Approximation of the Action-Value Function of a Markov Decision Process using Gaussian Processes

by Marco Sicklinger, Marco Sciorilli

26 September 2022

Markov Chain

A Markov Chain (or **Markov Process**) is a stochastic model that describes a sequence of possible events in which the probability of the single event depends only on the state characterizing the previous event (Markov Property):

$$P(X_n | X_{n-1}, X_{n-2}, \ldots, X_0) = P(X_n | X_{n-1})$$

Markov Decision Process

A **Markov Decision Process** is an extension of a Markov Chain.

An MDP can be defined as a tuple (S, A, P_a, R_a, γ), where:

- S is the set of states (state space)
- A is the set of possible actions (action space)
- P_a is the set of possible transitions $P_a(s, s') = \Pr(s_{t+1}=s' | s_t=s, a_t=a)$
- R_a is the reward distribution: $R_a(s, s')$ is the immediate reward for transitioning from state s to state s' with action a
- γ is the discount factor

Markov Decision Process (cont'd)

- A **policy π** is a family of distributions over actions given the states

$$\pi(a|s) = \Pr(a_t=a|s_t=s)$$

- The **return G_t** is the total discounted reward from time-step t:

$$G_t = \sum_t \gamma^t R_{at}(s_t, s_{t+1})$$

- The **value function V_π** is the *expected* return starting from a state s and following the policy π

$$V_\pi(s) = \mathrm{E}_\pi[G_t|s_t=s]$$

- The **action-value function $Q_\pi(s, a)$** is the *expected* return starting from state s, taking action a and following policy π

$$Q_\pi(s, a) = \mathrm{E}_\pi[G_t|s_t=s, a_t=a]$$

Markov Decision Process (cont'd)

- The goal is to learn a policy π which maximizes the expected cumulative reward
- This is equivalent to find the optimal action value Q* for each state:

$$Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

- In practice, a possible approach for the approximation of the action-value function is the **Q-learning algorithm**
- At each step, the action-value is updated according to the following rule

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Partially Observable Markov Decision Process

- In **partially observable Markov Decision Process** the one does not have complete knowledge of the state, that is, it does not observe directly the environment state
- A POMDP is represented by a tuple ($S, A, T, R, \Omega, O, \gamma$), where:
  - T is the set of conditional probabilities between states
  - Ω is the set of observations
  - O is the set of conditional probabilities between observations
- *Example*. Given a particular goal to reach on a two-dimensional plane, the observations may be the current position (x,y), while the MDP's states may be (x,y,x_G,y_G)

Gaussian Processes in Model-Free MDPs

- Model-Free means that the algorithm does not use the transition probability distribution and the reward distribution associated with the MDP
- A **Gaussian Process** is a stochastic process, that is, a collection of random variables, any finite subset of which has a joint Gaussian distribution
- GPs can be used as function approximators in model-free reinforcement learning applications: GP regression can be exploited to *approximate the action-value function* and obtain the optimal action-value Q^*

Gaussian Processes for Model-Free MDPs (cont'd)

Naive approach. A *single* GP is used to store the values of $Q_{approx}=Q_t$, for each action in A, and it is initialized optimistically. At each step an action a is chosen through the policy π and GP_a is updated with an input/output sample s_t, $R_t + \gamma \max_{a'} GP_{a'}.\text{predict}(s_{t+1})$.

Drawbacks of the naive approach. In the worst case scenario, this approach requires an exponential number of samples in order to learn the optimal Q^*, due to the variance reduction rate and the non-stationarity of the Q_{approx}.

Algorithm

- Initialization

Algorithm 1: Delayed GPQ (DGPQ)

Input: GP Kernel $k(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, threshold σ^2_{tol}, ϵ_1

for $a \in A$ **do**
  $\hat{Q}_a = \emptyset$
  $GP_a = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot))$
end
for each timestep t **do**
  $a_t = \arg\max_a \hat{Q}_a(s_t)$
  $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
  $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
  $\sigma_1^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol}$ **then**
    $GP_{a_t}.\text{update}(s_t, q_t)$
  end
  $\sigma_2^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2$ **and** $\hat{Q}_a(s_t) - GP_{a_t}.\text{mean}(s_t) > 2\epsilon_1$ **then**
    $\hat{Q}_a.\text{update}(s_t, GP_{a_t}.\text{mean}(s_t) + \epsilon_1)$
    $\forall a \in A, GP_a = GP.\text{init}(\mu = \hat{Q}_a, k(\cdot,\cdot))$
  end
end

Algorithm

- Initialization
- Choose actions greedily based on Q

Algorithm 1: Delayed GPQ (DGPQ)

