CONCORDIA UNIVERSITY

Department of Mathematics & Statistics

STAT 497 (MACF 491 H, MAST 679 H, MAST 881 H)

Reinforcement Learning

Assignment 2

Frédéric Godin, Ph.D., FSA, ACIA

January 2019

1 Instructions

- The assignment is done in teams of **one to four** people.
- You are allowed to discuss the content of the assignment with people from other teams. However, solutions must be designed separately. Copies of solution files (pdf report and R code) from different teams must be redacted separately. Plagiarism is formally prohibited. Please refer to Concordia's policy related to plagiarism if you are not sure you understand clearly what plagiarism is.
- The limit date to hand in your assignment is **Friday March 8, 2019 at 11h59 pm**.
- A zip file **".zip"** must be handed in by email. This file should contain (exactly)
    - A **pdf** report containing the answer to all of the questions. For some sections of the assignment, the report might simply need to refer to the R code. This file can be prepared among others by Microsoft Word or LaTeX. This pdf file does not need to contain the R code developed to obtain the answers.
    - A single file **"main.R"** which contains the integral R code needed to run all experiments which lead to your answers. All numerical experiments must be done with the R software.
- Ensure your document is clean and clearly written. Points can be deducted if the clarity is deemed inadequate. This includes putting comments within your R code to ensure readability.
- The zip file must be sent by email to the professor before the limit date. The

professor's email is frederic.godin@concordia.ca

- **Justify all your answers and present the integrality of the rationale leading to your answers.**
- **Ensure the name of all teammates and their Concordia ID number are written on the first page of your report**.

2 Assignment questions

The objective of this assignment is to solve various Markov decision problems with methods of dynamic programming and reinforcement learning using a tabular representation of state values.

Question 1 (30 points)

You want to help an Italian plummer with a stunning mustache called Mario to fight a mighty evil dragon called Bowser and save the Princess Peach. You want to design a fighting strategy for Mario which maximizes his chances of winning.

Rules of the battle are the following:

- Mario initially has 30 health points (HP) whereas Bowser starts the fight with 100 HP. Mario starts the fight with 3 mushrooms in his possession.
- The first contender to fall at or under 0 HP loses the fight.
- The fight consists in a sequence of rounds which continue until one of the opponents loses. On each round, Mario start by performing an action, followed from an action by Bowser.
- Everytime Mario performs an action he has two choices. He can either attack Bowser or eat a mushroom. When Mario attacks, Bowser loses 5 HP. When Mario eats a Mushroom, he loses a mushroom and recovers half of his missing HP (an amount which is rounded to the highest integer if fractional) i.e. he recovers $\lceil 0.5(30-x) \rceil$ where x is the current HP of Mario. If Mario has no mushroom left, the only possible action

for him is to attack Bowser.

- When Bowser performs an action, he either slashes Mario with his claws with probability 80% or blows fire at Mario with probability 20%. When Bowser slashes, Mario loses a number of HP distributed according to a Binomial($n = 5, p = 0.4$). When Bowser blows fire, Mario loses a number of HP distributed according to a Binomial($n = 10, p = 0.7$).

This framework can be represented as a Markov decision process where states are defined by $s = (x, y, m)$ where x is the remaining HP points of Mario, y is the remaining HP points of Bowser and m is the number of mushrooms Mario still possesses. The set of states is therefore defined by the Cartesian product

$$\mathcal{S} = \{0, 1, \ldots, 30\} \times \{0, 1, \ldots, 100\} \times \{0, 1, 2, 3\}.$$

Possible actions in a state s are either 'eat a mushroom' (1) if some are left, or 'attack Bowser' (0):

$$\mathcal{A}\left((x, y, m)\right) = \begin{cases} \{0, 1\} \text{ if } m > 0, \\ \{0\} \text{ if } m = 0. \end{cases}$$

.

This Markov Decision Problem can be solved explicitly with a single run of the Bellman Equation (see Exercise 4.1). Indeed, consider the bijective mapping $f : \mathcal{S} \to \{1, \ldots, 31 \times 101 \times 4\}$ defined through

$$f(x, y, m) = (x + 1) + 31(y + 1) + (31 \times 101)(m + 1)$$

which maps any state (x, y, z) into an integer n. f will be referred to as the **state mapping function**. Consider f^{-1} its inverse mapping.

