16-899 Assignment 3

Chi-Chian Wu
chichiaw@andrew.cmu.edu
November 22, 2020

1 Part I - Parameter Estimation

In 1.1 and 1.2, I used RLS method to estimate the parameters.

$$
\begin{aligned}
x_{k+1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} T \\ T \end{bmatrix} \begin{bmatrix} a & b \end{bmatrix} (x_G - x_k) + \begin{bmatrix} T \\ T \end{bmatrix} w_k \\
\begin{bmatrix} x^1_{k+1} \\ x^2_{k+1} \end{bmatrix} &= \begin{bmatrix} x^1_k + ax^1_G T - ax^1_k T + bx^2_G T - bx^2_k T \\ x^2_k + ax^1_G T - ax^1_k T + bx^2_G T - bx^2_k T \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix} w_k \\
&= \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix} w_k
\end{aligned}
$$

1.1

$$
\begin{aligned}
G_{k-1} &= \begin{bmatrix} \frac{\partial f_1}{\partial x^1_g} & \frac{\partial f_1}{\partial x^2_g} \\ \frac{\partial f_2}{\partial x^1_g} & \frac{\partial f_2}{\partial x^2_g} \end{bmatrix} = \begin{bmatrix} aT & bT \\ aT & bT \end{bmatrix} \\
H_k &= Gk - 1^T G_{k-1} + \lambda H_{k-1} \\
e_k &= \begin{bmatrix} x^1_{k+1} \\ x^2_{k+1} \end{bmatrix} - f_\theta(x_k) \\
\theta_k &= \theta_{k-1} + H_k^- 1 G^T_{k-1} e_k
\end{aligned}
$$

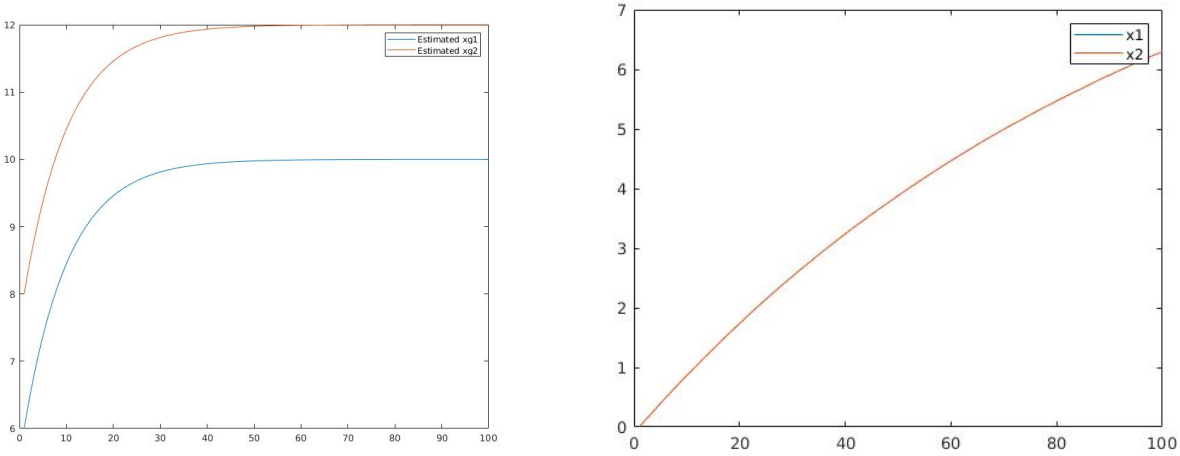

Figure 1: Unknown x_g

1.2

$$G_{k-1} = \begin{bmatrix} \frac{\partial f_1}{\partial a} & \frac{\partial f_1}{\partial b} \\ \frac{\partial f_2}{\partial a} & \frac{\partial f_2}{\partial b} \end{bmatrix} = \begin{bmatrix} (x_G^1 - x_k^1)T & (x_G^2 - x_k^2)T \\ (x_G^1 - x_k^1)T & (x_G^2 - x_k^2)T \end{bmatrix}$$
$$H_k = Gk - 1^T G_{k-1} + \lambda H_{k-1}$$
$$e_k = \begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix} - f_\theta(x_k)$$
$$\theta_k = \theta_{k-1} + H_k^- 1 G_{k-1}^T e_k$$

