[CSIC30046] Reinforcement Learning

Homework 1

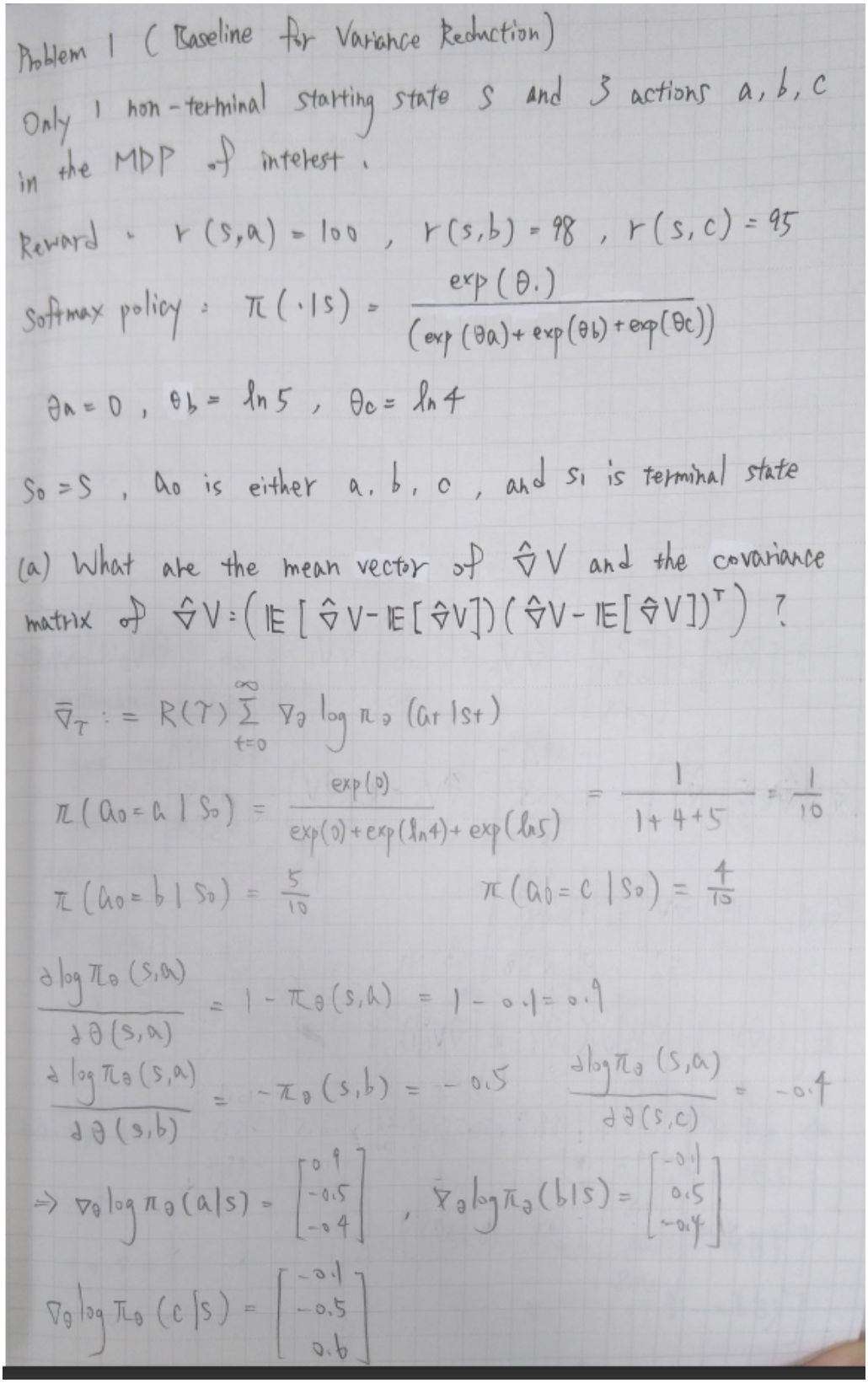
Po-Chuan, Chen
Student ID: 311511052
present90308.ee11@nycu.edu.tw

National Yang Ming Chiao Tung University
March 30, 2023

Contents

1 Problem 1 (Baseline for Variance Reduction)

(a)



Problem 1 (Baseline for Variance Reduction)

Only 1 non-terminal starting state S and 3 actions a, b, c in the MDP of interest.

Reward: $r(s,a) = 100$, $r(s,b) = 98$, $r(s,c) = 95$

Softmax policy: $\pi(\cdot|s) = \frac{\exp(\theta_\cdot)}{(\exp(\theta_a) + \exp(\theta_b) + \exp(\theta_c))}$

$\theta_a = 0$, $\theta_b = \ln 5$, $\theta_c = \ln 4$

$S_0 = S$, a_0 is either a, b, c, and s_1 is terminal state

(a) What are the mean vector of $\hat{\nabla} V$ and the covariance matrix of $\hat{\nabla} V = (\mathbb{E}[\hat{\nabla} V - \mathbb{E}[\hat{\nabla} V]](\hat{\nabla} V - \mathbb{E}[\hat{\nabla} V])^T)$?

