Master M2 MVA 2015/2016 - Reinforcement Learning - TP 3

Adversarial bandits

By: Lilian Besson (`lilian.besson at ens-cachan.fr`). Attached programs: The programs for this TP are included in the `zip` archive I sent, and are regular **MATLAB/Octave** programs[1].

---

1 EWT and EXP3 versus an oblivious adversary

Rem: The EXP3 strategy needs two parameters, η, β, and note that the last EXP3 algorithm seen in class used *three* parameters, η, β and γ. We were not sure the value to chose for γ, and not sure if the lecture slides and the TP assignment used to same convention. We assumed that γ (from the slide) is β in the programs, and that β (from the slide, the offset on $\hat{X}_i^n$) is zero.

Question 1

We compared the two algorithms EWT (which knows the full game matrix) and EXP3 (which observe only one reward after choosing an action), for these three adversarial settings:

- A constant setting, for example player B (the opponent) always chooses the go to the bar (action 2). We expect EWF and EXP3 to be almost as efficient, as they will both quickly detect that action 1 is the best for player A.
- A "tricky" setting, in the sense that B will choose his worse action (action 2) for a while (to make A believe that action 2 is the best, because $R_A(1,2) = -1 < 1 = R_A(2,2)$), and after only plays action 1 (the best action for him). We expect EWF to be really quicker to react to this change of behavior than EXP3.
- A "random" setting, ie. each action for B is chosen at random. We expect both algorithm the be quite ineffective, with a high regret.

The plots included below were drawn for a time horizon of $n = 500$, and for which we chose $\eta = \beta = \sqrt{(2\log(2)/((\exp(1)-1)n))}$ and $\gamma = 0$, as given by a formula in the slide (to minimize the error bound). Remark that η, β *have* to vary with n, and they have to be non-increasing and $\to 0$ when $n \to +\infty$ (which is indeed the case for this formula).

Below in Figures 1, 2 and 3 are plotted the observed regrets, of the two algorithms EWF versus EXP3, playing successively against the three strategies described above:

[1] Note: I only tested my programs with GNU Octave, and version `v4` at least is required – the `classdef` feature is new in Octave version 4.0.

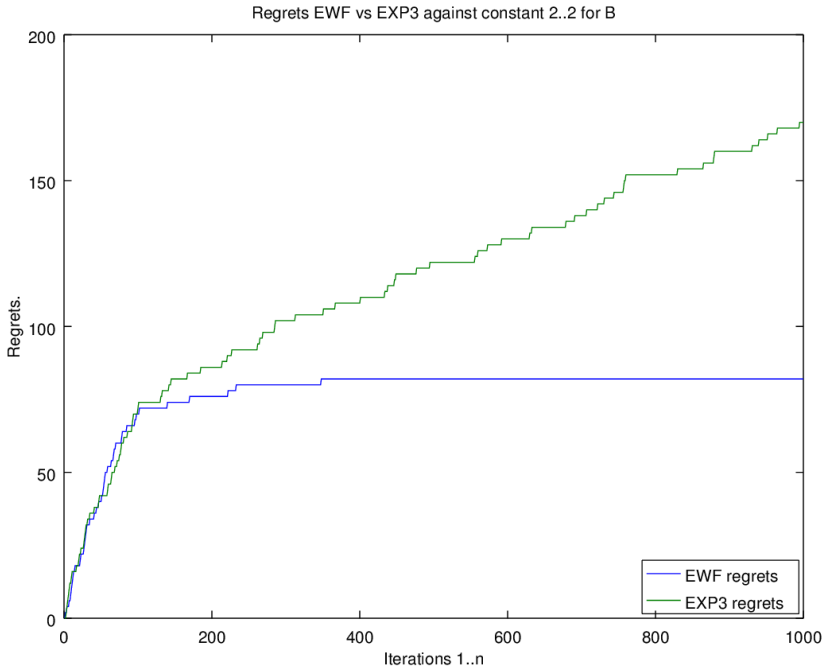

Figure 1: Regrets of EWF versus regrets of EXP3 against a constant adversary B.

