CONCORDIA UNIVERSITY

Department of Mathematics & Statistics

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

Reinforcement Learning

Assignment 1

Frédéric Godin, Ph.D., FSA, ACIA

January 2019

©Frédéric Godin, 2019

1 Instructions

- The assignment is done in teams of **one to three** 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 February 1, 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 experiment with the k-armed bandit framework presented in Chapter 2 of the textbook.

Question 1 (40 points)

This question aims at reproducing results presented in Section 2.3 (The 10-armed Testbed) of the textbook. More precisely, the objective is to reproduce Figure 2.2.

Part (a), 20 points:

Create a function `kArmedTestbed` which simulates a single run (i.e. the sequence of actions and corresponding rewards obtained) of the k-armed testbed. An ϵ-greedy approach is followed to select an arm at each stage. To get action value estimates, equally weighted rewards is used. For the run, the expected rewards associated to each arm should be simulated at the beginning of the run with a standard normal distribution i.e. $N(0, 1)$. Everytime an arm is sampled, the distribution of the reward from this arm is Gaussian with mean being the aforementioned simulated value and variance being 1.

The function `kArmedTestbed` should take as inputs

- `k`: the number of armed bandits in the experiment,
- `nplays`: the number of plays in a given run (i.e. the number of bandit drawn in the run),
- `epsilon`: a real number between 0 and 1 representing the probability of selecting a random action at a given step in an ϵ-greedy strategy,

- `theseed`: an integer representing the random seed used to ensure reproducibility of you simulation results,

and should output a list containing the following elements

- **First element**: `muvec`: a vector of length k containing the simulated expected reward for each arm in the run.
- **Second element**: `Rewardvec`: a vector of length `nplays` containing the reward obtained at each time step from time $t = 1$ to $t =$ `nplays`.
- **Third element**: `Actionvec`: a vector of length `nplays` containing the action performed (i.e. the selected arm) at each time step from time $t = 1$ to $t =$ `nplays`.

Note: Ensure your simulation is reproducible i.e. when you run you code multiple times you get the same answer each time. To do this, use the `set.seed` function from R within the core of your `kArmedTestbed` function. Choose any seed you want.

Part (b), 10 points:

Use the function `kArmedTestbed` from part (a) to simulate $M = 2{,}000$ runs of the k-armed testbed experiment with $k = 10$ bandits and `nplays`$= 1,000$. For each run, apply the ϵ-greedy policy. Repeat this experience three times, using the following three values for ϵ: 0, 0.01 and 0.1.

Note that every time a new run is performed, the seed passed to `kArmedTestbed` must be different (so that all runs don't give exactly the same results).

Then create two matrices `AvgRewardAllepsi` and `PercOptimalAllepsi` of size 3×`nplays`. Each row of these matrices correspond to one of the three possible values for ϵ. For matrix `AvgRewardAllepsi`, the element in column t should be the average (across the 2,000 runs) of the rewards obtained at time step (i.e. stage) t. For the matrix `PercOptimalAllepsi`, the element in column t should be the percentage of runs (across the 2,000 runs) that the optimal action was selected at stage t. Seeing which action is optimal across the k arms for a given run can be seen by looking at the `muvec` element of the list output of function `kArmedTestbed`.

Note: Depending on if your code is efficient and which computer you use to run this, it is possible this step might take a while. Don't wait until the last minute to do this part of the assignment.

Part (c), 5 points:

Use the `plot` function of `R` to generate two figures that you will put in your pdf report. The two figures are analogous to the two panels from Figure 2.2 in the textbook.

The first figure must contain the curves representing average reward obtained across all 2,000 runs at each respective stage versus the stage itself. Use the `AvgRewardAllepsi` matrix from part (b) for this. Three curves will be plotted in total: one for $\epsilon = 0$ (dark green color), $\epsilon = 0.01$ (red color) and $\epsilon = 0.1$ (blue color). Make sure to include a legend (with the function `legend`) to identify your three curves in the plot.

The second figure must contain the curves representing percentage of runs across all 2,000 runs that the optimal action was selected at each respective stage versus the stage itself. Use the `PercOptimalAllepsi` matrix from part (b) for this. Three curves will be plotted in total: one for $\epsilon = 0$ (dark green color), $\epsilon = 0.01$ (red color) and $\epsilon = 0.1$ (blue color). Make sure to include a legend to identify your three curves in the plot.

Question 2 (30 points)

This questions to armed bandits providing non-stationary rewards.