Input: GP Kernel $k(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, threshold σ^2_{tol}, ϵ_1

for $a \in A$ **do**
  $\hat{Q}_a = \emptyset$
  $GP_a = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot))$
end
for each timestep t **do**
  $a_t = \arg\max_a \hat{Q}_a(s_t)$
  $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
  $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
  $\sigma_1^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol}$ **then**
    $GP_{a_t}.\text{update}(s_t, q_t)$
  end
  $\sigma_2^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2$ **and** $\hat{Q}_a(s_t) - GP_{a_t}.\text{mean}(s_t) > 2\epsilon_1$ **then**
    $\hat{Q}_a.\text{update}(s_t, GP_{a_t}.\text{mean}(s_t) + \epsilon_1)$
    $\forall a \in A, GP_a = GP.\text{init}(\mu = \hat{Q}_a, k(\cdot,\cdot))$
  end
end

$\hat{Q}$

- For efficient sample complexity: $\hat{Q} \geq Q^* - \epsilon$ (optimism of $\hat{Q}$)
- Difficult to do with GP, so instead we use:

$$\hat{Q}(s,a) = \min\{\min_{\langle s_i,a\rangle \in BV} \hat{\mu}_i + L_Q d((s,a),(s_i,a)), V_{MAX}\}$$

- Basis Vectors (BV) stores values from previously learned GPs.
- To predict at points not in BV , we search over BV for the point with the lowest prediction including the weighted distance bonus.
- If no point in BV is sufficiently close, we choose V_{MAX} instead.

Algorithm

- Initialization
- Choose actions greedily based on Q
- Update GP based on observed reward at next state

Algorithm 1: Delayed GPQ (DGPQ)

Input: GP Kernel $k(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, threshold σ^2_{tol}, ϵ_1

for $a \in A$ **do**
  $\hat{Q}_a = \emptyset$
  $GP_a = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot))$
end
for each timestep t **do**
  $a_t = \arg\max_a \hat{Q}_a(s_t)$
  $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
  $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
  $\sigma_1^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol}$ **then**
    $GP_{a_t}.\text{update}(s_t, q_t)$
  end
  $\sigma_2^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2$ **and** $\hat{Q}_a(s_t) - GP_{a_t}.\text{mean}(s_t) > 2\epsilon_1$ **then**
    $\hat{Q}_a.\text{update}(s_t, GP_{a_t}.\text{mean}(s_t) + \epsilon_1)$
    $\forall a \in A, GP_a = GP.\text{init}(\mu = \hat{Q}_a, k(\cdot,\cdot))$
  end
end

How it is done

- Initialize $\hat{Q}$, use it as a prior.
- Update GP using points $z_t = q_t - \hat{Q}(s_t)$ to center it on $\hat{Q}$.
- When updating $\hat{Q}$, remember to add $\hat{Q}(s_t)$ back in.

Algorithm

- Initialization
- Choose actions greedily based on Q
- Update GP based on observed reward
- Update Q if above convergence threshold

Algorithm 1: Delayed GPQ (DGPQ)

Input: GP Kernel $k(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, threshold σ^2_{tol}, ϵ_1

for $a \in A$ **do**
  $\hat{Q}_a = \emptyset$
  $GP_a = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot))$
end
for each timestep t **do**
  $a_t = \arg\max_a \hat{Q}_a(s_t)$
  $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
  $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
  $\sigma_1^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol}$ **then**
    $GP_{a_t}.\text{update}(s_t, q_t)$
  end
  $\sigma_2^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2$ **and** $\hat{Q}_a(s_t) - GP_{a_t}.\text{mean}(s_t) > 2\epsilon_1$ **then**
    $\hat{Q}_a.\text{update}(s_t, GP_{a_t}.\text{mean}(s_t) + \epsilon_1)$
    $\forall a \in A, GP_a = GP.\text{init}(\mu = \hat{Q}_a, k(\cdot,\cdot))$
  end
end

$\hat{Q}$ update

- An update of $\hat{Q}$ corresponds to adding a new element to the set BV.
- Redundant constraints are eliminated by checking if the new constraint results in a lower prediction value at other basis vector locations.
- In practice:
    - Add point $\langle (s_i, a_i), \hat{\mu}_i \rangle$ to basis vector set
    - Check if for any j: $\mu_i + L_Q d((s_i, a), (s_j, a)) \leq \mu$
    - Take all j out of the set.