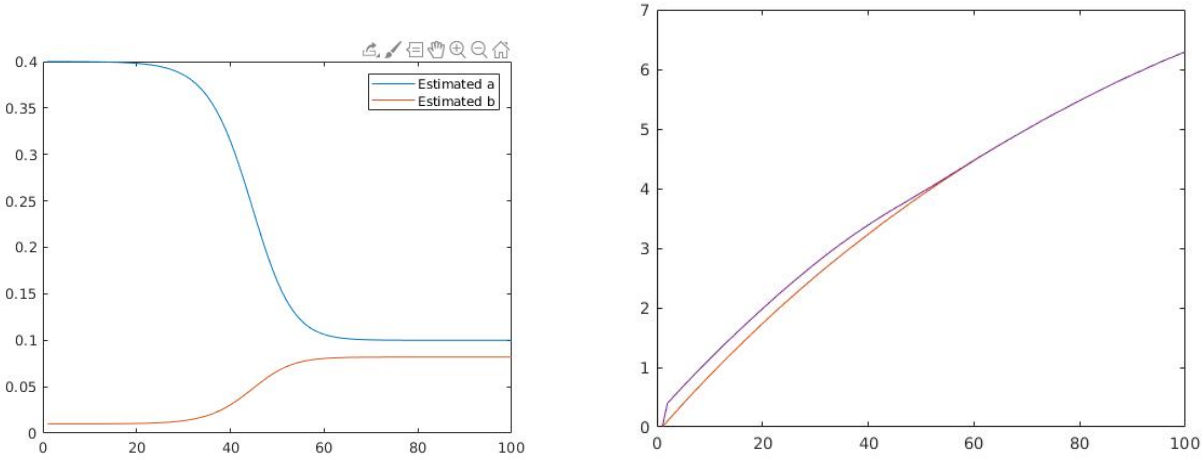

Figure 2: Unknown a, b

1.3

In 1.3 I used EKF to approximate the parameters.

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_{k-1}$$
$$M_k = AZ_{k-1}A^T + B^w W_{k-1} B^{wT}$$

Measurement update

$$Z_k = M_k - M_k C^T (CM_k C^T + V_k)^{-1} CM_k$$

Taylor expansion:

$$\bar{x_{k+1}} = \begin{bmatrix} x_k^1 + ax_G^1 T - ax_k^1 T + bx_G^2 T - bx_k^2 T \\ x_k^2 + ax_G^1 T - ax_k^1 T + bx_G^2 T - bx_k^2 T \\ a_k \\ b_k \\ xg_k^1 \\ xg_k^2 \end{bmatrix} = f(\bar{x_k}, u_k), \quad \bar{x_k} = \begin{bmatrix} x_{k1} \\ x_{k2} \\ a_k \\ b_k \\ xg_k^1 \\ xg_k^2 \end{bmatrix}$$

$$f_k(\bar{x_k}, u_k) \sim f_k(\hat{x_{k|k}}, u_k) + \frac{\partial}{\partial \bar{x}} f_k(\hat{x_{k|k}}, u_k)(\bar{x_k} - \hat{x_{k-1|k-1}})$$

$$\frac{\partial}{\partial \bar{x}} f_k(\hat{x_{k|k}}, u_k) = \begin{bmatrix} 1 - Ta & -bT & Tx_g^1 - Tx_k^1 & Tx_g^2 - Tx_k^2 & Ta & Tb \\ -Ta & 1 - bT & Tx_g^1 - Tx_k^1 & Tx_g^2 - Tx_k^2 & Ta & Tb \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \bar{A_k}$$

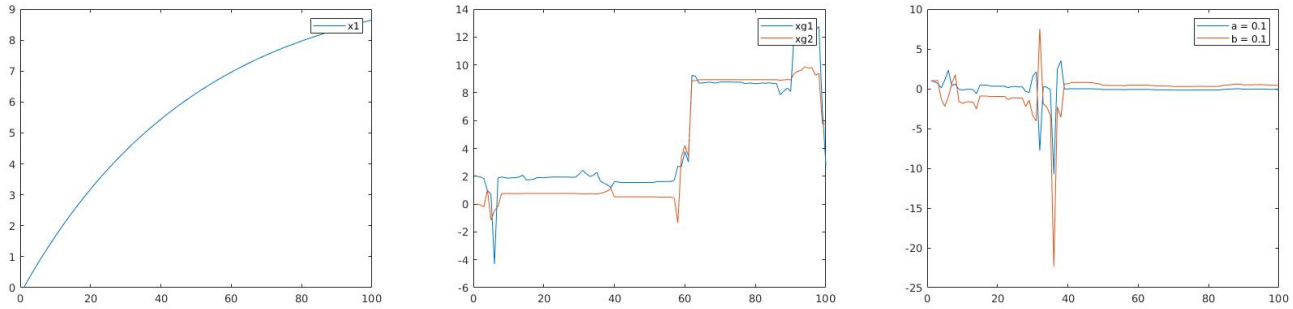

Figure 3: Unknown xg, a, b

2 Part II - Value Approximation

2.1

Here I chose a parameterized Q function in a form of $\frac{1}{2}f^T W f$. The reason is that the run-time cost is also in a quadratic form. In order to make sure the system minimize the distance between the current position and the target, also the direction is aligned, the function f is defined in the following way.