$\bar{\nabla}_T := R(\tau) \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)$

$\pi(a_0 = a | s_0) = \frac{\exp(0)}{\exp(0) + \exp(\ln 4) + \exp(\ln 5)} = \frac{1}{1+4+5} = \frac{1}{10}$

$\pi(a_0 = b | s_0) = \frac{5}{10}$ $\quad$ $\pi(a_0 = c | s_0) = \frac{4}{10}$

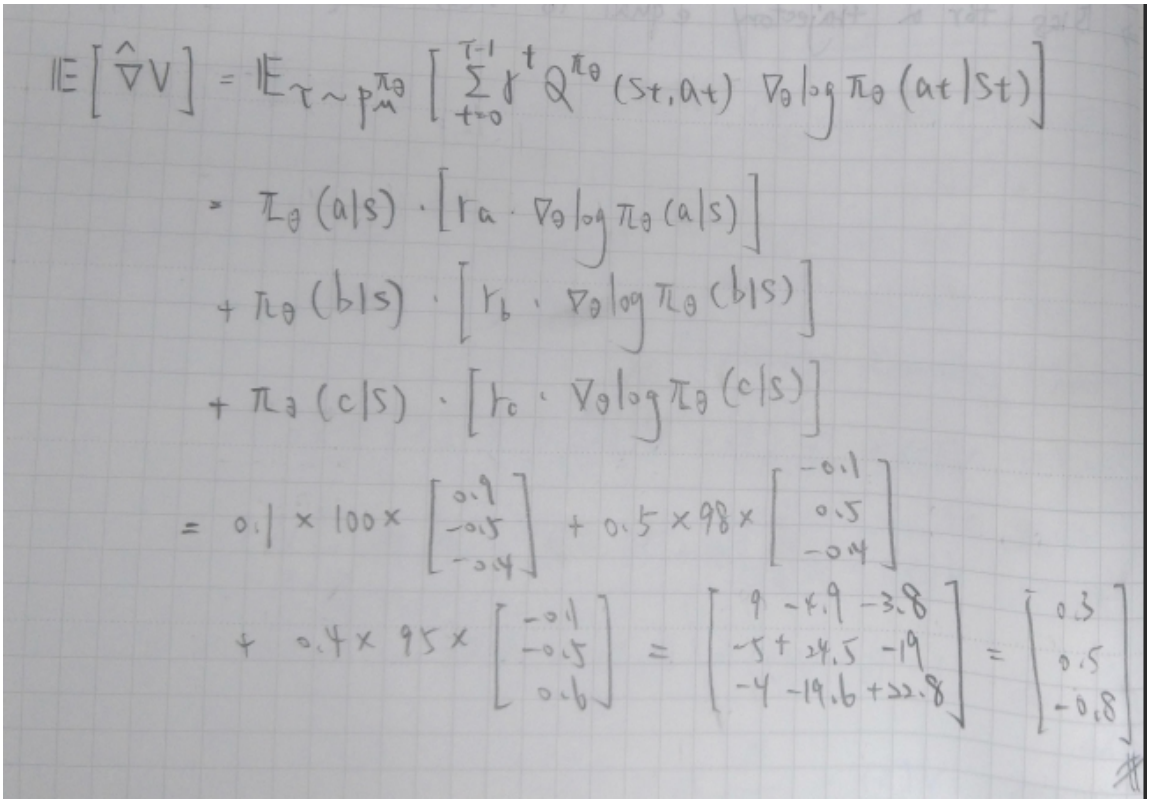
$\frac{\partial \log \pi_\theta(s,a)}{\partial \theta(s,a)} = 1 - \pi_\theta(s,a) = 1 - 0.1 = 0.9$

$\frac{\partial \log \pi_\theta(s,a)}{\partial \theta(s,b)} = -\pi_\theta(s,b) = -0.5$ $\quad$ $\frac{\partial \log \pi_\theta(s,a)}{\partial \theta(s,c)} = -0.4$