Algorithm

- Initialization
- Choose actions greedily based on Q
- Update GP based on observed reward
- Update Q if above convergence threshold
- Reset GPs

Algorithm 1: Delayed GPQ (DGPQ)

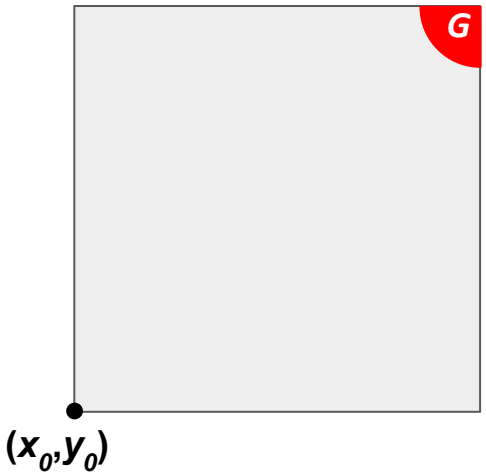
Input: GP Kernel $k(\cdot,\cdot)$, Environment Env, Actions A, initial state s_0, discount γ, threshold σ^2_{tol}, ϵ_1

for $a \in A$ **do**
  $\hat{Q}_a = \emptyset$
  $GP_a = GP.\text{init}(\mu = \frac{R_{max}}{1-\gamma}, k(\cdot,\cdot))$
end
for each timestep t **do**
  $a_t = \arg\max_a \hat{Q}_a(s_t)$
  $\langle r_t, s_{t+1} \rangle = Env.\text{takeAct}(a_t)$
  $q_t = r_t + \gamma \max_a \hat{Q}_a(s_{t+1})$
  $\sigma_1^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol}$ **then**
    $GP_{a_t}.\text{update}(s_t, q_t)$
  end
  $\sigma_2^2 = GP_{a_t}.\text{variance}(s_t)$
  if $\sigma_1^2 > \sigma^2_{tol} \geq \sigma_2^2$ **and** $\hat{Q}_a(s_t) - GP_{a_t}.\text{mean}(s_t) > 2\epsilon_1$ **then**
    $\hat{Q}_a.\text{update}(s_t, GP_{a_t}.\text{mean}(s_t) + \epsilon_1)$
    $\forall a \in A, GP_a = GP.\text{init}(\mu = \hat{Q}_a, k(\cdot,\cdot))$
  end
end

Experiment

Setting.

- Goal search in a 1×1 two-dimensional box: the goal *G* is positioned in the top right corner, while the agent starts from the bottom left corner and has to get closer than 0.15 to the goal
- MDP is defined as:
  - $\Omega = \{(x,y) \mid (x,y) \in [0,1]^2\} \Rightarrow$ observations are the positions
  - $A = \{(-0.1,0), (0.1,0), (0,-0.1), (0, 0.1)\} \Rightarrow$ actions are additive movements in the four compass directions
  - $R = 1$ if $(x,y) = G$ else 0
- Noise: movements are affected by additive Gaussian noise with $\mu = 0$, $\sigma^2 = 0.01$