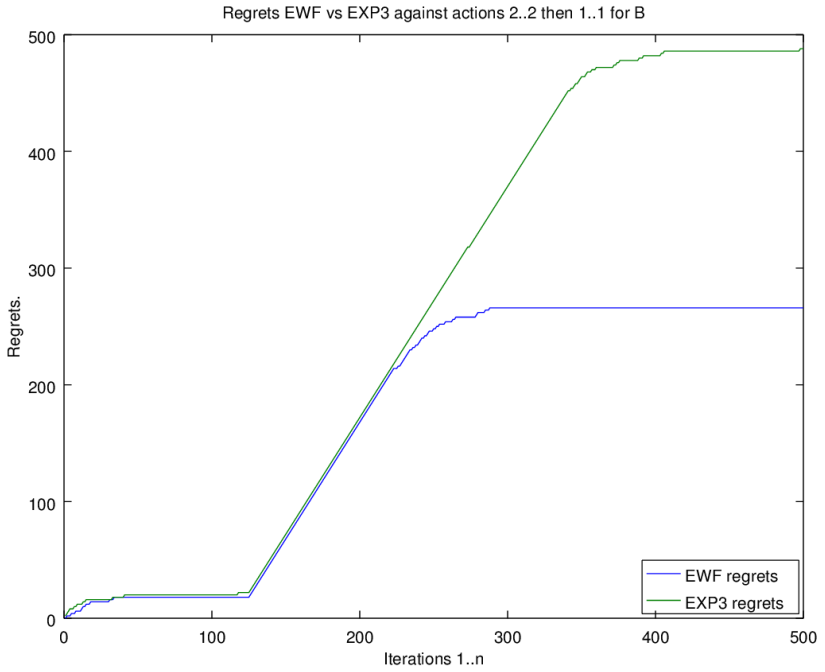

Figure 2: Regrets of EWF versus regrets of EXP3 against a tricky adversary B.

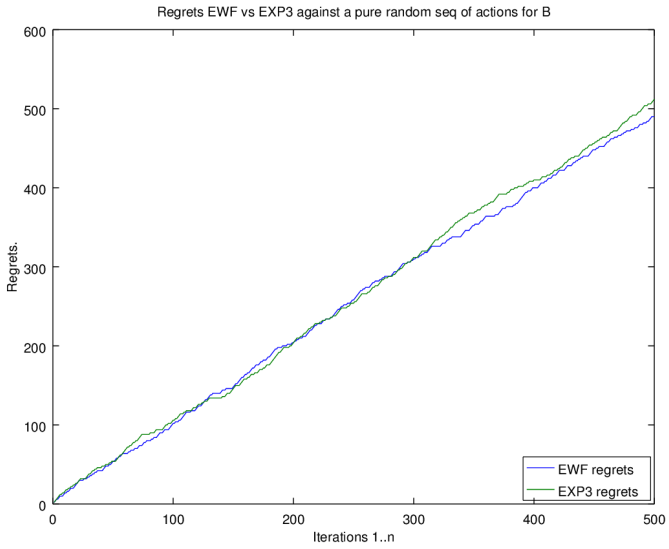

Figure 3: Regrets of EWF versus regrets of EXP3 against a random adversary B.

The plots show exactly the behavior we expected: EWF is better than EXP3 (it reads the entire G when EXP3 read one value at a time), EXP3 is really bad against a constant "stupid" player when EWF is optimal, and both are as inefficient against a random player.

---

2 EXP3 versus EXP3: Nash equilibrium

See the file `EXP3vEXP3.m` (it is well commented) for the implementation, and the second part in the main file `mainTP3.m` for the demo.

Question 2

Below the core of that simulation, where two EXP3 objects play against each other:

```
for t = 1:n  % Time horizon n
    % A and B plays, both have same interface
    % (but internal weights will evolve differently)
    actionA = exp3A.play();
    ActionsA(t) = actionA;
    actionB = exp3B.play();
    ActionsA(t) = actionB;

    % Get reward for these two actions
    reward = G(actionA, actionB);
    RewardsA(t) = reward;
    % Both player reacts, respectively to r and -r
```

```
    exp3A.getReward(reward);
    exp3B.getReward(-reward);  % R_B(i,j) = - R_A(i,j)
end;
```

In the 3rd part in the `mainTP3.m` file, we then compute the required quantity $p_{a,n}, p_{b,n}$ and $p_a^\star$ and $p_b^\star$ like this (respectively estimated probability of player A or B to pick the first action, and last estimation is used as optimal $p^\star$):

```
% Run the EXP3 vs EXP3 simulation
[ActionsA, ActionsB, RewardsA] = EXP3vEXP3(n, eta, beta, G);

% Compute p_A,n and p_B,n and 'optimal value' p_A^* p_B^*
p_A_n = cumsum(ActionsA == 1) ./ (1:n);
p_A_star = p_A_n(end);
p_B_n = cumsum(ActionsB == 1) ./ (1:n);
p_B_star = p_B_n(end);
```

For this simulation, we observe a bigger time horizon of $n = 5000$, and use the same formula for $\eta \simeq 0.0127$ but chose a bigger $\beta \simeq 1.27$.