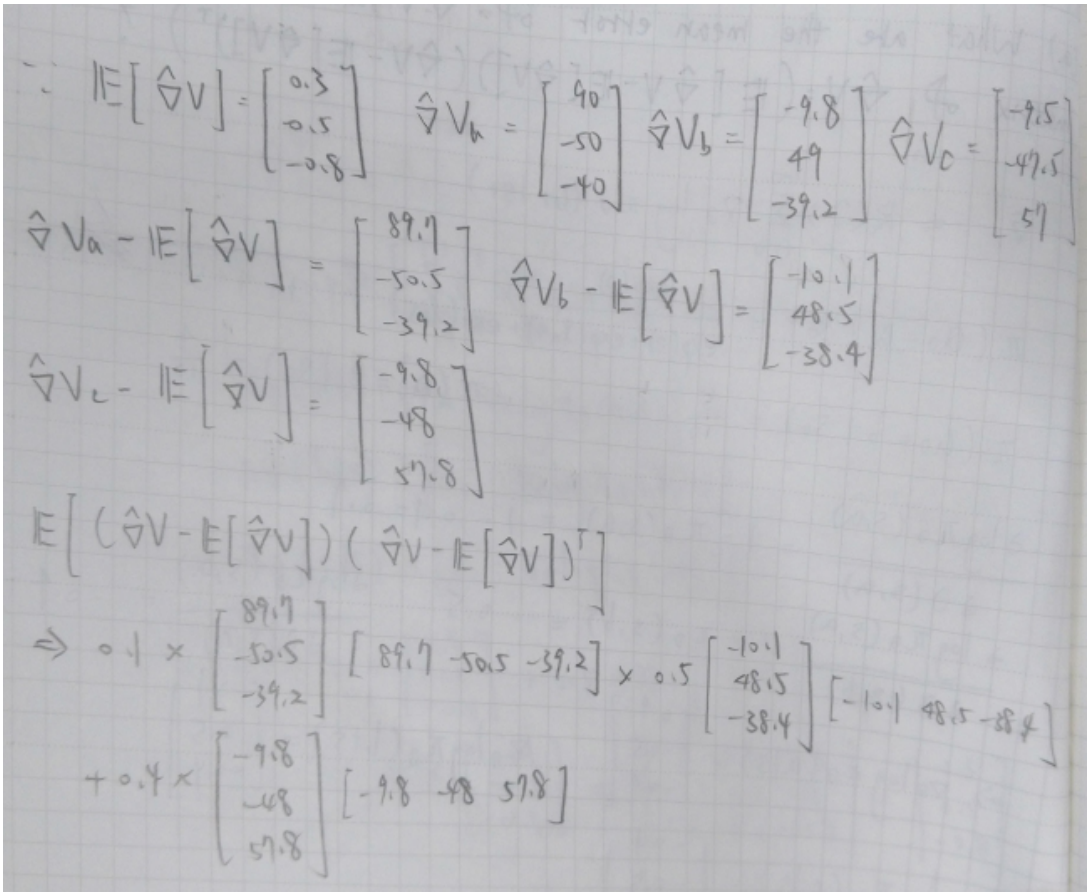
$\Rightarrow \nabla_\theta \log \pi_\theta(a|s) = \begin{bmatrix} 0.9 \\ -0.5 \\ -0.4 \end{bmatrix}$, $\nabla_\theta \log \pi_\theta(b|s) = \begin{bmatrix} -0.1 \\ 0.5 \\ -0.4 \end{bmatrix}$

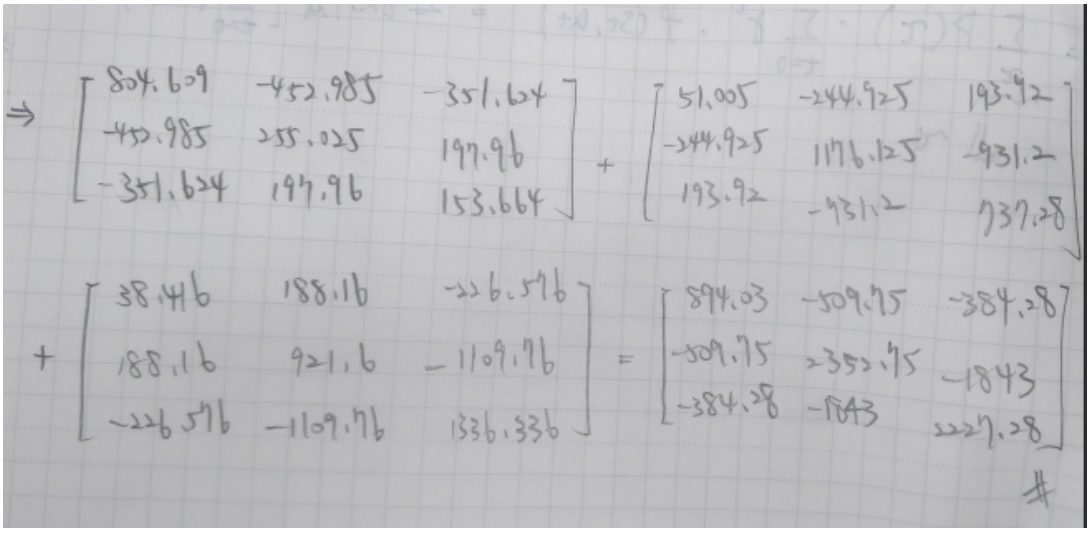
$\nabla_\theta \log \pi_\theta(c|s) = \begin{bmatrix} -0.1 \\ -0.5 \\ 0.6 \end{bmatrix}$

