_0} = V_\alpha^{f_0}$. Now observe that we must have strictly improved in $i = -1$ because $f_0(-1) \neq f_1(-1)$, assuming we consistently break ties when performing the argmax operation. Therefore, we have $(T_\alpha^{f_1} V_\alpha^{f_0})(-1) > V_\alpha^{f_0}(-1)$, so Corollary 3.1.1 from the lecture notes gives that $V_\alpha^{f_1}(i) \geq V_\alpha^{f_0}(i)$ for all $i \in \mathcal{I}$ and $V_\alpha^{f_1}(-1) > V_\alpha^{f_0}(-1)$, showing that the original strategy f_0 was not optimal.

Looking at the long term average reward, we note that our transition matrix under f_0 is not positive recurrent; once we reach state $\mathcal{L}$, we stay there forever. This means that we do not satisfy Assumption 2.2.1 from the lecture notes, which means that we cannot calculate the average reward using the steady state distribution. However, we realize that at some point in time, we will take action 1, meaning that in the long term, we will end up in state $\mathcal{L}$. Since the reward in state $\mathcal{L}$ is 0, our long term average reward is 0.

Question 2c

The procedure in the previous section is commonly refered to as *policy iteration*. They key principle is to alternate between policy evaluation and policy improvement. The policy evaluation step involved either solving a linear set of equation or iteratively substituting a functional equation. Instead of waiting for the latter method to reach convergence, we can also choose to stop after a fixed number of substitutions. The extreme case of only performing one substitution is known as *successive approximation* (in the lecture notes) or *value iteration* (Section 4.4 in [2]).

Using GNU Octave [1], we find the optimal policy

$$f^* = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

with corresponding total discounted expected rewards

$$V_\alpha^{f^*} = \begin{pmatrix} 0 & 0 & 0 & 1.00796 & 6.33255 & 14.42900 & 24.04430 & 34.08532 & 43.50376 & 51.18578 & 55.83473 & 0 \end{pmatrix}.$$

We also implemented policy iteration to verify these results, see the Appendix for the code.

Question 2d

Let us define the transition probabilities in terms of ρ. Note that p^1 is not affected by the reformulation. For action 0, we get the new transition probabilities

$$p^0(i,j) = \begin{cases} \rho & \text{if } i \in [-4,4],\ j = i+1, \\ 1-\rho & \text{if } i \in [-4,4],\ j = i-1, \\ \rho & \text{if } i = 5,\ j = 5 \text{ or } i = -5,\ j = -4, \\ 1-\rho & \text{if } i = 5,\ j = 4 \text{ or } i = -5,\ j = -5, \\ 1 & \text{if } i = \mathcal{L},\ j = \mathcal{L}, \\ 0 & \text{otherwise.} \end{cases}$$

Please see the Appendix B for the full updated matrix p^0.

(1) We can evaluate policy f_0 by solving the corresponding linear system (3), which yields the expected discounted rewards in Table 1. As we can see from these numbers, the rewards seem to increase monotonically with ρ. Furthermore, the expected discounted rewards for the *negative states* stays zero.

(2) We compute V_α^* and the optimal policy f^* for a fixed number of values for $\rho \in [0,1]$. Table 2 shows the expected discounted rewards, from which we can see that for ρ close to zero, the expected discounted reward for the *negative states* tends to zero. As ρ gets closer to one, we see that most states (except $\mathcal{L}$, of course) eventually get a positive expected discounted reward. We can do a quick sanity check for $\rho = 1$ by noting for example that $\sum_{n=0}^{\infty} 0.95^n \cdot 5 = 100 = V_\alpha^*(5)$. From Table 3, we can see that f_0 is at least optimal for $0 \leq \rho \leq 0.35$ and that it is not optimal anymore from at least $\rho \geq 0.4$ onwards. Note that we implicitly assume here that both the policy and the expected discounted reward are monotone functions of ρ. We are not sure how to formally prove this, but increasing the step size for ρ in our calculations provides convincing empirical evidence for this.