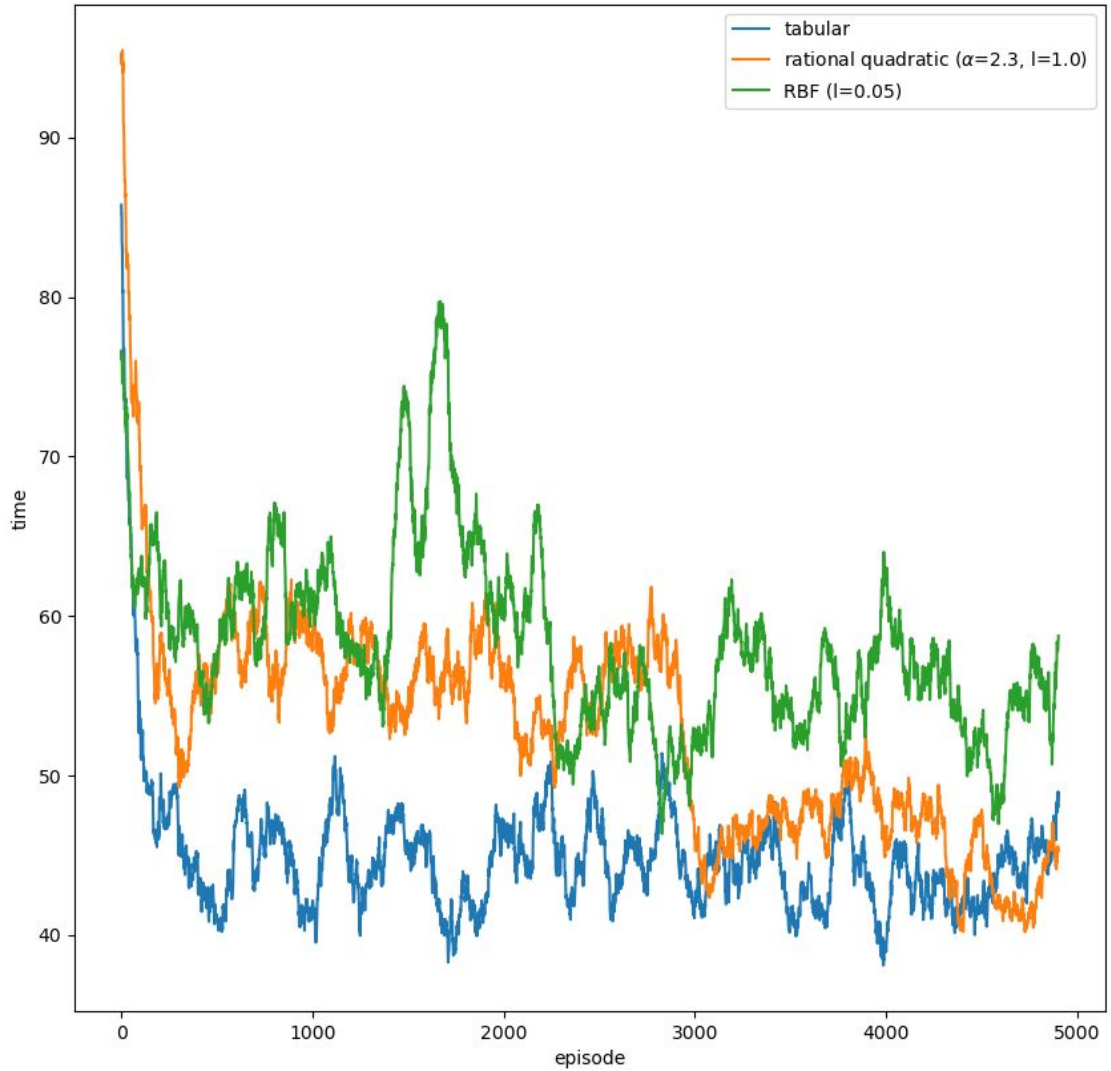


Experiment (cont'd)

- Greedy policy for GP models:

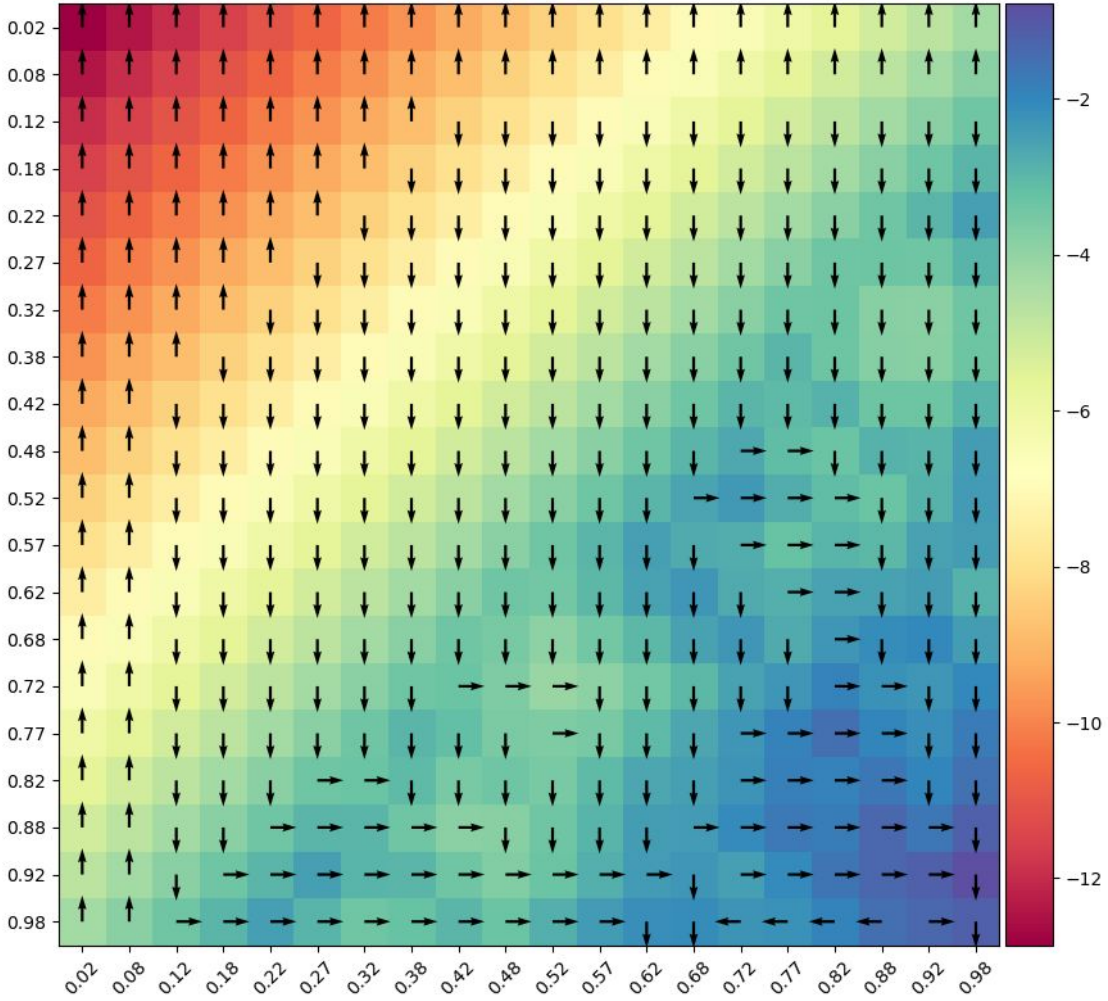
$$a_t = \text{argmax}_a \, Q_t(s_t, a)$$