Mean vector



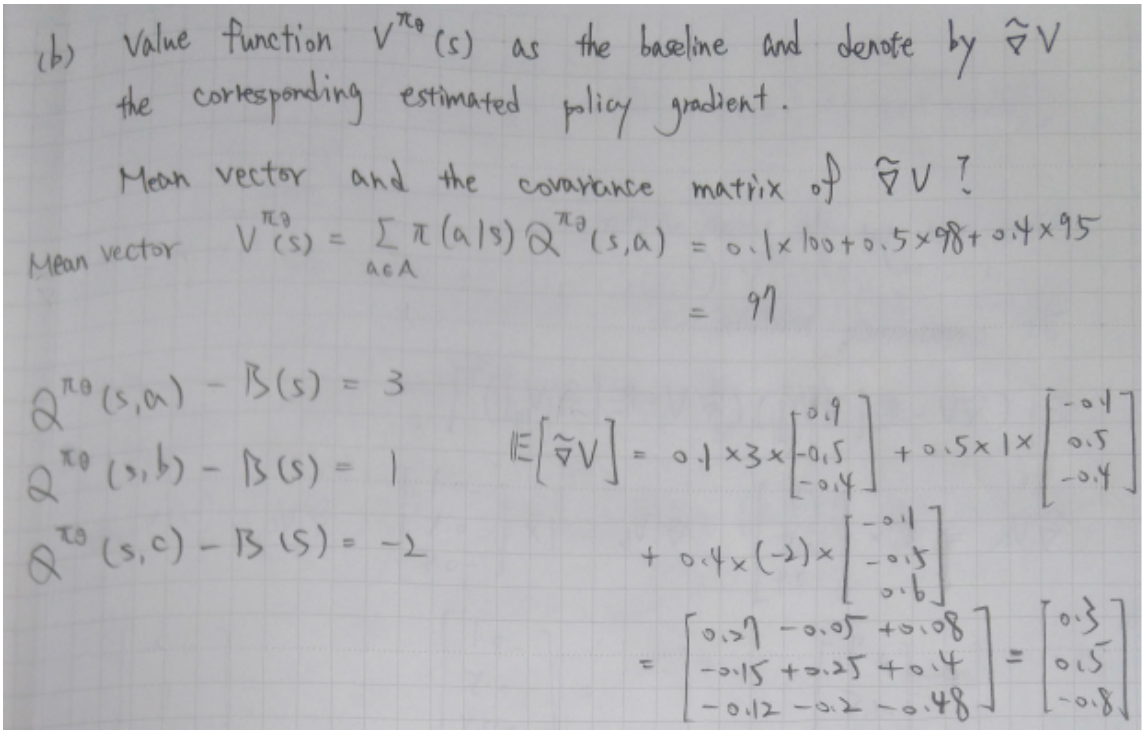
Covariance matrix



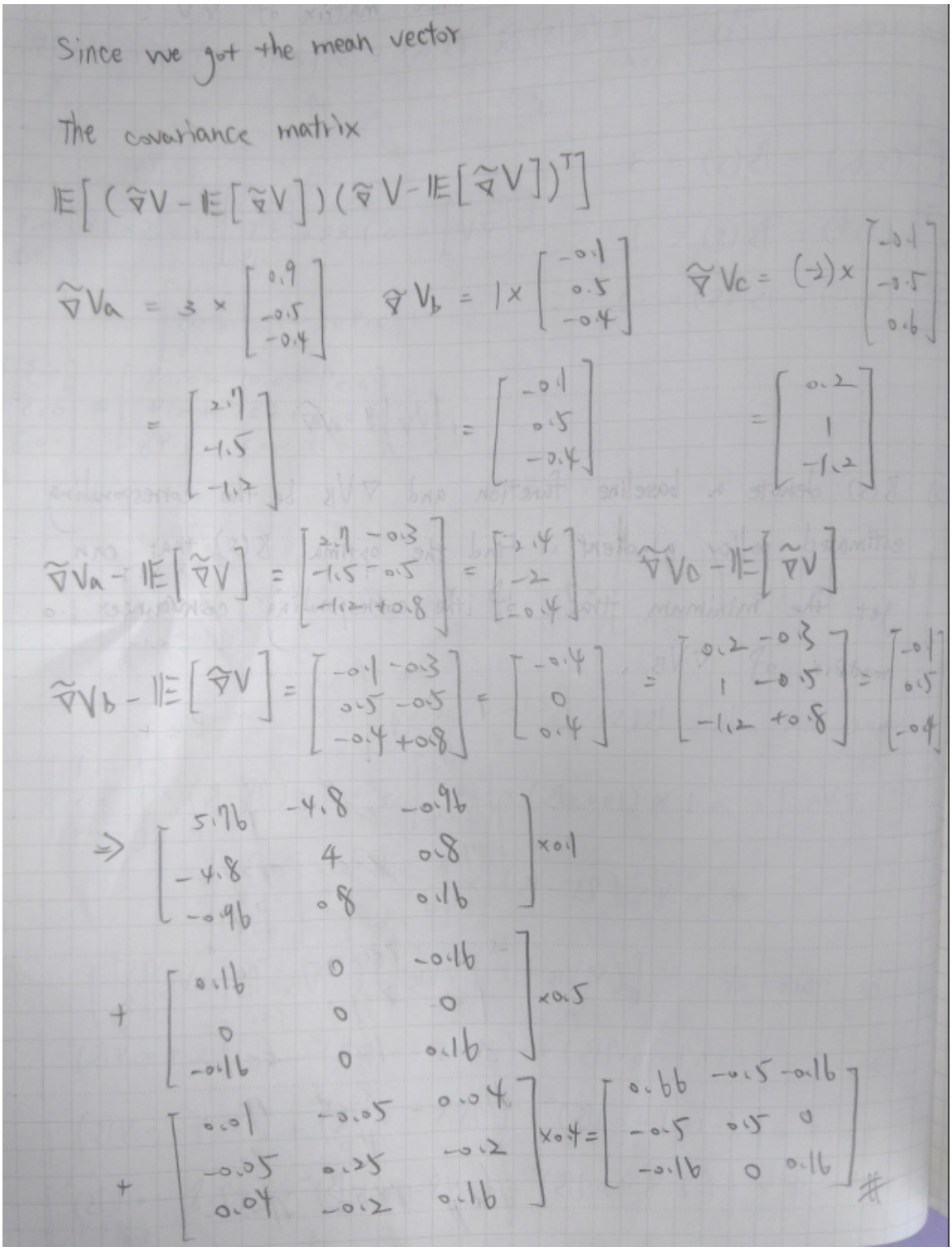


(b)

