CONCORDIA UNIVERSITY

Department of Mathematics & Statistics

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

Reinforcement Learning

Assignment 3

Frédéric Godin, Ph.D., FSA, ACIA

February 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 **Sunday April 14, 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

Question 1 (50 points)

This question considers the can recycling robot problem previously encountered in Question 2 of Assignment II. Exactly the same dynamics (states, actions, rewards, transition probabilities) are considered here. Indeed, state 1 and 2 correspond respectively to '*high energy*' and '*low energy*', and actions 1, 2 and 3 are respectively '*search*', '*wait*' and '*recharge*'. Go back to Assignment II and to Example 3.3 of the Sutton & Barto textbook (p.52) to refresh your memory about the problem.
The same parameters are considered as before:

$$\alpha = 0.6, \quad \beta = 0.8, \quad r_{search} = 2, \quad r_{wait} = 1, \quad \gamma = 0.9.$$

The only additional possibility we add here is that the user is allowed to 'recharge' (action 3) in state 1, which will lead to a transition to state 1 (energy will remain high) and provide a null reward with probability 1.
This time, temporal difference methods sur as SARSA and Watkin's Q-learning will be applied to estimate action-value functions in this problem. We will therefore also consider stochastic policies instead of only deterministic ones.

Part (a), 10 points:

In `R`, re-enter the same matrix `SASRp` characterizing the transition probabilities that was defined in Assignment II. 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 $p(s', r|s, a)$.

The only change is that now the matrix `SASRp` should contain an additional row which allows for recharging (action 3) in the high energy state (state 1).

Write a function `CalculatePolicyStateValueFunction` which calculates the exact value function v_π for a given stochastic policy π.
Inputs of the function must be

- `Policy`: A matrix

$$\begin{bmatrix} \pi(1|1) & \pi(2|1) & \pi(3|1) \\ \pi(1|2) & \pi(2|2) & \pi(3|2) \end{bmatrix}$$

  containing a stochastic policy i.e. each row correspond the probability of taking each respective action in a given state,
- `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 $\boldsymbol{b}_s = \sum_{s',r,a} rp(s', r|s, a)\pi(a|s)$,
- M is a 2×2 matrix whose element on row s and column s' is $M_{s,s'} = \gamma \sum_{r,a} p(s', r|s, a)\pi(a|s)$.

This system of equations can therefore be solved as

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

where I is the identity matrix.

Finally, write a function `CalculatePolicyActionValueFunction` which calculates the exact value function q_π for a given stochastic policy π. This function should call the function `CalculatePolicyStateValueFunction`. Inputs of the function must be

- `Policy`: A matrix
$$\begin{bmatrix} \pi(1|1) & \pi(2|1) & \pi(3|1) \\ \pi(1|2) & \pi(2|2) & \pi(3|2) \end{bmatrix}$$
containing a stochastic policy i.e. each row correspond the probability of taking each respective action in a given state,
- `SASRp`: The transition probability table (the one built in "part a"),
- `DF`: The discount factor γ.

The output of the function should be

- `PolicyAVF`: A matrix having giving the action value function q_π associated with the policy π provided as input to the function. The matrix should have the following format:
$$\begin{bmatrix} q_\pi(1,1) & q_\pi(1,2) & q_\pi(1,3) \\ q_\pi(2,1) & q_\pi(2,2) & q_\pi(2,3) \end{bmatrix}$$

where $q_\pi(s, a)$ is the value of action a in state s.

Question: Use the function `CalculatePolicyActionValueFunction` to calculate the action value function q_π of the following policy:

$$\begin{bmatrix} \pi(1|1) = 0.7 & \pi(2|1) = 0.2 & \pi(3|1) = 0.1 \\ \pi(1|2) = 0.1 & \pi(2|2) = 0.7 & \pi(3|2) = 0.2 \end{bmatrix}. \tag{1}$$

Store this function q_π in a matrix `PolicyAVF` and provide it in your report.

Part (b), 10 points:

Write a function `SimulateEpsiodeCanRecycler` which simulates the beginning of an episode of the can recycling robot MDP (only the beginning can be simulated as the length of episodes in this problem is infinite).
Inputs of the function must be

- `Policy`: A matrix

$$\begin{bmatrix} \pi(1|1) & \pi(2|1) & \pi(3|1) \\ \pi(1|2) & \pi(2|2) & \pi(3|2) \end{bmatrix}$$

  containing a stochastic policy i.e. each row correspond the probability of taking each respective action in a given state,
- `SASRp`: The transition probability table (the one built in "part a"),
- `Eplength`: the number of time steps simulated (i.e. number of actions taken in the simulated episode),
- `initstate`: the initial state in the simulated episode,
- `theseed`: the seed used to generate random numbers in the episode.