(3) For the long-term reward, we distinguish two cases. For $\rho < 1$, we will eventually end up in state $\mathcal{L}$ under strategy f_0, meaning that our long-term average reward is zero, just as in question 2b. For $\rho = 1$, the long-term average reward under strategy f_0 will depend on the initial state. If we start in $i_0 \in \{-5, -4, \ldots, -1, \mathcal{L}\}$, we immediately move to state $\mathcal{L}$ under f_0, so the long-term average reward is zero. However, if the initial state is $i_0 \in \{0, 1, \ldots, 5\}$, we will eventually arrive at and stay in state 5, so the long-term average reward is five.

(4) From Lemma 3.2.1 in the lecture notes, we know that value iteration will always eventually converge. However, the number of iterations may vary with ρ. For the fixed set of values for ρ, we can compare the results computed using policy iteration and using value iteration (successive approximation). By using $\epsilon = 0.00001$ in value iteration, the maximum difference between the computed expected discounted rewards is smaller than 0.0002. Furthermore, we found that the number of iteration required in value iteration grows with ρ, ranging between 6 and 258.

ρ \ i	-5	-4	-3	-2	-1	0	1	2	3	4	5	$\mathcal{L}$
0	0	0	0	0	0	0	1	2.95	5.8	9.51	14	0
0.05	0	0	0	0	0	0.0577	1.21	3.41	6.58	10.7	15.4	0
0.1	0	0	0	0	0	0.143	1.5	4	7.57	12.1	16.9	0
0.15	0	0	0	0	0	0.271	1.9	4.8	8.84	13.8	18.9	0
0.2	0	0	0	0	0	0.47	2.47	5.88	10.5	16	21.2	0
0.25	0	0	0	0	0	0.787	3.31	7.38	12.7	18.8	24.1	0
0.3	0	0	0	0	0	1.31	4.58	9.53	15.7	22.4	27.8	0
0.35	0	0	0	0	0	2.17	6.53	12.6	19.8	27	32.5	0
0.4	0	0	0	0	0	3.62	9.53	17	25.2	33	38.4	0
0.45	0	0	0	0	0	6.01	14.1	23.2	32.4	40.5	45.7	0
0.5	0	0	0	0	0	9.78	20.6	31.5	41.4	49.5	54.3	0
0.55	0	0	0	0	0	15.3	29.3	41.6	51.8	59.4	63.7	0
0.6	0	0	0	0	0	22.6	39.6	52.7	62.5	69.3	72.8	0
0.65	0	0	0	0	0	31.1	50.4	63.3	72.1	77.8	80.7	0
0.7	0	0	0	0	0	40.2	60.5	72.2	79.7	84.4	86.7	0
0.75	0	0	0	0	0	49.2	69.1	79.1	85.2	89.1	91	0
0.8	0	0	0	0	0	57.7	76	84.2	89.2	92.3	93.9	0
0.85	0	0	0	0	0	65.6	81.3	87.8	92	94.7	96.1	0
0.9	0	0	0	0	0	73	85.3	90.5	94.1	96.4	97.7	0
0.95	0	0	0	0	0	79.7	88.3	92.6	95.7	97.9	99	0
1	0	0	0	0	0	86	90.5	94.2	97	99	100	0

Table 1: Expected discounted rewards for f_0 for a fixed set values for $\rho \in [0, 1]$, computed by solving the corresponding system of linear equation (3).