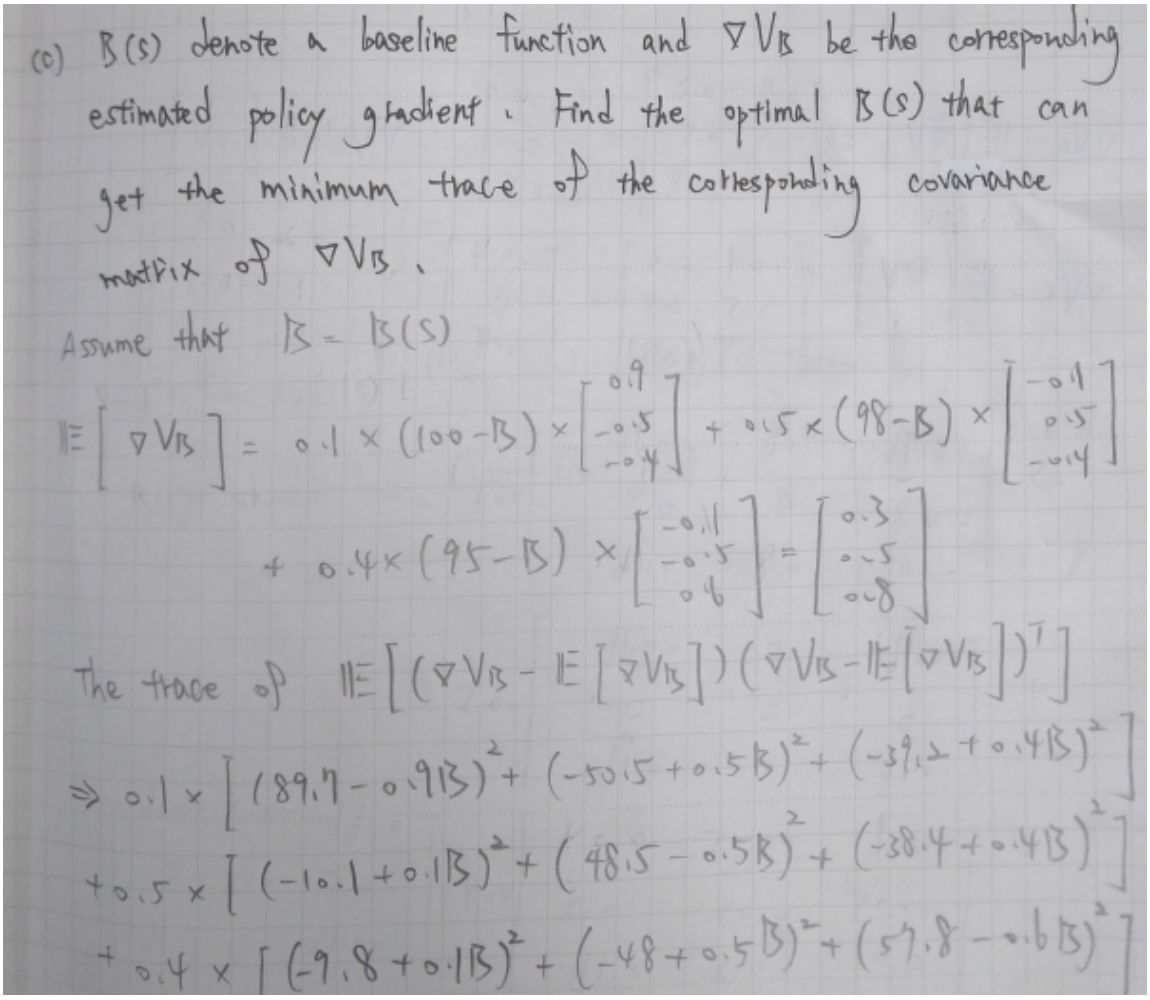
Mean vector



Covariance matrix



(c)



(c) $B(s)$ denote a baseline function and ∇V_B be the corresponding estimated policy gradient. Find the optimal $B(s)$ that can get the minimum trace of the corresponding covariance matrix of ∇V_B.

Assume that $B = B(s)$

$$\mathbb{E}[\nabla V_B] = 0.1 \times (100 - B) \times \begin{bmatrix} 0.9 \\ -0.5 \\ -0.4 \end{bmatrix} + 0.5 \times (98 - B) \times \begin{bmatrix} -0.1 \\ 0.5 \\ -0.4 \end{bmatrix}$$

$$+ 0.4 \times (95 - B) \times \begin{bmatrix} -0.1 \\ -0.5 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 0.3 \\ -0.5 \\ -0.8 \end{bmatrix}$$