Below in Figure 4 is plotted the observed mean frequencies of choosing the action 1 for player A and B (on 20 simulations).

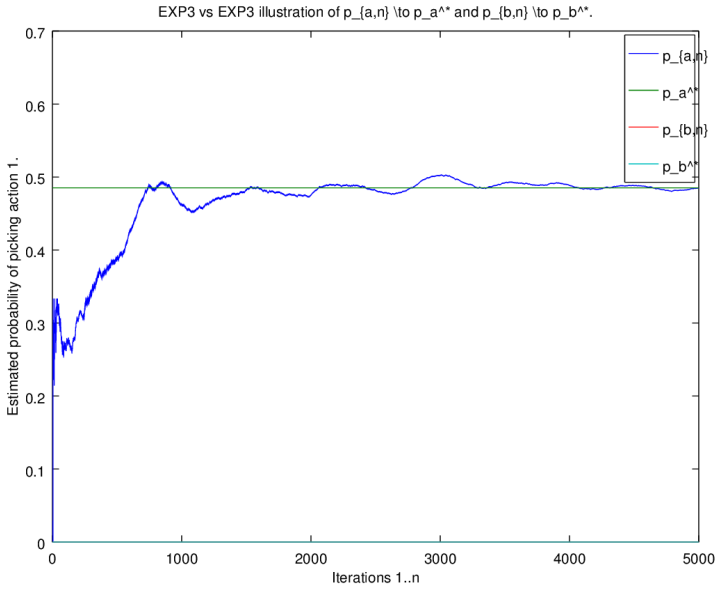

Figure 4: Checking that $p_{a,n}$ and $p_{b,n}$ resp. converge to $p_a^\star \simeq 0.48$ and $p_b^\star \simeq 0$.

As expected, we observed indeed that these mean empirical frequencies $p_{a,n} = \frac{1}{n}\sum_{t=1}^n \mathbf{1}_{(A_t=1)}$ (choices of 1 for player A) and $p_{b,n} = \frac{1}{n}\sum_{t=1}^n \mathbf{1}_{(B_t=1)}$ (choices of 1 for player B) do converge to two values, respectively $p_a^\star \simeq 0.48$ and $p_b^\star \simeq 0$.

The value of the game is the limit of $\sum_{t=1}^{n}(R_A(A_t, B_t))/n$, which is computed in Octave and plotted as `cumsum(RewardsA) ./ (1:n)`:

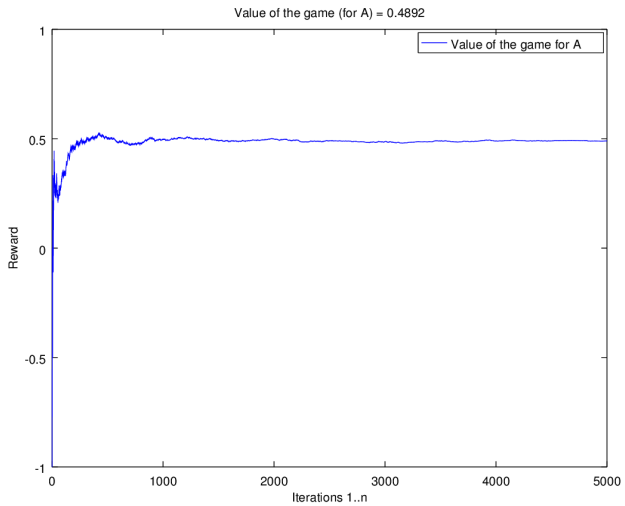

Figure 5: Checking that $\sum_{t=1}^{n}(R_A(A_t, B_t))/n$ converges to 0.489.

We computed the value of the game to be 0.489. In fact, because $p_{b,n} \to_{n\infty} p_b^\star \simeq 0$ and due to the special value of G (best A-reward is 1), it is satisfactory to observe the value to be about $\mathcal{V}(G) \simeq 1 \times p_a^\star$, and $p_{a,n} \to_{n\infty} p_b^\star \simeq 0.48$.

Remark: I think in practice this value depends on the numerical values chosen for the parameters (η, β etc), but I guess it should have converged to $1/2$ (because of the special shape of the matrix G). So finding a limit value of 0.489 is correct, with a relative error of 2.2%.