ρ \ i	-5	-4	-3	-2	-1	0	1	2	3	4	5	$\mathcal{L}$
0	0	0	0	0	0	0	1	2.95	5.8	9.51	14	0
0.05	0	0	0	0	0	0.0577	1.21	3.41	6.58	10.7	15.4	0
0.1	0	0	0	0	0	0.143	1.5	4	7.57	12.1	16.9	0
0.15	0	0	0	0	0	0.271	1.9	4.8	8.84	13.8	18.9	0
0.2	0	0	0	0	0	0.47	2.47	5.88	10.5	16	21.2	0
0.25	0	0	0	0	0	0.787	3.31	7.38	12.7	18.8	24.1	0
0.3	0	0	0	0	0	1.31	4.58	9.53	15.7	22.4	27.8	0
0.35	0	0	0	0	0	2.17	6.53	12.6	19.8	27	32.5	0
0.4	0	0	0	0	0.552	4.08	9.92	17.3	25.5	33.2	38.6	0
0.45	0	0	0	0	2.38	7.9	15.6	24.4	33.4	41.3	46.4	0
0.5	0	0	0	1.01	6.33	14.4	24	34.1	43.5	51.2	55.8	0
0.55	0	0	0	5.41	14.2	24.6	35.5	46	55.1	62	66	0
0.6	0	0	5.49	14.9	26	37.4	48.3	58	66.1	72	75.3	0
0.65	0	5.59	15.5	27	38.6	49.6	59.5	68.1	74.9	79.9	82.5	0
0.7	8.74	16.9	27.7	38.9	49.7	59.5	68.2	75.5	81.4	85.5	87.6	0
0.75	19.5	27.9	38.3	48.6	58.3	67	74.6	81	86	89.5	91.3	0
0.8	28.5	37	46.8	56.3	65	72.7	79.5	85.1	89.4	92.4	94	0
0.85	36.1	44.5	53.7	62.3	70.2	77.2	83.2	88.2	92	94.7	96.1	0
0.9	42.4	50.7	59.3	67.3	74.4	80.7	86.1	90.6	94.1	96.4	97.7	0
0.95	47.8	56	64	71.3	77.8	83.6	88.5	92.6	95.7	97.9	99	0
1	52.5	60.5	67.9	74.6	80.7	86	90.5	94.2	97	99	100	0

Table 2: Optimal expected total discounted reward $V_\alpha^*(i)$ starting in state i for a fixed set values for $\rho \in [0, 1]$, computed using policy iteration.

ρ \ i	-5	-4	-3	-2	-1	0	1	2	3	4	5	$\mathcal{L}$
0	1	1	1	1	1	0	0	0	0	0	0	1
0.05	1	1	1	1	1	0	0	0	0	0	0	1
0.1	1	1	1	1	1	0	0	0	0	0	0	1
0.15	1	1	1	1	1	0	0	0	0	0	0	1
0.2	1	1	1	1	1	0	0	0	0	0	0	1
0.25	1	1	1	1	1	0	0	0	0	0	0	1
0.3	1	1	1	1	1	0	0	0	0	0	0	1
0.35	1	1	1	1	1	0	0	0	0	0	0	1
0.4	1	1	1	1	0	0	0	0	0	0	0	1
0.45	1	1	1	1	0	0	0	0	0	0	0	1
0.5	1	1	1	0	0	0	0	0	0	0	0	1
0.55	1	1	1	0	0	0	0	0	0	0	0	1
0.6	1	1	0	0	0	0	0	0	0	0	0	1
0.65	1	0	0	0	0	0	0	0	0	0	0	1
0.7	0	0	0	0	0	0	0	0	0	0	0	1
0.75	0	0	0	0	0	0	0	0	0	0	0	1
0.8	0	0	0	0	0	0	0	0	0	0	0	1
0.85	0	0	0	0	0	0	0	0	0	0	0	1
0.9	0	0	0	0	0	0	0	0	0	0	0	1
0.95	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	1

Table 3: Optimal policies $f^*(i)$ for a fixed set values for $\rho \in [0, 1]$, computed using policy iteration.

References

[1] John W. Eaton et al. *GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations*. 2020. URL: `https://www.gnu.org/software/octave/doc/v5.2.0/`.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

A Probability matrices Question 2a

$$
p^0 = \begin{pmatrix}
0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

$$
p^1 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

B Probability matrices in Question 2d

$$
p^0 = \begin{pmatrix}
1-\rho & \rho & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1-\rho & 0 & \rho & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1-\rho & 0 & \rho & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1-\rho & 0 & \rho & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1-\rho & 0 & \rho & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1-\rho & 0 & \rho & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1-\rho & 0 & \rho & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1-\rho & 0 & \rho & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-\rho & 0 & \rho & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-\rho & 0 & \rho & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-\rho & \rho & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