Part (a), 20 points:

Create a function `NonStationaryTestbed` by adapting your function `kArmedTestbed` from Question 1. This new function should simulate a single run of a 2-armed bandit framework. The run should involve 1,000 plays. At stage t, a reward from arm j has the distribution $N(\mu_{j,t-1}, 1)$, where the process $\{(\mu_{1,t-1}, \mu_{2,t-1})\}_{t=1}^{1000}$ is an autoregressive process evolving according to

$$\mu_{j,0} \equiv 0, \quad \mu_{j,t} \equiv \mu_{j,t-1} + 0.01 z_{j,t}, \quad t = 1, \ldots, 999$$

where $z_{j,t} \sim N(0,1)$, with z_{j_1,t_1} and z_{j_2,t_2} being independent whenever $j_1 \neq j_2$ or $t_1 \neq t_2$. The process z is independent of all other sources of randomness. An ϵ-greedy method with $\epsilon = 0.1$ is still used to determine the arms selected during the run. Depending on the value of the input `isCstStep` of the function, the action value estimates should either be computed through exponential weighting of rewards (case `isCstStep`=TRUE) where the exponential coefficient is α=`alphaparam`, or otherwise (case `isCstStep`=FALSE) through equally weighted rewards.

The function `NonStationaryTestbed` should take as inputs

- `isCstStep`: a boolean (true or false) variable determining if you apply a constant step size or a sample average method to update your action value estimates,
- `alphaparam`: the coefficient of the step size you want to use if you use constant step sizes to estimate action values,

and should output a list containing the following elements

- **First element**: `mumat`: a matrix of size 1,000 by 2 containing the expected reward for each arm at each stage in the run.
- **Second element**: `Rewardvec`: a vector of length `nplays` containing the reward obtained at each time step from time $t = 1$ to $t =$ `nplays`.
- **Third element**: `Actionvec`: a vector of length `nplays` containing the action performed (i.e. the selected arm) at each time step from time $t = 1$ to $t =$ `nplays`.
- **Fourth element**: `Qmat`: a matrix of size (`nplays`+1) by 2 whose element at row i and column j is the action value estimate associated with arm j after $i - 1$ stages.

Note: Ensure your simulation is reproducible i.e. when you run you code multiple times you get the same answer each time. To do this, use the `set.seed` function from R within the core of your `NonStationaryTestbed` function. Choose any seed you want.

Part (b), 5 points:

Use the function `NonStationaryTestbed` to generate two runs of the 2-armed bandit problem:

1. A first run with the exponentially weighted rewards methods for estimating action value estimates, with $\alpha = 0.025$.
2. A second run with equally weighted rewards.

Note: the same random numbers should be used for both runs so that when the same arm j is selected in both runs at a given stage t, the reward should be the same for both runs.

Use the `plot` function of `R` to generate two figures (on figure for each arm) that you will put in your pdf report.

The j^{th} figure associated with the j^{th} arm, $j = 1, 2$, must contain the curves representing the true and estimated expected reward for arm j at each respective stage versus the stage itself. Three curves will be plotted in total: the true expected reward (dark green color), the estimated action value with exponentially weighted rewards (red color) and the estimated action value with equally weighted rewards (blue color). Make sure to include a legend (with the function `legend`) to identify your three curves in each plot.

Part (c), 5 points:

Interpret your results from part (b): Which method performs better predictions for the action value? Why?

Question 3 (30 points)

Part (a), 25 points:

Simulate a single run of the 3-armed bandit framework. The run must have 1,000 plays. Rewards from the arm j are normally distributed $N(\mu_j, 1)$, where $\mu_1 = -0.5$, $\mu_2 = 0$, $\mu_3 = 0.5$.

This time, instead of using an ϵ-greedy strategy, the action-preference based method presented in the slides will be used to select the actions. The parameter $\alpha = 0.02$ is used for the evolution of action preferences through time. Initial action preferences are set to zero for all actions i.e. $H_1(a)$ for all a.

Store the action preferences for each action and each stage in a matrix `Hmat` of size 1,001 by 3. The element in row j and column i of this matrix should be the action preference associated with arm i after $j - 1$ plays.

Similarly, define a matrix `Pimat` of the same size than `Hmat`. The element in row j and column i of matrix `Pimat` should be the probability after $j - 1$

plays to choose arm i for the next play.

Note: Ensure your simulation is reproducible i.e. when you run you code multiple times you get the same answer each time. To do this, use the `set.seed` function from R. Choose any seed you want.

Part (b), 5 points:

Use the `plot` function of `R` to generate a figure that you will put in your pdf report.

The figure must contain curves representing the simulated action preference of a given arm at each stage of the run. Use the `Pimat` matrix from part (a) for this. Three curves will be plotted in total: one for arm 1 (dark green color), one for arm 2 (red color) and one for arm 3 (blue color). Make sure to include a legend (with the function `legend`) to identify your three curves in the plot.