---

3 Stochastic bandit or adversarial bandit

This time, the file `Thompson.m` was given. We wrote the `EXP3Stochastic` function by following the explanations in the slides.

Below is included the key part of that simulation, showing what happens at each time step:

```
actionA = exp3.play(); % A plays
Actions(t) = actionA;
% Different from EXP3play: B plays from the MAB
% Reward X_{i,t} is drawn from the arm chosen by A
X_it = MAB{actionA}.play();
Rewards(t) = X_it;     % A recieves reward from this action
exp3.getReward(X_it); % A updates its weights this action
```

Question 3

To conclude, we wanted to compare the two algorithms: Thompson Sampling versus a stochastic EXP3. Thompson sampling one only works for Bernoulli MAB, so we first chose a "simple" Bernoulli MAB (where EXP3 will be able to quickly identify the best "arm" ie. the best action to play), and an "harder" one when we guess that Thompson Sampling will be more efficient[2].

- "Easy" Bernoulli MAB, 3 arms of parameters $0.5, 0.7, 0.4$. Best one has mean 0.7 (quite far from the other arms). Complexity was computed to be 7.8.
- "Hard" Bernoulli MAB, 4 arms of parameters $0.43, 0.45, 0.49, 0.48$. Best one has mean 0.49 (really close from the other arms). Complexity was computed to be 199.73.

On the plot below (Figure 6), we compare the rewards of **Thompson Sampling** with the rewards of **EXP3**, for *one* simulation. We observe what we were expecting: EXP3's performance is not very far from Thompson's, and they seem to have the same behavior on the harder MAB problem:

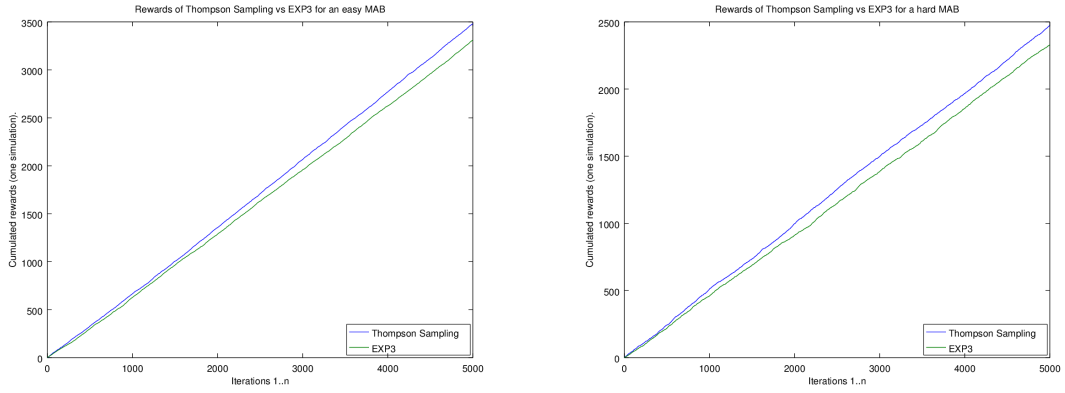

Figure 6: *One run* of Thompson Sampling vs EXP3, "easy" (left), and "hard" (right) Bernoulli MAB.

And the next plot (Figures 7 and 8) compare the average regrets (for 20 simulation[3]). We clearly observe what the previous plot started to show: in average, Thompson is really better than EXP3 (at least for an easy MAB). It is harder to interpret the result of the plot on the right for the harder MAB, where both algorithms perform "as badly".

[2] I chose the same Bernoulli Multi-Armed Bandits problems as in the TP2, in order to also compare UCB versus EXP3. Long story short: for simple MAB, UCB beats Thompson which beats EXP3, but for harder MAB, it's the opposite.

[3] All these simulations were really slow so it would have taken a lot of time to do more...