C Code question 1

Dynamic programming (Python)

```
import numpy
I=(0,1,2,3,4,5,6,7,8)
A=("nothing","discount","social")
D=(0,1)
L=("s","f")

def V(i,l,a,d,n):
    if n==4:
        return i
    if (n<4):
        if (d==1):
            if (a=="nothing"):
                if (l=="s"):
                    return (0.7*V(i+3,"s",
                            A[numpy.argmax([V(i+3,"s","nothing",0,n+1),
                            V(i+3,"s","discount",0,n+1),
                            V(i+3,"s","social",0,n+1)])],0,n+1)
                            + 0.3*V(i-1,"f",
                            A[numpy.argmax([V(i-1,"f","nothing",0,n+1),
                            V(i-1,"f","discount",0,n+1),
                            V(i-1,"f","social",0,n+1)])],0,n+1))
                if (l=="f"):
                    return (0.4*V(i+3,"s",
                            A[numpy.argmax([V(i+3,"s","nothing",0,n+1),
                            V(i+3,"s","discount",0,n+1),
                            V(i+3,"s","social",0,n+1)])],0,n+1)
                            + 0.6*V(i-1,"f",
                            A[numpy.argmax([V(i-1,"f","nothing",0,n+1),
                            V(i-1,"f","discount",0,n+1),
                            V(i-1,"f","social",0,n+1)])],0,n+1))
            else:
                return V(0,"s",a,1,n+1)
        if (d==0):
            if (a=="social"):
                if (i>1):
                    return V(i,l,A[numpy.argmax([V(i,l,"nothing",1,n+1),
                    V(i,l,"discount",1,n+1), V(i,l,"social",1,n+1)])],1,n+1)
                else:
                    return V(0,l,"nothing",1,n+1)
            if (a=="discount"):
                if (i>0):
                    if(l=="s"):
                        return (0.7*V(i+1,"s",
                            A[numpy.argmax([V(i+1,"s","nothing",0,n+1),
                            V(i+1,"s","discount",0,n+1),
                            V(i+1,"s","social",0,n+1)])],0,n+1)
                            + 0.3*V(i,"f",
                            A[numpy.argmax([V(i,"f","nothing",0,n+1),
                            V(i,"f","discount",0,n+1),
                            V(i,"f","social",0,n+1)])],0,n+1))
                    if(l=="f"):
                        return (0.4*V(i+1,"s",
                            A[numpy.argmax([V(i+1,"s","nothing",0,n+1),
                            V(i+1,"s","discount",0,n+1),
                            V(i+1,"s","social",0,n+1)])],0,n+1)
```

```
                                + 0.6*V(i,"f",
                                A[numpy.argmax([V(i,"f","nothing",0,n+1),
                                V(i,"f","discount",0,n+1),
                                V(i,"f","social",0,n+1)])],0,n+1))
                    else:
                        return V(i,l,"nothing",0,n+1)
                else:
                    return V(i,l,A[numpy.argmax([V(i,"f","nothing",0,n+1),
                    V(i,"f","discount",0,n+1), V(i,"f","social",0,n+1)])],0,n+1)
```

Dynamic programming and generated policy diagram (Python)

```
import numpy as np

# (action, k2-k1, l1, l2, m1, m2) : prob
probabilities = {
    ('S',  0, 0, 0, 0, 1) : 1,
    ('S',  0, 1, 1, 0, 1) : 1,

    ('N',  3, 0, 1, 1, 0) : 0.4,
    ('N', -1, 0, 0, 1, 0) : 0.6,
    ('N',  3, 1, 1, 1, 0) : 0.7,
    ('N', -1, 1, 0, 1, 0) : 0.3,

    ('D',  1, 0, 1, 0, 0) : 0.4,
    ('D',  0, 0, 0, 0, 0) : 0.6,
    ('D',  1, 1, 1, 0, 0) : 0.7,
    ('D',  0, 1, 0, 0, 0) : 0.3,
}

def p(a, k1, l1, m1, k2, l2, m2):
    key = (a, k2 - k1, l1, l2, m1, m2)
    if key in probabilities:
        return probabilities[key]
    else:
        return 0

def actions(k,m):
    if m == 1:
        # don't allow any actions if we are still in a social campaign
        return ['N']
    if k >= 2:
        return ['S','D','N']
    else:
        return ['D','N']

V = np.zeros((5,9,2,2))
A = np.zeros((5,9,2,2), dtype='str')

K = [0,1,2,3,4,5,6,7,8]
L = [0,1]
M = [0,1]

I = [(k,l,m) for k in K for l in L for m in M]

for k in K:
    V[0,k,:,:] = k

for n in range(1,4+1):
```

```
    for k1,l1,m1 in I:
        maximum = 0
        argmax = ''
        for a in actions(k1,m1):
            S = sum(
                p(a, k1,l1,m1, k2,l2,m2) * V[n-1,k2,l2,m2]
                for k2,l2,m2 in I
            )
            if S > maximum:
                maximum = S
                argmax = a

        V[n,k1,l1,m1] = maximum
        A[n,k1,l1,m1] = argmax
```