The states can be partitioned as $\mathcal{S} = \cup_{n=1}^{31\times101\times4} \mathcal{S}_n$ where $\mathcal{S}_n = \{f^{-1}(n)\}$ contains a single state. Since the number of mushrooms never increases, and for a fixed number of consumed mushrooms the HP of both Mario and Bowser never increases, being in a state $f^{-1}(N)$ guarantees you will transition into one of the states in $\cup_{n=1}^{N-1} \mathcal{S}_n = \{f^{-1}(n) : n = 1, \ldots, N-1\}$.[1]

Part (a), 10 points:

Consider the following policy π: whenever Mario's HP becomes smaller or equal to 5, Mario consumes a mushroom if any is left.

Create an array `PolicyStateValueFunction` of dimension 31 by 101 by 4 which will store the calculated policy state value function $v_\pi(s)$ for every state $s \in \mathcal{S}$.[2] The element `PolicyStateValueFunction[i,j,k]` of the array will contain $v_\pi\left((i-1, j-1, k-1)\right)$ the value function when $x = i-1$, $y = j-1$ and $m = k-1$. Apply recursively the Bellman equation associated with $v_\pi(s)$ to fill that matrix i.e. use

$$v_\pi(s) = \sum_{s'} \sum_{r} p\left(s', r | s, \pi(s)\right)\left(r + v_\pi(s')\right) \tag{1}$$

where

- $\pi(s)$ is the action taken in state s by the policy π,
- all states where either $x = 0$ or $y = 0$ are terminal states, i.e. $v_\pi\left((x, y, m)\right) = 0$ for such states,

[1] The notation convention here is slightly different from Exercise 4.1 since here being in state n guarantees you will be transition in a state smaller than a, whereas in Exercise 4.1 you were guaranteed to end up in a state greater than n. However, both formulations are clearly equivalent by properly renaming the states.

[2] We could instead have stored the value function into a vector by using the state mapping function. However we won't use this approach here.

- a reward 1 is obtained when and only when transitioning to a state of the form $(x, 0, m)$,
- note that when reaching a state $(0, 0, m)$, the reward is still obtained since Mario attacks first in the round.

For this to work, the set of states must be swept in a specific order when applying (1) i.e.

- Do all states where $m = 0$, then all states, where $m = 1$, etc. i.e. in an increasing order of m.
- For a fixed value of m, you should calculate v_π for states in increasing order with respect to either y or x.

Part (b), 15 points:

Create two arrays `StateValueFunction` and `OptimalActionMat` of dimension 31 by 101 by 4. `StateValueFunction` will store the optimal state-value function v_*, whereas `OptimalActionMat` will store the optimal action in each state. The element `StateValueFunction[i,j,k]` of the array will contain $v_*((i-1, j-1, k-1))$ the value function when $x = i - 1$, $y = j - 1$ and $m = k-1$. The element `OptimalActionMat[i,j,k]` of the array will contain $\pi_*((i-1, j-1, k-1))$ i.e. the optimal action when $x = i - 1$, $y = j - 1$ and $m = k - 1$.

Apply recursively the Bellman equation associated with $v_*(s)$ to fill these matrices i.e. use

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r|s, a)(r + v_*(s')), \tag{2}$$

$$\pi_*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} \sum_{r} p(s', r|s, a)(r + v_*(s')). \tag{3}$$

States should be swept in the same order than in part (a) to ensure the validity of the answer (i.e. sweeping the states in an incorrect order will result in an erroneous answer).

Part (c), 5 points:

Answer the following questions:

- (1) What is the probability at the beginning of the fight to beat Bowser by following the policy to consume a mushroom whenever Mario has 5 HP or less and a mushroom is available?
- (2) What is the probability at the beginning of the fight to beat Bowser if the optimal policy is followed?
- (3) What is the optimal action to perform if a single mushroom is left, Mario has 12 HP left and Bowser has 22 HP left (assuming the optimal policy will be followed subsequently)?
- (4) When a single mushroom is left, Mario has 5 HP left and Bowser has 11 HP left, calculate the value associated with the 'attack Bowser' action provided the optimal policy is followed at all time steps after this action (use the content of the matrix `StateValueFunction` for this).

Question 2 (40 points)

This question is about the identification of the optimal policy in the Recycling Robot problem outlined in Example 3.3 of the Sutton & Barto textbook (p.52). First, read this Example 3.3 in the textbook.