The output of the function should be

- `EpisodeOutcomes`: A matrix of three rows by `Eplength` columns. The first, second and third rows respectively contain the sequence of states, actions and rewards faced in the episode:

$$\text{EpisodeOutcomes} = \begin{bmatrix} S_1 & S_2 & \cdots \\ A_1 & A_2 & \cdots \\ R_2 & R_3 & \cdots \end{bmatrix}$$

Part (c), 10 points:

Create an `R` function `ApplySARSA` which applies the n-step SARSA algorithm to estimate the action-value function q_π using a simulated episode. Recall that this approach entails using

$$\begin{aligned} Q_1 &= \ldots = Q_n, \\ \text{For } t \geq 1, \quad Q_{t+n}(S_t, A_t) &= Q_{t+n-1}(S_t, A_t) + \tilde{\alpha}\left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)\right], \\ G_{t:t+n} &= \sum_{j=1}^{n} \gamma^{j-1} R_{t+j} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \end{aligned}$$

The inputs of the function should be

- `stepsize`: the update step size $\tilde{\alpha}$,
- `nparam`: the parameter n in n-step SARSA,
- `initQestim`: the initial estimate of the q-function (a matrix of 2 rows by three columns, the row corresponding to the state and the column corresponding to the action),
- `EpisodeOutcomes`: the episode outcomes (the sequence $S_1, A_1, R_2, S_2, A_2, \ldots$) organized in a matrix with three rows (the first, second and third rows are respectively the sequence of states, actions and rewards).
- `DF`: the discount factor γ.

The output should be

- `Qestimmat`: the estimate of the function q after each period. This matrix should have the same number of columns than `EpisodeOutcomes`, where column t contains the estimate Q_t. Each row corresponds to a state-action pair according to:

$$\text{Row 1: } Q_t(1,1), \quad \text{Row 2: } Q_t(1,2), \quad \text{Row 3: } Q_t(1,3),$$
$$\text{Row 4: } Q_t(2,1), \quad \text{Row 5: } Q_t(2,2), \quad \text{Row 6: } Q_t(2,3).$$

Part (d), 10 points:

Simulate an episode of length 10,000 with the function `SimulateEpsiodeCanRecycler` from part (b) starting in state 1 (high energy) and using the policy π outlined in Equation (1) of part (a).

Consider three values for the parameter n: 1, 5 and 20.

For each of these value, apply your function `ApplySARSA` on the simulated episode with a step size of $\alpha = 0.01$ and an initial value estimate of 0 for all state-action pairs.

For each n, compute for each time step t the root mean square error (RMSE) of your approximation in comparison to the true q_π which you obtained in part (a). The RMSE is defined as

$$\text{RMSE}_t = \sqrt{\sum_{s,a} \left(Q_t(s,a) - q_\pi(s,a)\right)^2}$$

The series $\text{RMSE}_t, t = 1, \ldots, 10{,}000$ should be stored in a matrix `RMSEmat` of three row (each row corresponds to a value of n) and 10,000 columns.

Plot each of the three series (versus $t = 1, \ldots, 10{,}000$). $n = 1$ $n = 5$ and $n = 20$ should be respectively a blue, a red and a green curve. Add a legend to identify your curves. Put the plot in your report.

Question: Among the three values tested for n, which seems to be the best for the current problem?

Part (e), 10 points:

Create an `R` function `ApplyQLearning` which applies Watkin's Q-learning methodology to estimate the optimal action-value function q_* using a simulated episode. Recall that this approach entails using

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \tilde{\alpha}\left[R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t)\right].$$

The inputs of the function should be

- `stepsize`: the update step size $\tilde{\alpha}$,
- `initQestim`: the initial estimate of the q-function (a matrix of 2 rows by three columns, the row corresponding to the state and the column corresponding to the action),
- `EpisodeOutcomes`: the episode outcomes (the sequence $S_1, A_1, R_2, S_2, A_2, \ldots$) organized in a matrix with three rows (the first, second and third rows are respectively the sequence of states, actions and rewards).
- `DF`: the discount factor γ.

The output should be

- `Qestim`: the final estimate of the function q after going through the episode (a matrix of 2 rows by three columns, the row corresponding to the state and the column corresponding to the action).

Simulate an episode of length 25,000 with the function `SimulateEpsiodeCanRecycler` from part (b) starting in state 1 (high energy) and using the policy π outlined in Equation (1) of part (a). Apply your function `ApplyQLearning` on

this episode with a step size of $\alpha = 0.01$ and an initial value estimate of 0 for all state-action pairs. Compare your estimate $\hat{q}_*$ of the optimal function to the true function q_* (the latter can be obtained with the function `CalculatePolicyActionValueFunction` from part (a) by recalling from Assignment II that the optimal policy is action 1 in state 1 and action 3 in state 2). Put the results for each state-action pair in a table within your report.