We could also automatically produce a policy diagram, similar to the hand-drawn Figure 1. This requires the `pydot` package and GraphViz[1] software to be installed on your system.

```
import pydot

graph = pydot.Dot("policy", graph_type="graph")

# list of lists containing possible states for each timestep
s = []

# initial state, value and next action
i0 = (2,0,0)
v0 = V[(4,*i0)]
a0 = A[(4,*i0)]
s.append([(i0, v0, a0)])
graph.add_node(pydot.Node(str((4, i0[0], a0)),
            label=f"{i0[0]} : {v0:.5g} -> {a0}"))

# only consider states that have a positive probability of actually occuring
for n in range(3,-1,-1):
    states = set()
    # iterate over all states that were reachable in the previous timestep
    for i,_,a in s[-1]:
        for j in I:
            trans = p(a, *i, *j)
            if trans > 0: # only reachable states
                vn = V[(n,*j)] # current value
                an = A[(n,*j)] # optimal next action
                states.add((j, vn, an))

                state_label = (f"{j[0]}" + (f" : {vn:.5g} > {an}"
                                if an != '' else ''))
                graph.add_node(pydot.Node(str((n,j[0],an)),
                                label=state_label))

                edge_label = 'nothing' if j[2] else ('success' if j[1]
                                else 'failure')
                graph.add_edge(pydot.Edge(str((n+1,i[0],a)), str((n,j[0],an)),
                label=edge_label))

    s.append(states)

```

[1] http://www.graphviz.org/documentation/

```
file_name = 'policy.png'
graph.write_png(file_name)
from IPython.display import Image
Image(filename=file_name)
```

D Code question 2

This code was run using Octave [1] version 5.2.0.

```
% Direct rewards
global r0 = [-5 -4 -3 -2 -1 0 1 2 3 4 5 0];
global r1 = [ 0  0  0  0  0 0 0 0 0 0 0 0];
% Discount factor
global alpha = 0.95;

% Transition probabilities after taking action 0: p^0(i,j)
function prob = get_p0 (rho)
  prob = zeros(12);
  prob(1,1) = 1-rho;
  prob(1,2) = rho;
  prob(11,10) = 1-rho;
  prob(11,11) = rho;
  for i = 2:10
    prob(i,i-1) = 1-rho;
    prob(i,i+1) = rho;
  endfor
  prob(12,12) = 1;
end

% Transition probabilities after taking action 1: p^1(i,j)
global p1 = zeros(12);
p1(:,12) = 1;

function rewards = policyToRewards (policy)
  global r0, global r1
  for i = 1:12
    if (policy(i) == 0)
      rewards(i) = r0(i);
    elseif (policy(i) == 1)
      rewards(i) = r1(i);
    end
  end
end

function transitions = policyToTransitions (policy, p0)
  global p1
  transitions = zeros (12,12);
  for i = 1:11
    if policy(i) == 0
      transitions(i,:) = p0(i,:);
    elseif policy(i) == 1
      transitions(i,:) = p1(i,:);
    end
  end
  % Always stay in the final state
  transitions(12,:) = p1(12,:);
end

function policy = valuesToPolicy (values, p0)
  global alpha
```

```
  policy = zeros(1,12);
  for i = 1:11
    % Take the argmax to determine the optimal action.
    % The direct reward and weighted value for choosing action 1 is always
        0.
    [_ ix] = max([(i-6) + alpha .* (p0(i,:) * values), 0]);
    policy(i) = ix - 1;
  end
  policy(12) = 1; % keep taking action 1 once out (does not really matter)
end

%%% Policy Evaluation %%%
function v = policyEvaluation (f, p0)
  global alpha

  % Derive the transitions and rewards given the policy
  Pf = policyToTransitions(f, p0);
  r = policyToRewards(f);

  % Solve the functional equation.
  v = (eye(12) - alpha * Pf) \ r';
end

%%% Policy iteration %%%
function [v, f] = policyIteration (rho)
  p0 = get_p0(rho);

  % The initial policy is to stay invested if X_t >= 0.
  f = [1 1 1 1 1 0 0 0 0 0 0 1];
  f_prev = NaN(1,12);

  while any(f != f_prev)
    % Evaluate the policy
    v = policyEvaluation(f, p0);

    % Perform a step of policy improvement
    f_prev = f; % store to check for optimality
    f = valuesToPolicy(v, p0);
  end
end

%%% Value iteration (successive approximation) %%%
function [v, f, steps] = valueIteration (rho)
  global alpha
  p0 = get_p0(rho);

  v_prev = zeros(12,1);
  v = zeros(12,1);

  M = 1000;
  m = 0;
  eps = 0.00001;
  steps = 0; % count number of steps till convergence

