Rejection Sampling for Off-Policy Learning

Wesley Chung, Sina Ghiassian, Somjit Nath and Martha White
Department of Computing Science
University of Alberta
Edmonton, Canada
{wchung, ghiassia, somjit, whitem}@ualberta.ca

Abstract

Importance sampling has been the basis for most popular off-policy value function estimation algorithms. Despite this popularity, it can suffer from high-variance and instability issues. In this paper we investigate rejection sampling as an alternative to importance sampling to correct off-policy temporal difference updates. For TD(0), we show a connection between the two algorithms, indicating that both approaches should behave similarly in the long run though rejection sampling can save computation by rejecting some updates. We compare both algorithms empirically in a continuing environment with multiple value functions learned in parallel and find that rejection sampling achieves almost identical performance to importance sampling with significantly fewer updates, confirming the theory.

1 Introduction

Off-policy learning enables an agent to learn about many (target) policies simultaneously, even though it can only act according to a single behaviour policy. This ability is key for a continual-learning agent, which only receives one stream of interaction. For example, the agent may want to learn about how to reach different subgoals [15, 1]. This problem can be divided into learning different optimal policies, off-policy. The agent might also simply want to make many policy-contingent predictions about the future, such as in Horde [18] or Unreal [4]. To obtain each prediction, the agent needs to learn a value function for each policy; with many policies, this can only practically be achieved using off-policy learning.

The most common algorithms for off-policy learning combine importance sampling (IS) with temporal difference (TD) update rules [12, 13]. Each update is multiplied by the IS ratio: the ratio between action probabilities of the target and behaviour policies. These methods can suffer from high variance, especially under a significant mismatch between target and behaviour [14, 7].

Rejection sampling (RS) [19] is a well-known alternative to IS for correction the sampling distribution, but has been under-explored in reinforcement learning. There have been a few mentions of RS for TD [17, 16, 6], though none of these works use it. RS has been used for evaluating policies using a fixed batch of data [5, 9], where the goal is to obtain a single score for a policy rather than learn the value function for a policy. The lack of rejection sampling algorithms for learning value functions may be due to the fact that RS is often inferior to IS for estimating expectations [3]. RS, however, can be more computationally efficient [11], which could be of significant benefit in a continual learning setting. Further, different properties may arise when using RS or IS for TD, because each update changes a set of parameters rather than a sample average, and so can influence the trajectory of parameters.

In this paper, we investigate the use of RS as an alternative to IS for off-policy learning. We first show how to use RS for learning value functions, as there are some subtleties in its application to TD. We then show a connection to IS, in particular, when rescaling the step sizes by the maximum importance

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

sampling ratio for TD with IS. In fact, the rescaled IS step size is equal to the expected step size from rejection sampling, suggesting that both algorithms should behave similarly. Empirically, we find that rejection sampling achieves very similar performance to importance sampling in terms of error but with much fewer updates.

2 Problem Formulation

We assume a standard Markov Decision Process framework for reinforcement learning [20], with states $\mathcal{S}$, Actions $\mathcal{A}$ and transitions P such that when in a (random) state S_t, after taking action A_{t+1}, the agent transitions to S_{t+1} and receives reward R_{t+1} according to $P(S_{t+1}, R_{t+1}|S_t, A_t)$. We are interested in policy evaluation: estimating the state-value function $V(s)$—the expected discounted sum of rewards from state $s \in \mathcal{S}$ with discount $\gamma \in [0, 1]$—under a fixed target policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. In the off-policy setting, we assume the agent chooses actions using a behaviour policy $\mu : \mathcal{S} \times \mathcal{A} \to [0, 1]$.

We consider the case of linear function approximation with temporal difference (TD) updates. In linear function approximation, the true values $V(s)$ are approximated with $\hat{V}(s) = \boldsymbol{\theta}^T \mathbf{x}(s)$ where $\mathbf{x}(s)$ are the features for state s and $\boldsymbol{\theta}$ is a vector of parameters. The on-policy TD(0) algorithm does updates of the form, with stepsize $\alpha_t > 0$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \delta(S_t, R_{t+1}, S_{t+1})\mathbf{x}(S_t) \qquad \triangleright\, \delta(S_t, R_t + 1, S_{t+1}) = R_{t+1} + \gamma \hat{V}(S_{t+1}) - \hat{V}(S_t)$$

These two choice are for simplicity in this short paper; all the proposed approaches can be used with other algorithms for learning value functions, and with nonlinear function approximation.