Consider the following values for parameters of the problem:

$$\alpha = 0.6, \quad \beta = 0.8, \quad r_{search} = 2, \quad r_{wait} = 1.$$

Moreover, assume a discount factor of $\gamma = 0.9$.

Furthermore, use the following names (labels) for the states and actions:

$$\text{State } 1 = \text{high}, \quad \text{State } 2 = \text{low},$$

$$\text{Action } 1 = \text{search}, \quad \text{Action } 2 = \text{wait}, \quad \text{Action } 3 = \text{recharge}.$$

Throughout this question we **will only consider deterministic policies** which assign a single action in each state.

Part (a), 2 points:

In `R`, enter the transition probabilities (provided in the table of Example 3.3 from the textbook) in a matrix `SASRp`. Columns in the matrix should be

- Column 1: Current state s,
- Column 2: Action a,
- Column 3: Subsequent state s',
- Column 4: Reward r,
- Column 5: Transition probability $p(s', r|s, a)$.

You can ignore the rows associated with a null transition probability (they don't need to be included in the matrix).

Part (b), 10 points:

Write a function `CalculatePolicyValueFunction` which calculates the exact value function v_π for a given policy π.

Inputs of the function must be

- `Policy`: A vector $[\pi(1)\ \pi(2)]$ containing a deterministic policy i.e. the action in both of the states,
- `SASRp`: The transition probability table (the one built in "part a"),
- `DF`: The discount factor γ.

The output of the function should be

- `PolicyVF`: A vector $\boldsymbol{v}_\pi = [v_\pi(1)\ v_\pi(2)]^\top$ containing the state-value function in each state for the policy π.

Note that the vector $\boldsymbol{v}_\pi$ is the solution to the system of equations

$$\boldsymbol{v}_\pi = M\boldsymbol{v}_\pi + \boldsymbol{b}$$

where

- $\boldsymbol{b}$ is a vector of length 2 whose element s is $\boldsymbol{b}_s = \sum_{s',r} rp(s', r|s, \pi(s))$,
- M is a 2×2 matrix whose element on row s and column s' is $M_{s,s'} = \gamma \sum_r p(s', r|s, \pi(s))$.

This system of equations can therefore be solved as

$$\boldsymbol{v}_\pi = (I - M)^{-1}\boldsymbol{b}$$

where I is the identity matrix.

There are a total of 6 possible policies $\pi^{(1)}, \ldots, \pi^{(6)}$ in this problem, i.e.

$$\begin{aligned}
\pi^{(1)}(1) &= 1, \quad \pi^{(1)}(2) = 1, \\
\pi^{(2)}(1) &= 1, \quad \pi^{(2)}(2) = 2, \\
\pi^{(3)}(1) &= 1, \quad \pi^{(3)}(2) = 3, \\
\pi^{(4)}(1) &= 2, \quad \pi^{(4)}(2) = 1, \\
\pi^{(5)}(1) &= 2, \quad \pi^{(5)}(2) = 2, \\
\pi^{(6)}(1) &= 2, \quad \pi^{(6)}(2) = 3.
\end{aligned}$$

Use the function `CalculatePolicyValueFunction` to calculate the state-value function associated with all 6 policies. Store these state-value functions

in a table `VFAllPolicies` with 6 rows (one row per policy) and two columns (column s corresponds to state s). In other words, the j^{th} row of this table should contain $[v_{\pi^{(j)}}(1)\ v_{\pi^{(j)}}(2)]$.

- **Question**: Write the content of table `CalculatePolicyValueFunction` in your report.
- **Question**: What is the optimal policy?

Part (c), 8 points:

Write a function `PolicyEvaluation` which applies iterations of the policy evaluation procedure for a given policy π inputted to the function, using a given initial estimate of the state-value function.
Inputs of the function must be

- `StartVF`: A vector $[V_0(1)\ V_0(2)]$ containing the initial estimate of the state-value function,
- `Policy`: A vector $[\pi(1)\ \pi(2)]$ containing a deterministic policy i.e. the action in both of the states,
- `SASRp`: The transition probability table (the one built in "part a"),
- `DF`: The discount factor γ,
- `niter`: the number of iterations (sweeps) of policy evaluation applied.