- $\gamma = 0.9$ for all the models
- In this simple setting, tabular Q-learning may be the most convenient choice

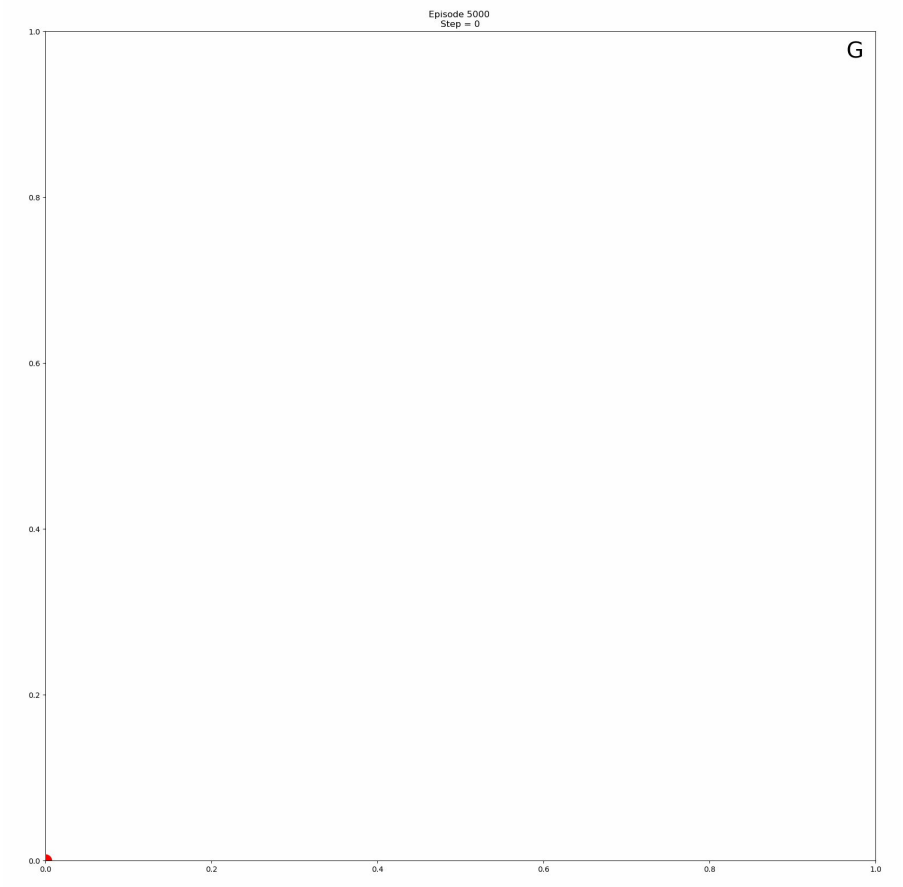


Experiment (cont'd)

- The final Q_{approx} shows higher values in the vicinity of the goal (violet)
- The arrows represent the actions corresponding to the policy extracted from Q_{approx}
- Far away from the goal, a strange behavior occurs (actions are opposite to expected ones): the value of γ may be too low



Experiment (cont’d)



References

1. Grande, Robert C., Thomas J. Walsh and Jonathan P. How. “Sample Efficient Reinforcement Learning with Gaussian Processes.” *ICML* (2014).