Our goal is to obtain unbiased samples of the TD update, even when following behaviour $\mu \neq \pi$. The above update can be interpreted as a sample from the *expected* TD update:

$$\mathbb{E}_\pi \left[\delta(S_t, R_{t+1}, S_{t+1})\mathbf{x}(S_t)\right] = \sum_{s,a,r,s'} d_\mu(s)\pi(a|s)P(s', r|s, a) \left[\delta(s, r, s')x(s)\right] \tag{1}$$

where $d_\mu : \mathcal{S} \to [0, 1]$ is the stationary distribution over states for the behaviour policy. Typically, unbiased estimates in off-policy learning have been obtained using importance sampling (IS), with update $\rho(A_t|S_t)\delta(S_t, R_{t+1}, S_{t+1})\mathbf{x}(S_t)$ for IS ratio $\rho(A_t|S_t) = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$. This update corrects the action distribution, so that it is as if actions where taken according to π, because

$$\begin{aligned}\mathbb{E}_\mu \left[\rho(A_t|S_t)\delta(S_t, R_{t+1}, S_{t+1})\mathbf{x}(S_t)\right] &= \sum_{s,a,r,s'} d_\mu(s)\mu(a|s)\frac{\pi(a|s)}{\mu(a|s)}P(s', r|s, a) \left[\delta(s, r, s')x(s)\right] \\ &= \mathbb{E}_\pi \left[\delta(S_t, R_{t+1}, S_{t+1})\mathbf{x}(S_t)\right]\end{aligned}$$

In the next section, we consider another strategy to do so: rejection sampling.

3 Rejection Sampling

Rejection sampling (RS) [19] is a well-known Monte Carlo method for sampling from a target distribution $p(y)$ when only having access to samples from a different proposal distribution $q(y)$. For a fixed $L > 0$, first y is sampled from q, then a uniform (0,1) random number u is sampled and finally the sample y is rejected if $\frac{p(y)}{Lq(y)} \geq u$, and accepted otherwise. The intuition is that q is scaled up by L, so that it completely lies above p for all y. Then, by rejecting the right proportion of samples—all the points between the curves $p(y)$ and $Lq(y)$—the distribution of accepted samples matches the target. The probability that RS accepts a sample is $1/L$. Thus, to maximize the sample efficiency, we want to choose L as small as possible. On the other hand, to obtain an unbiased estimate, we need L to satisfy $p(y) \leq Lq(y),\ \forall y$. To satisfy both criteria, a reasonable choice is $L = \sup_y p(y)/q(y)$. For discrete distributions p and q, it is easy to find the minimal L.

To adapt RS to the reinforcement learning setting, we treat each observed transition (state, action, reward, next state) as a sample from a proposal distribution and then either accept it—and do a TD update—or reject it and do nothing. It is natural to choose a proposal and target for each state s separately, giving $p(a) = \pi(a|s)$ and $q(a) = \mu(a|s)$. However this does not provide an unbiased

sample with RS. Rather, we need to choose $p(s,a) = d_\mu(s)\pi(s|a)$ and $q(s,a) = d_\mu(s)\mu(s|a)$. The ratio remains the same, $\frac{d_\mu(s)\pi(a|s)}{d_\mu(s)\mu(a|s)} = \frac{\pi(a|s)}{\mu(a|s)}$; however, the choice of L is different because $L = \max_{s,a} \frac{d_\mu(s)\pi(a|s)}{d_\mu(s)\mu(a|s)} = \max_{s,a} \frac{\pi(a|s)}{\mu(a|s)}$. For the incorrect choice—which we call RS-local—we would be picking a different L per state, $L(s) = \max_a \frac{\pi(a|s)}{\mu(a|s)}$ and would be implicitly obtaining a sample of the expected update $\sum_{s,a,r,s'} \frac{d_\mu(s)}{L(s)} \pi(a|s) P(s', r|s, a)$.