The output of the function should be

- `VFEstimateNext`: A vector $V_{niter} = [V_{niter}(1)\ V_{niter}(2)]^\top$ containing the final state-value function estimate for the policy π after applying the `niter` iterations of the policy evaluation procedure.

Question: What estimate do you obtain if you apply a single policy evaluation iteration for the policy $[\pi(1) = 1,\ \pi(2) = 1]$ using an initial state-value function estimate of $[V_0(1) = 3,\ V_0(2) = 2]$? Write the answer in your report.

Part (d), 7 points:

Write a function `PolicyImprovement` which applies the policy improvement procedure (i.e. identifies the greedy action in each state) given an estimate of the state-value function.

Inputs of the function must be

- `VFEstim`: A vector $[V(1)\ V(2)]$ containing the estimate of the state-value function,
- `SASRp`: The transition probability table (the one built in "part a"),
- `DF`: The discount factor γ.

The output of the function should be

- `PolicyEstimate`: A vector $[\pi(1)\ \pi(2)]$ containing the policy provided by the policy improvement procedure.

Question: What policy do you obtain if you apply the policy improvement procedure using the state-value function estimate $[V_0(1) = 3,\ V_0(2) = 2]$? Write the answer in your report.

Part (e), 3 points:

Consider initial estimates of the value function and the optimal policy:

$$\hat{v}_*(1) = 0, \quad \hat{v}_*(2) = 0, \quad \hat{\pi}_*(1) = 2, \quad \hat{\pi}_*(2) = 2.$$

Use the functions `PolicyEvaluation` and `PolicyImprovement` from part (c) and (d) to apply 20 cycles of the Generalized Policy Iteration (GPI)

procedure. A single cycle consists in applying successively 10 iterations of the policy evaluation procedure followed by the policy improvement procedure to the current estimates of the optimal value function and optimal policy.

Question: What is the final estimate of the optimal policy and optimal value function you obtain after the 20 GPI cycles? Write the answer in your report.

Part (f), 10 points:

Write a function `ValueIteration` which applies iterations of the value iteration procedure using a given initial estimate of the state-value function V_0.

Inputs of the function must be

- `StartVF`: A vector $[V_0(1)\ V_0(2)]$ containing the initial estimate of the state-value function,
- `SASRp`: The transition probability table (the one built in "part a"),
- `DF`: The discount factor γ,
- `niter`: the number of iterations (sweeps) of value iteration applied.

The output of the function should be a list `ListVFPolicy` of length 2 containing the following elements

- Element 1: A vector $V_{niter} = [V_{niter}(1)\ V_{niter}(2)]$ containing the final state-value function estimate after applying the `niter` iterations of the policy evaluation procedure.
- Element 2: A vector $[\pi_{niter}(1)\ \pi_{niter}(2)]$ containing the estimated optimal policy provided by the policy improvement procedure after the `niter` iterations.

Question: Consider an initial estimate of the state-value function of $[V_0(1) = 0,\ V_0(2) = 0]$.

What estimate of the optimal value function and optimal policy do you obtain after applying respectively $2, 5, 25$ or 500 iterations of the value iteration procedure? Write the answer in your report.

Question 3 (30 points)

This question relates to the use of Monte-Carlo estimators of the value function in the context of the Blackjack cards game.

First, read Example 5.1 (Blackjack, p. 93) of the Sutton & Barto textbook (second edition). We assume the size of the card deck is infinite, and therefore the cards drawn at every stage are independent from previously drawn cards. Let's describe assumptions we make about the use of aces. Assume aces always count for eleven, except when the total of 21 is exceeded, at which point one of the aces is set to be worth 1. If a usable ace is available and a new ace is drawn, then the new ace automatically counts for 1 and the previous ace still counts as a usable ace.

The state faced in this problem at any stage where the player has to make a decision can be defined as a triplet $s = (x, y, z)$. x is the total of the dealer after his first card is visible. x ranges from 2 to 11 as an ace initially counts for 11. Note that the dealer can also have is ace value reduced to 1 eventually if he exceeds 21. y is the total of the player, which ranges from 2 to 21. z is the number of usable aces by the player, which is either 0 or 1 since all new aces drawn when a usable ace is available are automatically worth one.

Part (a), 20 points:

Create an `R` function `SimulateBlackJackEpisodeStick20` which simulates an episode of the Blackjack game and outputs its outcomes. The single input of the function should be

- `theseed`: the random seed used to generate the cards drawn in the episode.