$$
\begin{aligned}
f &= \begin{bmatrix} P_x + Vcos\theta - P_{gx} \\ P_y + Vsin\theta - P_{gy} \\ V - V_g \\ \theta - \theta_g \\ u \end{bmatrix} = \begin{bmatrix} A \\ u \end{bmatrix} \\
\therefore Q_w &= \frac{1}{2} f^T W f = \frac{1}{2} f^T \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} f \\
&= \frac{1}{2}(A^T W_{11} A + A^T W_{12} u + u^T W_{21} A + u^T W_{22} u) \\
\frac{\partial Q_w}{u} &= W_{21} A + W_{22} u \\
argmin Q_w(A, u) &= -W_{22}^{-1} W_{21} A
\end{aligned}
$$

2.2

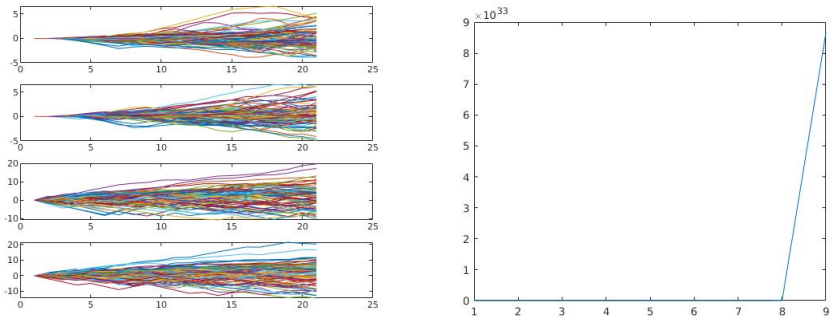

Figure 4: Monte Carlo

2.3

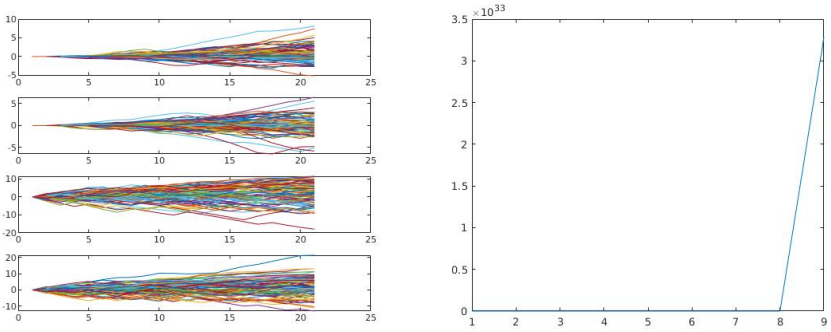

Figure 5: SARSA

2.4

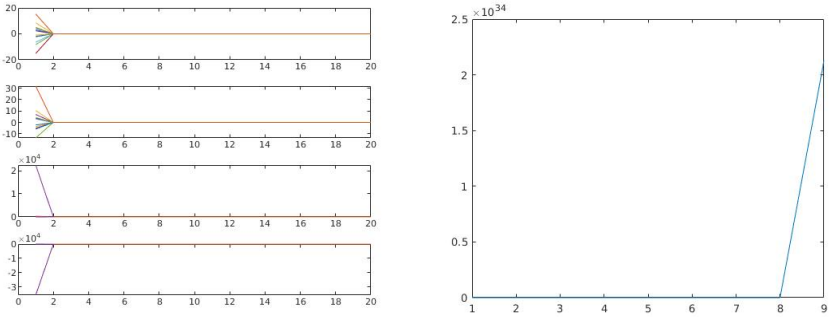

Figure 6: Q learning

2.5

Since I didn't get my value approximations to converge, I could only deduct some results from the lecture slides. Monte Carlo method tends to have higher variance at the beginning of the episodes. Since Q- learning is directly learning the optimal policy, it tends to have a higher mean. However, it is also subject to higher variance and might suffer from problems to converging as a result.

3 Part III - Policy Gradient

3.1

I was thinking using Gaussian method to do policy gradient. Feature function are the same as the previous problem.

$$\pi_\theta(u|x) = \frac{1}{\sigma_\theta\sqrt{2\pi}} exp(-\frac{(u-m_\theta(x))^2}{2\sigma_\theta(x)^2})$$
$$\sigma_\theta = exp(\theta_1^T f1)$$
$$m_\theta = \theta_2^T f_2$$
$$\nabla_\theta ln\pi_\theta(u|x) = \nabla_\theta[-ln\sigma_\theta(x) - \frac{(u-m_\theta(x))^2}{2\sigma_\theta(x)^2}]$$

3.2

3.3

3.4

Since I havn't been able to get a convergence results in the previous problem, I read the following paper as a reference to answer the question. (" J. Beitelspacher, J. Fager, G. Henriques, and A. McGovern, "Policy gradient vs. value function approximation: A reinforcement learning shootout," School of Computer Science, University of Oklahoma, Tech. Rep, 2006"). In general, using value approximation is relatively easier, since we do not have to tell the system how to achieve the goal. Additionally, the rewards setup in the policy gradient method will require significant prior knowledge. Or else the system will be set up in a way that is only good at collecting rewards but deviate from the primary task of reaching the goal.