The trace of $\mathbb{E}[(\nabla V_B - \mathbb{E}[\nabla V_B])(\nabla V_B - \mathbb{E}[\nabla V_B])^T]$

$$\Rightarrow 0.1 \times [(89.7 - 0.9B)^2 + (-50.5 + 0.5B)^2 + (-39.2 + 0.4B)^2]$$

$$+ 0.5 \times [(-10.1 + 0.1B)^2 + (48.5 - 0.5B)^2 + (-38.4 + 0.4B)^2]$$

$$+ 0.4 \times [(-9.8 + 0.1B)^2 + (-48 + 0.5B)^2 + (57.8 - 0.6B)^2]$$

$$\frac{dT}{dB} = 0.1 \times 2 \times \left[(-0.9) \times (89.7 - 0.9B) + (0.5) \times (-50.5 + 0.5B) + (0.4)(-39.2 + 0.4B) \right]$$

$$+ 0.5 \times 2 \times \left[(0.1) \times (-10.1 + 0.1B) + (-0.5)(48.5 - 0.5B) + (0.4)(-38.4 + 0.4B) \right]$$

$$+ 0.4 \times 2 \times \left[(0.1) \times (-4.8 + 0.1B) + (0.5)(-48 + 0.5B) + (-0.6)(57.8 - 0.6B) \right]$$

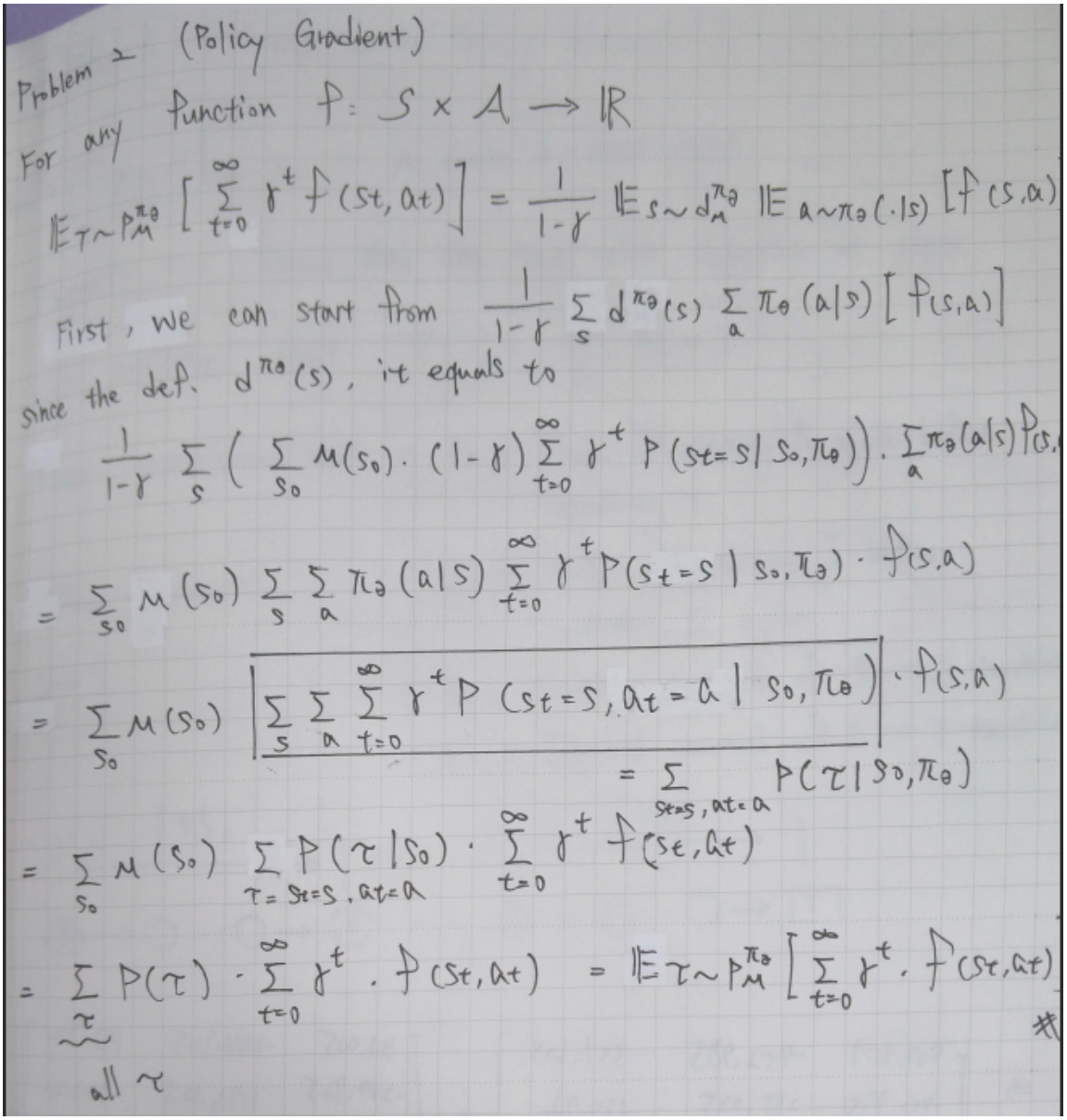
$$= 2 \times \left[0.1 \times (-121.66 + 1.22B) + 0.5 \times (-40.62 + 0.42B) + 0.4 \times (-59.66 + 0.62B) \right]$$

$$= 2 \times \left[-56.34 + 0.58B \right]$$

If the $\frac{dT}{dB} = 0$, we can find the best B

$$B = \frac{56.34}{0.58} = 97.13793034 \approx 97.14 \#$$

2 Problem 2 (Policy Gradient)



Problem 2 (Policy Gradient)