Question 2 (50 points)

This question is about the use of the REINFORCE policy gradient method. The impact of using a baseline within the method is investigated.

First, read Example 13.1 on p.323 of the textbook about the short corridor with switched actions. The policy considered if of the form of equations (13.2)-(13.3) from the textbook i.e. a softmax applied on action preferences represented by a features approximation:

$$\begin{aligned} \pi(a|s,\boldsymbol{\theta}) &= \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{b\in\mathcal{A}(s)} e^{h(s,b,\boldsymbol{\theta})}}, \\ h(s,a,\boldsymbol{\theta}) &= \boldsymbol{\theta}^\top \mathbf{x}(s,a), \\ \mathbf{x}(s,a) &= \left[\mathbb{1}_{\{a=1\}}, \mathbb{1}_{\{a=-1\}}\right]^\top \end{aligned}$$

where $\boldsymbol{\theta}^\top = [\theta_1, \theta_{-1}]$ and actions are represented as -1 being 'left' and 1 being 'right'. Since the features only depend on the action (and not the state), the system is unable to detect in which state it currently is. States are numbered as $1, 2, 3, 4$, with 1 being the initial position at the extreme left and 4 being the terminal state are the extreme right.

Part (a), 15 points:

Create an `R` function `SimulCorridorEpisodeFixedPolicy` which simulates

an episode of the corridor problem and outputs its outcomes. The inputs of the function should be

- `pright`: the probability of choosing the action 'right' at any stage in the episode,
- `theseed`: the random seed used to generate the random decisions in the episode.

The output should be a list of size 2 containing the following elements:

- `stateseq`: the sequence of all states visited during the episode,
- `actionseq`: the sequence of all actions performed during the episode.

Use the function `SimulCorridorEpisodeFixedPolicy` to simulate 20,000 episodes, first with a probability of selecting right of 0.95, and then with a 0.05 probability of selecting right. For both policies, using the simulated episodes, estimate the expected returns when starting in the initial state through a first-visit approach.

Note: these values should be near the true states values of -44 and -82 outlined in Example 13.1.

Part (b), 5 points:

Show that the policy gradient is given by

$$\nabla_{\boldsymbol{\theta}} \log \pi(a, \boldsymbol{\theta}) = \begin{bmatrix} \mathbb{1}_{\{a=1\}} - \pi(1, \boldsymbol{\theta}) \\ \mathbb{1}_{\{a=-1\}} - \pi(-1, \boldsymbol{\theta}) \end{bmatrix}.$$

Part (c), 15 points:

Create an `R` function `OneRunCorridor` which a single run of `nepis` episodes of the corridor problem and applies the REINFORCE approach (see pseudo-code in p.328 of the textbook). The inputs of the function should be

- `inittheta`: the initial value during the run of the vector $\boldsymbol{\theta}$ driving the policy,
- `alphasteptheta`: the step size α for the update of the policy,
- `nepis`: the number of episodes per run,
- `theseed`: the random seed which is used to generate random numbers for each episode (note that the episodes should not all use the same seed e.g. episode j could used the seed `theseed`$+j$).

The output should be a list of size 2 containing two vectors of length `nepis`:

- `RewardEachEpisode`: its j^{th} element corresponds to the time-0 return obtained the j^{th} episode of the run,
- `ProbRightEachEpisode`: its j^{th} element corresponds to final probability of selecting the action 'right' at the end of the j^{th} episode of the run (after all policy updates are performed).

The function `OneRunCorridor` should call the function `SimulCorridorEpisodeFixedPolicy` from part (a).

Part (d), 5 points:

Use your function `OneRunCorridor` from part (c) to generate 100 runs of 1000 episodes (nepis= 1000) of the corridor problem. In each run, the initial value of $\boldsymbol{\theta}$ should be $= [log(1/0.95 - 1), 0]^\top$, which yields an initial probability of selecting 'right' of 0.05. Repeat this process three times respectively with the step size values of $\alpha = 2^{-12}$, $\alpha = 2^{-13}$ and $\alpha = 2^{-14}$.

Create two three-dimensional arrays `returnmat` and `probrightmat` of size 100 by 1000 by 3 which will store the results of each simulated run; element $[i, j, k]$ will store results for the the j^{th} episode of the i^{th} run with the k^{th} considered value of the step size α. `returnmat` contains the time-0 return

of the run, whereas `probrightmat` contains the final probability of selecting 'right' at the end of the episode (after all policy gradient updates).

Use `returnmat` and `probrightmat` to compute the average across all 100 runs of the time-0 return G_0 in respectively episode 1, episodes 2, etc. Calculate the analogous average across 100 runs of the final probability of selecting 'right' after each respective episode.

For each of the two quantities (time-0 return G_0 and probability of selecting 'right'), you should obtain three sequences (one for each value of α) of length 1000.