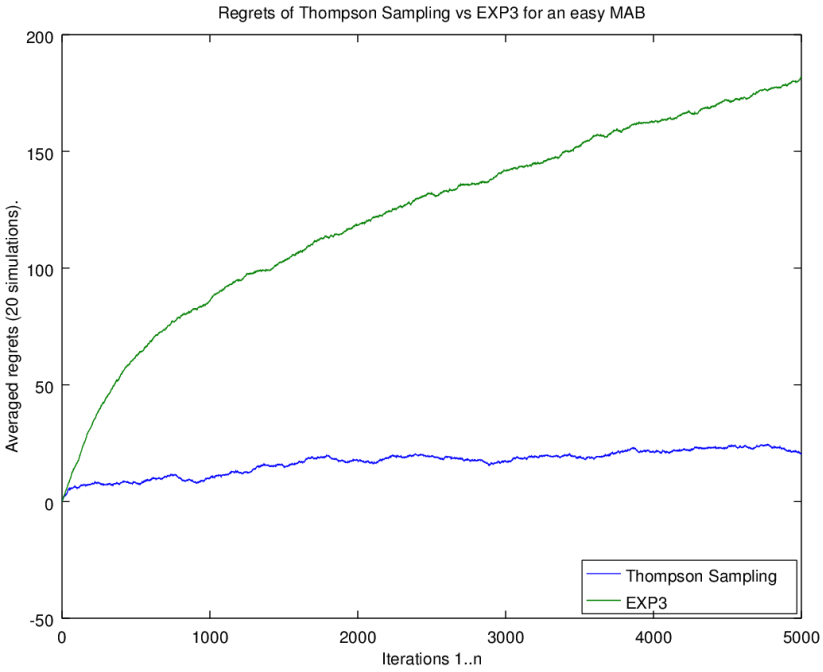

Figure 7: Comparing mean regrets of Thompson Sampling vs EXP3, on a "easy" Bernoulli MAB ($c = 7.8$), on $N = 20$ simulations.

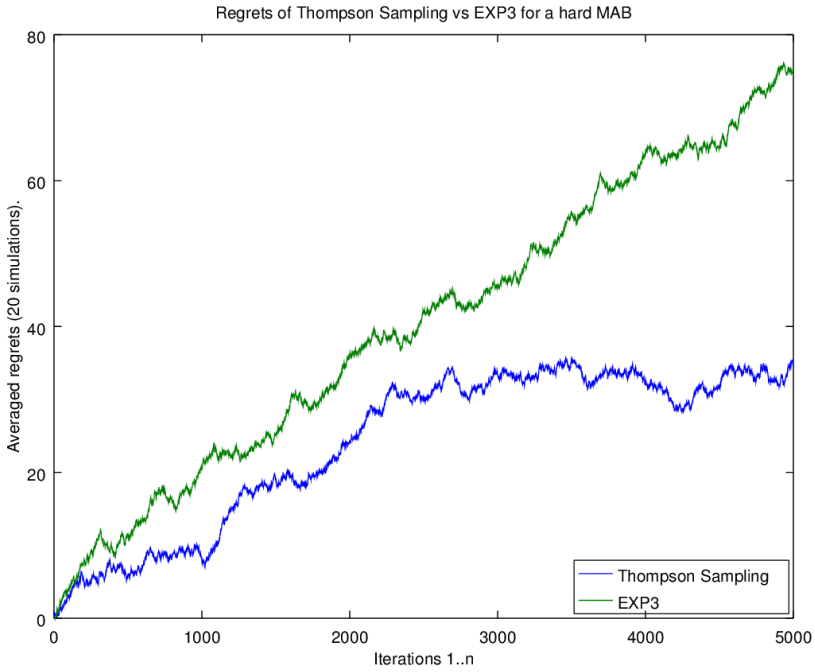

Figure 8: Comparing mean regrets of Thompson Sampling vs EXP3, "hard" (right) Bernoulli MAB ($c = 199.7$), on $N = 20$ simulations.

Finally, to conclude, we observed that the exploratory/exploitation algorithm (`EXP3`) seems more interesting for harder Bernoulli bandit problems (it is closer to Thompson) but never appear better that the Thompson Sampling method.

Additionally, on harder problems, both performs more "erratically", which is seen in the plot on the right because the curve is really less smooth in comparison to the one for easy MAB problem.

Conclusion

The TP was interesting, thanks.

I wanted to try apply the EXP3 vs EXP3 on a more complex zero-sum game (not 2 players, or more than 2×2 rewards) but it would have require a (way) more complex algorithm. However, I was able to use it for another example of 2×2 zero-sum game (another matrix G, $3 \times G$), and observed similar results (the value of the game was $\simeq 1.5$ and same behavior when comparing EXP3 and Thompson Sampling).

Remark: Inspired by the Object Oriented point of view we took last time to implement the arms, I implemented the `EWFplay`, `EXP3play` and `EXP3vEXP3` functions thanks to two objects, `EWF.m` and `EXP3.m` (which provides two methods: `.play()` to draw an arm, and `.getReward(r)` to update the weights).