Practically, to apply RS, we need to estimate L. In some cases, this information maybe known upfront, for example if both policies are ϵ-greedy. Alternatively, we could estimate L online by keeping a running maximum [2]. Note that for RS-local, $L(s)$ is straightforward to compute and is likely an algorithm that would be considered in practice. Though it is biased, it does correspond to a reweighted objective that de-emphasizes states where π and μ do not match well, which could actually be advantageous. We include it in our experiments, therefore, to assess its viability as an alternative.

Finally, we provide a unified view of RS and IS, by scaling the TD update with a potentially random non-negative scalar $\beta(S_t, A_t)$,

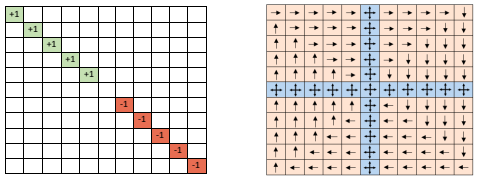
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\beta(S_t, A_t)\delta(S_t, R_{t+1}, S_{t+1})\mathbf{x}(S_t)$$

If we treat $\beta(S_t, A_t)$ as a Bernoulli random variable with probability $\rho(A_t|S_t)/L$ of being 1, we recover rejection sampling, since $\beta(S_t, A_t) = 0$ is equivalent to rejecting a sample. If we set $\beta(S_t, A_t) = \rho(A_t|S_t)/L$, then we get a rescaled version of TD with IS, where the step size is divided by L. Notice that the expected value of $\beta(S_t, A_t)$ for RS is identical to the fixed $\beta(S_t, A_t)$ for IS, and so RS can be seen as a sampled variant of this rescaled IS. This suggests that if the stepsize is appropriately scaled by $1/L$ for IS, then both algorithms should behave similarly, with RS saving computation by rejecting some updates.

4 Experiments

For our experiments, we set up a continuing gridworld task with multiple value functions to be learned at once and change the behaviour policy at several points in time. This is closer to a continual learning problem, since we want to learn about multiple policies simultaneously but can only act according to one behaviour policy at a time. In this 11×11 gridworld, depicted in Fig 1 a, the agent gets a reward of +1 f it leaves certain good states and -1 for a few bad states, with no reward elsewhere. The actions for each state are moving up, down, left or right with deterministic transitions. Making a move that would take the agent past the boundaries of the gridworld have no effect on its position.

There are 9 different policies for which we want to learn the value function (discounted by $\gamma = 0.9$). Policy 0 takes actions uniformly at random in any state. Policies 1-4 move clockwise around the gridworld (see Fig. 1 b). The main action (moving clockwise) is taken with different probabilities for each policy, starting with 0.4 for policy 1, 0.7 for policy 2, 0.85 for policy 3 and 0.97 for policy 4. The remaining three actions are taken with equal probability. Policies 5-8 have the same probabilities as policies 1-4 but move the agent counterclockwise instead. Again, policy 5 is closest to uniform random while policy 8 is the most focused in moving counterclockwise. For all the policies, the states in the sixth column or row have a uniform distribution over actions.



(a) (b)

Figure 1: (a) Rewards in the gridworld. (b) The main actions of a clockwise policy.

To evaluate the agent, we initialize it to a starting state chosen uniformly at random and follow a sequence of policies for a total of 90000 steps. We present results for two policy sequences. The first—labelled *cycle*—goes through each policy in order from 0 to 9, for 2500 steps each. The second—labelled *hard*—alternates between using policy 4 and 8, the most extreme policies for 2500 steps each time. The evaluation metric is the root mean-squared error (RMSE) between the approximate value function and the true value function, averaged over all states. We compare four different agents by considering all combinations of two settings: (a) RS vs. IS and (b) using a global constant L (correcting for state and action distributions) vs. a local constant L (correcting only for actions). All agents use tabular features.