For any function $f: S \times A \to \mathbb{R}$

$$\mathbb{E}_{\tau \sim P_M^{\pi_\theta}}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t)\right] = \frac{1}{1-\gamma}\mathbb{E}_{s \sim d_M^{\pi_\theta}}\mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[f(s,a)]$$

First, we can start from $\frac{1}{1-\gamma}\sum_s d^{\pi_\theta}(s)\sum_a \pi_\theta(a|s)[f(s,a)]$

since the def. $d^{\pi_\theta}(s)$, it equals to

$$\frac{1}{1-\gamma}\sum_s\left(\sum_{s_0} M(s_0)\cdot(1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t=s \mid s_0, \pi_\theta)\right)\cdot\sum_a \pi_\theta(a|s) f(s,a)$$

$$= \sum_{s_0} M(s_0)\sum_s\sum_a \pi_\theta(a|s)\sum_{t=0}^{\infty}\gamma^t P(s_t=s \mid s_0, \pi_\theta)\cdot f(s,a)$$

$$= \sum_{s_0} M(s_0)\left|\sum_s\sum_a\sum_{t=0}^{\infty}\gamma^t P(s_t=s, a_t=a \mid s_0, \pi_\theta)\right|\cdot f(s,a)$$

$$= \sum_{s_t=s,\, a_t=a} P(\tau \mid s_0, \pi_\theta)$$

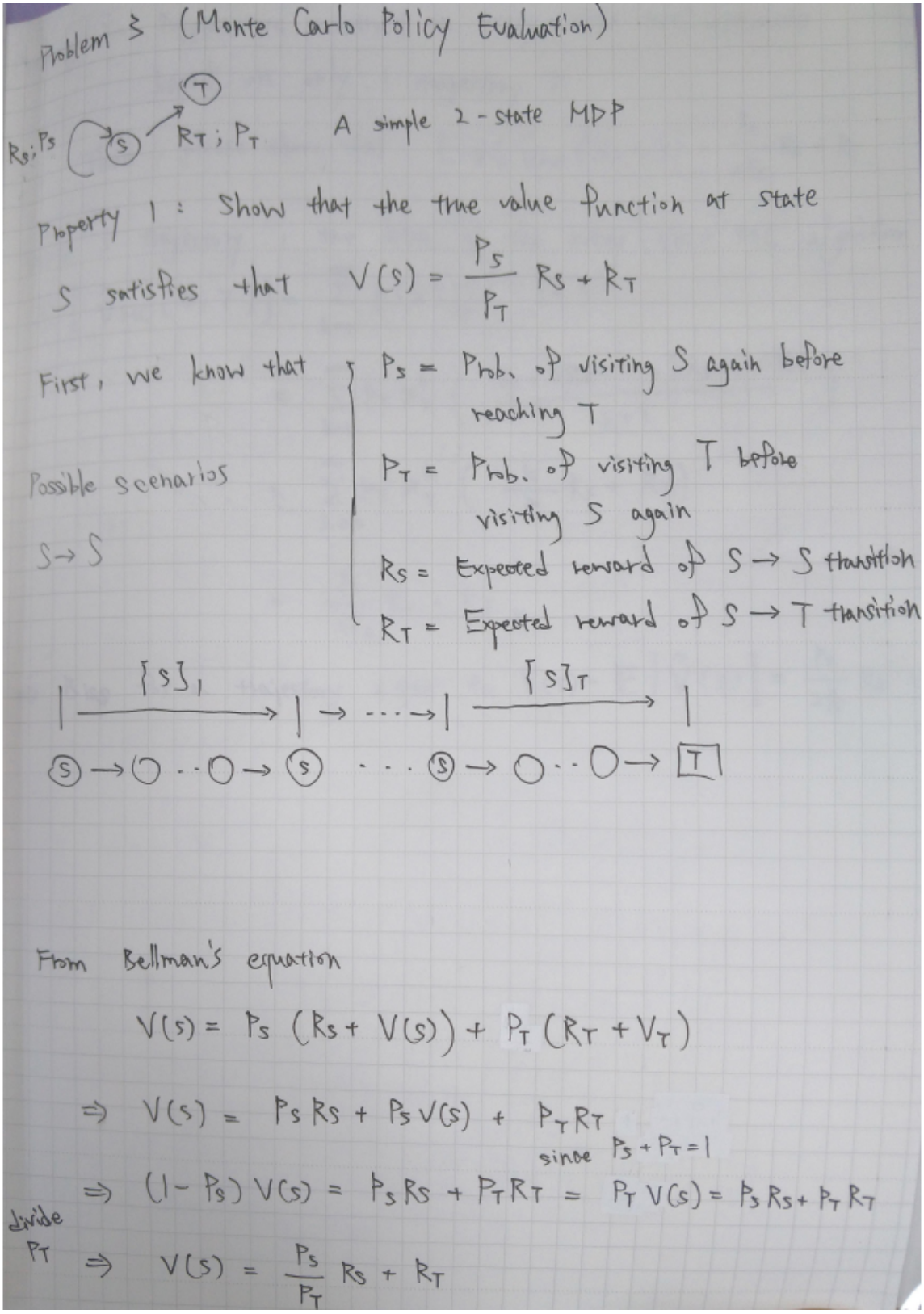
$$= \sum_{s_0} M(s_0)\sum_{\tau=s_t=s,\, a_t=a} P(\tau|s_0)\cdot\sum_{t=0}^{\infty}\gamma^t f(s_t,a_t)$$

$$= \sum_{\tau} P(\tau)\cdot\sum_{t=0}^{\infty}\gamma^t\cdot f(s_t,a_t) = \mathbb{E}_{\tau\sim P_M^{\pi_\theta}}\left[\sum_{t=0}^{\infty}\gamma^t\cdot f(s_t,a_t)\right]$$