```

```
105   while M - m > eps
106     steps = steps + 1;
107
108     for i = 1:11
109       v(i) = max(0, (i-6) + alpha .* (p0(i,:) * v_prev));
110     end
111
112     M = max(v - v_prev);
113     m = min(v - v_prev);
114
115     v_prev = v;
116   end
117
118   f = valuesToPolicy(v, p0);
119  end
120
121
122 %%% Comparison of different values of rho %%%
123
124 N = 20;
125 policies1 = zeros(N+1, 12);
126 policies2 = zeros(N+1, 12);
127 values1 = zeros(N+1, 12);
128 values2 = zeros(N+1, 12);
129 valueIterationSteps = zeros(N+1,1);
130 valuesf0 = zeros(N+1, 12);
131 rhos = zeros(N+1);
132
133 for n = 1:N+2
134   rho = (1/N) * (n-1);
135   rhos(n) = rho;
136
137   % Optimal rewards, policies
138   [v1, f1] = policyIteration(rho);
139   [v2, f2, steps] = valueIteration(rho);
140   valueIterationSteps(n) = steps;
141   policies1(n,:) = f1;
142   policies2(n,:) = f2;
143   values1(n,:) = v1;
144   values2(n,:) = v2;
145
146   % Rewards for f0
147   valuesf0(n,:) = policyEvaluation(f0, get_p0(rho));
148 endfor
149
150 % print column with rhos, then column with policies, then column with
      values
151 function printPolicies (rhos, policies, values)
152   for i = 1 : length(policies) - 1
153     valuestr = "";
154     for j = 1:12
155       str = num2str(values(i,j), 3);
156       valuestr = [valuestr sprintf("%6s ", str)];
157     end
```

```
158     fprintf( '%4d %s  %s \n', rhos(i), mat2str(policies(i,:)), valuestr );
159   end
160 end
161 
162 fprintf("\nPolicy Iteration:\n")
163 printPolicies(rhos, policies1, values1)
164 fprintf("\nValue Iteration:\n")
165 printPolicies(rhos, policies2, values2)
166 fprintf("\nRequired number of iteration:\n %s \n", mat2str(
      valueIterationSteps))
167 
168 
169 f0 = [1 1 1 1 1 0 0 0 0 0 0 1];
170 fprintf("\nPolicy Evaluation for f0:\n")
171 for i = 1 : N + 1
172     valuestr = "";
173     for j = 1:12
174       str = num2str(valuesf0(i,j), 3);
175       valuestr = [valuestr sprintf("%6s ", str)];
176     end
177     fprintf( '%4d %s \n', rhos(i), valuestr );
178   end
```