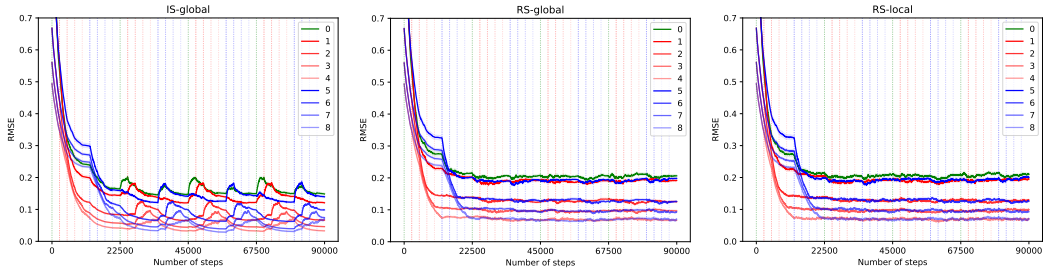


Figure 3: Learning curves for IS-global, RS-global and RS-local with the *cycle* policy sequence and tabular features. The vertical bars are placed whenever the policy changes every 2500 steps with the colours matching the policies being used. Both IS-local and RS-local are nearly identical to IS-global and RS-global, respectively, and so are omitted.

In Figure 3, we see that the learning curves are very similar for all settings. Table 2 indicates that there is, in fact, a slight edge for IS compared to RS. This can be attributed to the slight up and down dips that one can see in the IS-global plot, when the agent has reached its asymptotic performance. The dips down are due to the step sizes being smaller when doing off-policy updates with IS. RS does not have this rescaling so the error stays at the same level, no matter which policies are used. If we focus on the asymptotic error when the behaviour and target policies match and the additional step size β is 1 for both IS and RS (e.g. policy 0 for steps 45000-47500), then we see that the errors are close.

Algorithm	Cycle	Hard
IS-global	0.128	0.363
RS-global	0.161	0.372
IS-local	0.141	0.363
RS-local	0.160	0.367

Figure 2: Root mean-squared error for all four algorithms and the two policy sequences averaged over all value functions and timesteps.

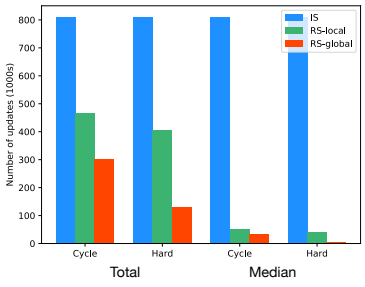


Figure 4: Total number of updates done by each algorithm over 90000 steps. Left: Total sum over all 9 value functions. Right: The median number of updates over the 9 value functions.

For the *hard* policy sequence, the performance of IS and RS are almost identical. From table 4, we find that for this setting, RS only needs to update a fraction of the time for this level of performance. In particular, looking at RS-global, we see that it only performs about 130 thousand updates in total, compared to the 810 thousand done by the IS methods. Interestingly, even though RS-local performs a significantly greater number of updates, the error is only slightly lower than RS-global for both policy sequences. This is probably due to the fact that the increased number of updates are mostly concentrated on a subset of the states, the blue states in Fig. 1 for which the target and behaviour policy match, but TD algorithms require updates to be done on sequences of states to propagate the rewards properly.

5 Conclusion

Rejection sampling (RS) can be a promising alternative to importance sampling (IS). RS can give computational benefits by doing fewer updates while maintaining almost identical performance to IS. These benefits are more pronounced as the behaviour and target policies differ to a greater extent, because when the IS ratio is small, RS will often reject the update whereas IS always updates, even though the step size is substantially scaled down. In a setting where the behaviour cycled through 9 different policies, corresponding to the target policies, RS reduced the number of updates by as much as a factor of 3. For an even more off-policy setting, where some target policies more consistently did not match the behaviour policy, the reduction was up to a factor of 30x fewer updates.

RS has other potential benefits not yet explored in this work. For example, with RS, it may be possible to take advantage of the fact that the updates are not weighted, as though they are being sampled on-policy. It is possible that application of standard stepsize adaptation strategies will be more effective for such unweighted updates. We are also currently working on extending rejection sampling to n-step TD methods and eligibility traces. In those settings, the variance issues of importance sampling are greatly magnified [10, 8] and we believe RS may offer a more stable alternative.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[2] B. S. Caffo, J. G. Booth, and A. C. Davison. Empirical supremum rejection sampling. *Biometrika*, 89(4): 745–754, 2002.