The output should be a list of size 6 containing the following elements:

- Element 1: the episode reward,
- Element 2: the sequence of cards drawn by the player,
- Element 3: the sequence of totals of the player during the episode after each card he draws. Note: an ace counts as 11, except if it leads to exceed 21 in which case it is worth 1. Moreover, an ace that is drawn when a usable ace is available is automatically worth 1,
- Element 4: the sequence of 0-1 indicators indicating if the player has a usable ace every time the player draws a card,
- Element 5: the sequence of cards drawn by the dealer,
- Element 6: the sequence of totals of the dealer during the episode after each card he draws.

Advice: I suggest you to simulate a few games and check the output to ensure they make sense and are consistent with the rules. This is helpful for debugging.

Part (b), 5 points:

Use the function `SimulateBlackJackEpisodeStick20` from part (a) to generate $500,000$ episodes of the Blackjack game. Every time an episode is generated, use the outputs of the function to update the state-value function

estimate for all states that were visited during the episode. Use an equally weighted approach to construct the estimate i.e. use the rule

$$\begin{aligned} V_i(s) &= V_{i-1}(s) + \frac{1}{N_s^{(i-1)} + 1} \left[R^{(i)} - V_{i-1}(s) \right] \mathbb{1}_i(s), \\ N_s^{(i)} &= N_s^{(i-1)} + \mathbb{1}_i(s), \end{aligned}$$

where $\mathbb{1}_i(s)$ is the dummy variable which is worth one if state s was visited during episode i or zero otherwise, $N_s^{(i)}$ is the number of times state s was visited in the first i episodes, $V_i(s)$ is the estimate state-value function evaluated in state s after i episodes and $R^{(i)}$ is the reward in episode i.
The total number of times each state was visited should be stored in an array `Nsamplebystate` of size 10 by 10 by 2. The final value function estimate should be stored in an array `ValueFunctionEstim` of the same dimension. The entry `ValueFunctionEstim[i,j,k]` should contain the value function estimate when the dealer's total is $i + 1$, the player's total is $j + 11$ and the number of usable ace of the player is $k - 1$ with $i = 1, \ldots, 10$, $j = 1, \ldots, 10$, and $k = 1, 2$. Entries from `Nsamplebystate` are analogous.

Questions: what are

- the estimated value function when the dealer's total is 7, the player's total is 14 and the player has no usable ace,
- the estimated value function when the dealer's total is 4, the player's total is 16 and the player has one usable ace,
- the estimated value function when the dealer's total is 4, the player's total is 20 and the player has no usable ace,
- the number of times among the 500,000 episodes the following state was visited: the dealer's total is 4, the player's total is 18 and the player has no usable ace.

Part (c), 5 points:

Use the content of the array `ValueFunctionEstim` to represent the estimated value function exactly the same way as in the two right plots of Figure 5.1 from the textbook, i.e. make one plot for when a usable ace is available, and another one when such an ace is not available.

Use the function `surf3D` of `R`, which will required loading `R` the `plot3D` package.[3]

Note: the states where both 'a usable ace is available' and 'the player has 12' is unattainable[4], and thus you should exclude them from the plot.

An example of code which could be used to produce the plot is the following:

```
gridPlayerTotal = 12:21
gridDealerTotal = 2:11
M <- mesh(gridPlayerTotal, gridDealerTotal)
surf3D(x = M$x, y = M$y, z = t(ValueFunctionEstim[,,1]) )
```

Note that the following parameters could also be used in `surf3D` to address the readability of the plot:

- `xlab`, `ylab`, `zlab`: puts labels on your axis,
- `theta`, `phi`: rotation angle to view your plot,
- `bty`: set `bty="b2"` or `bty="g"` to display axis grids and labeling,
- `ticktype`: allows for explicit axis grading,
- `border`: draws lines on edges of the tiles of the 3D surface,

[3]If such a package is not installed on your computer, you can install it through the command `install.packages("surf3D")`.

[4]The only way a total of 12 could be reached with at least a usable ace is with two aces. But in that case the second ace nullifies the first one and then is used to avoid a total of 22, so no usable ace is left.

- `colkey`: set `colkey=FALSE` to remove the color key legend and have more space for the figures,
- `zlim`: set limits for the z axis,
- `cex.axis`: axis numbers magnification,
- `cex.lab`: axis label magnification.