Plot these three sequences (versus $1, \ldots, 1000$), and compare with Figure 13.1 of the textbook for the time-0 gain G_0. You should use the same colors in your plot than in the textbook (blue is $\alpha = 2^{-12}$, red is $\alpha = 2^{-13}$, green is $\alpha = 2^{-14}$) with a corresponding legend. A gray horizontal line at $y = -11.6$ for the time-0 return and $y = 0.59$ for the probability of selecting 'right' should be drawn to denote the optimal expected return (for the optimal probability of selecting 'right').

Question: do you obtain the same results than the textbook? For all curves?

Part (e), 7.5 points:

Create a new function `OneRunCorridorBaseline` by modifying your function `OneRunCorridor` from part (c) to include a baseline in your REINFORCE algorithm (see algorithm 'REINFORCE with Baseline' on p.330 of the textbook). The baseline used is an approximation of the state-value function with four parameters $\mathbf{w} = [w_1, w_2, w_3, w_4]$. The baseline is given by

$$b(S_t, \mathbf{w}) = \hat{v}(S_t, \mathbf{w}) \equiv w_{S_t}, \quad S_t = 1, 2, 3, 4.$$

An exponential weighting scheme is used to estimate the state value function

and update the baseline:

$$\hat{v}(S_t, \mathbf{w}^{(t+1)}) \equiv \hat{v}(S_t, \mathbf{w}^{(t)}) + \alpha^{\mathbf{w}}(G_t - \hat{v}(S_t, \mathbf{w}^{(t)}))$$

$$\Rightarrow \quad \begin{cases} w_{S_t}^{(t+1)} = w_{S_t}^{(t)} + \alpha^{\mathbf{w}}(G_t - w_{S_t}^{(t)}) \\ w_s^{(t+1)} = w_s^{(t)} \quad \forall s \neq S_t, \end{cases}$$

for a chosen constant step size $\alpha^{\mathbf{w}}$.

The inputs of the `OneRunCorridorBaseline` function should be

- `inittheta`: the initial value during the run of the vector $\boldsymbol{\theta}$ driving the policy,
- `initw`: the initial value during the run of the baseline parameter vector $\mathbf{w}$,
- `alphasteptheta`: the step size α for the update of the policy,
- `alphastepw`: the step size $\alpha^{\mathbf{w}}$ for the update of the baseline parameters,
- `nepis`: the number of episodes per run,
- `theseed`: the random seed which is used to generate random numbers for each episode (note that the episodes should not all use the same seed e.g. episode j could used the seed `theseed`$+j$).

The output should be a list of size 2 containing two vectors of length `nepis`:

- `RewardEachEpisode`: its j^{th} element corresponds to the time-0 return obtained the j^{th} episode of the run,
- `ProbRightEachEpisode`: its j^{th} element corresponds to final probability of selecting the action 'right' at the end of the j^{th} episode of the run (after all policy updates are performed).

The function `OneRunCorridorBaseline` should also call the function built in part (a) i.e. the function `SimulCorridorEpisodeFixedPolicy`.

NOTE: to avoid being stuck in an almost infinite loop (a very very long episode), I suggest you to bound the probability of selecting 'right' in each step below by 0.01 and above by 0.99. In other words, if your algorithm tells you to use a probability of 0.9998, then use 0.99 instead.

Part (f), 2.5 points:

Do a task analogous to part (d) where you simulate 100 runs of 1000 episodes for the following step size values for the policy parameters $\boldsymbol{\theta}$: $\alpha^{\boldsymbol{\theta}} = 2^{-9}$ and $\alpha^{\boldsymbol{\theta}} = 2^{-12}$. Consider a baseline step size of $\alpha^{\mathbf{w}} = 2^{-6}$ with an initial value function estimate (baseline parameters) of $\mathbf{w} = [0,\, 0,\, 0,\, 0]^\top$.

Store the time-0 returns for each episode of each run (for each value of the paramater) in an array `returnmatBaseline` of dimension 100 by 1000 by 2. Again, like in part (d), average the results across the 100 runs and plot the average time-0 return after 1 episode, 2 episodes, etc, versus the sequence $1, \ldots, 1000$. Plots the two curve (one for for each value of $\alpha^{\boldsymbol{\theta}}$). Include also the curve with $\alpha = 10^{-12}$ from part (d) and plot a gray horizontal bar at $y = -11.6$ in your plot, on top of adding a legend to identify all curves. Use all the same colors for the curves than in Figure 13.2 of the textbook (except the curve with baseline and $\alpha^{\boldsymbol{\theta}} = 2^{-12}$ which you should draw in blue since this curve is not in Figure 13.2).

Question: compare your results with those of Figure 13.2 of the textbook. Do you obtain the same results(same curves)?