[3] Y. Chen. Another look at rejection sampling through importance sampling. *Statistics & probability letters*, 72(4):277–283, 2005.

[4] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[5] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[6] B. Liu. *Algorithms for First-Order Sparse Reinforcement Learning*. PhD thesis, 2016.

[7] A. R. Mahmood, H. P. van Hasselt, and R. S. Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

[8] A. R. Mahmood, H. Yu, and R. S. Sutton. Multi-step off-policy learning without importance sampling ratios. *arXiv preprint arXiv:1702.03006*, 2017.

[9] T. Mandel, Y.-E. Liu, E. Brunskill, and Z. Popovic. Offline evaluation of online reinforcement learning algorithms. 2016.

[10] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[11] A. B. Owen. *Monte Carlo theory, methods and examples*, chapter 9. 2013.

[12] D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[13] D. Precup, R. S. Sutton, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. 2001.

[14] D. Precup, C. Paduraru, A. Koop, R. S. Sutton, and S. P. Singh. Off-policy learning with options and recognizers. In *Advances in Neural Information Processing Systems*, pages 1097–1104, 2006.

[15] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[17] R. S. Sutton, H. R. Maei, and C. Szepesvári. A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in neural information processing systems*, pages 1609–1616, 2009.

[18] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

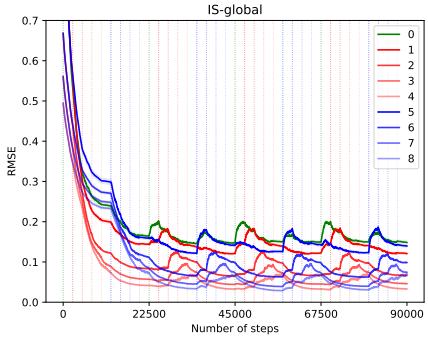
[19] J. von Neumann. Various techniques used in connection with random digits. In *Monte Carlo Method*, pages 36–38. National Bureau of Standards Applied Mathematics Series, 12, 1951.

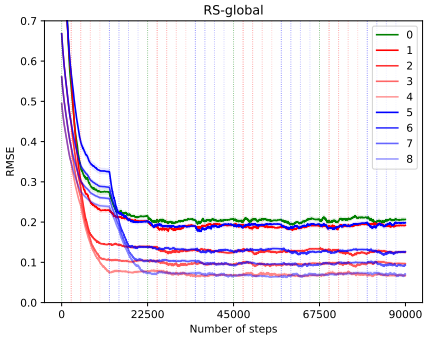
[20] M. White. Unifying task specification in reinforcement learning. *arXiv preprint arXiv:1609.01995*, 2016.

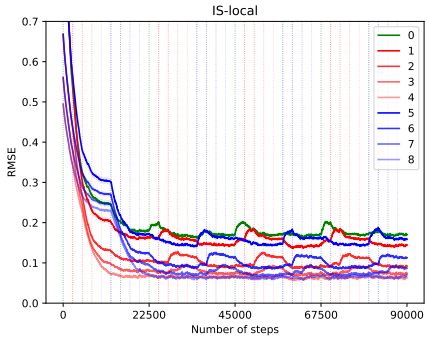
A Other Experiments

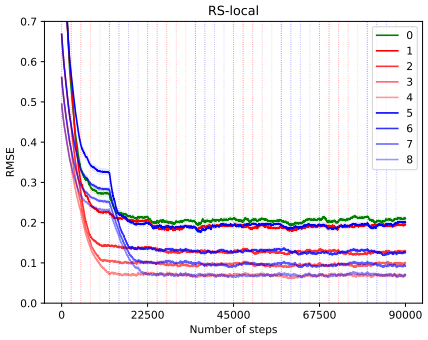
Here we include the learning curves for the algorithm settings omitted from the main paper.

***Cycle* policy sequence** (some figures duplicated from main paper for ease of comparison)

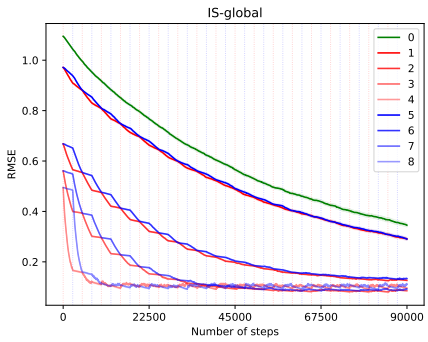


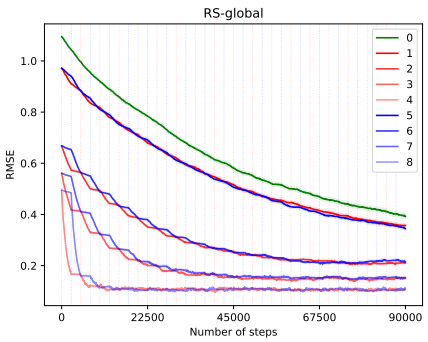


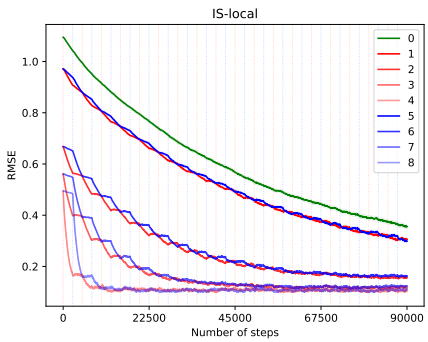


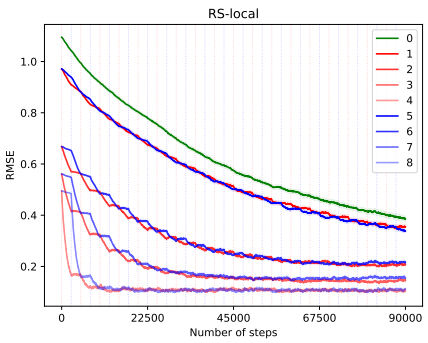


***Hard* policy sequence**





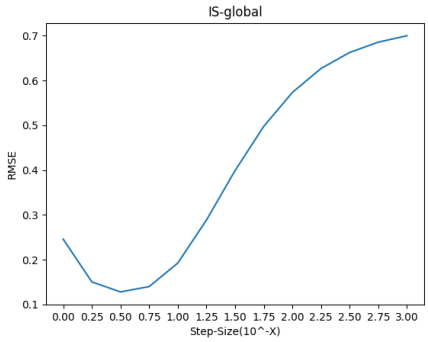


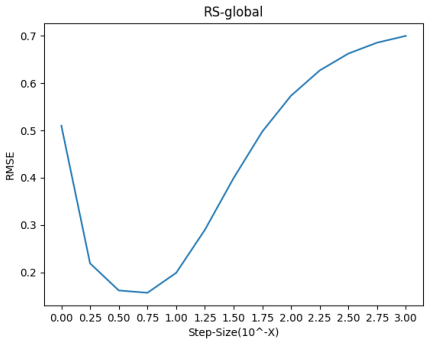


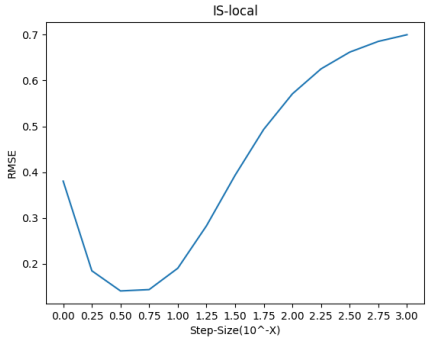
A.1 Step size sensitivity

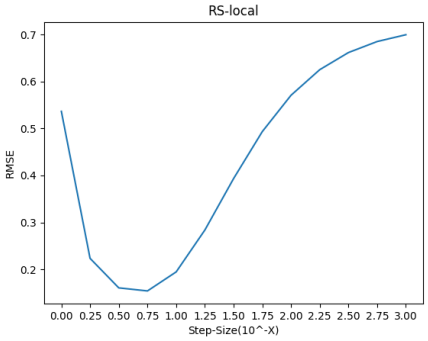
Here, we include plots of the average RMSE over all steps and all value functions for different step sizes. We find that the curves for importance sampling and rejection sampling are similar.

***Cycle* policy sequence**









***Hard* policy sequence**