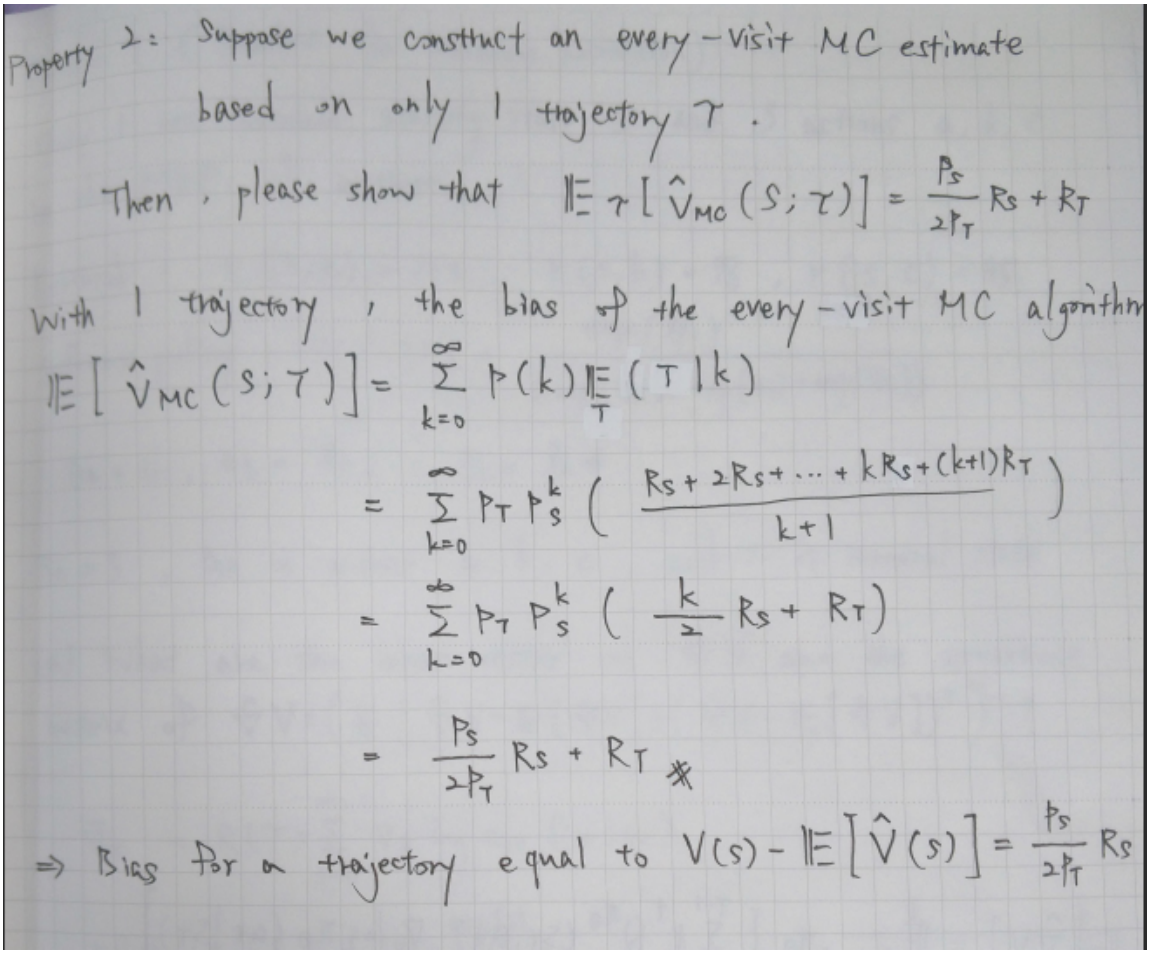
all τ #

3 Problem 3 (Monte Carlo Policy Evaluation)

Property 1



Property 2


Property 2: Suppose we construct an every-visit MC estimate based on only 1 trajectory τ.
Then, please show that $\mathbb{E}_\tau[\hat{V}_{MC}(S;\tau)] = \frac{P_S}{2P_T} R_S + R_T$
With 1 trajectory, the bias of the every-visit MC algorithm
$\mathbb{E}[\hat{V}_{MC}(S;\tau)] = \sum_{k=0}^{\infty} P(k) \mathbb{E}_T(T|k)$
$= \sum_{k=0}^{\infty} P_T P_S^k \left(\frac{R_S + 2R_S + \dots + kR_S + (k+1)R_T}{k+1} \right)$
$= \sum_{k=0}^{\infty} P_T P_S^k \left(\frac{k}{2} R_S + R_T \right)$
$= \frac{P_S}{2P_T} R_S + R_T$ *
⇒ Bias for a trajectory equal to $V(S) - \mathbb{E}[\hat{V}(S)] = \frac{P_S}{2P_T} R_S$

4 Problem 4 (Policy Gradient Algorithms With Function Approximation)

(a)

Hyperparameters

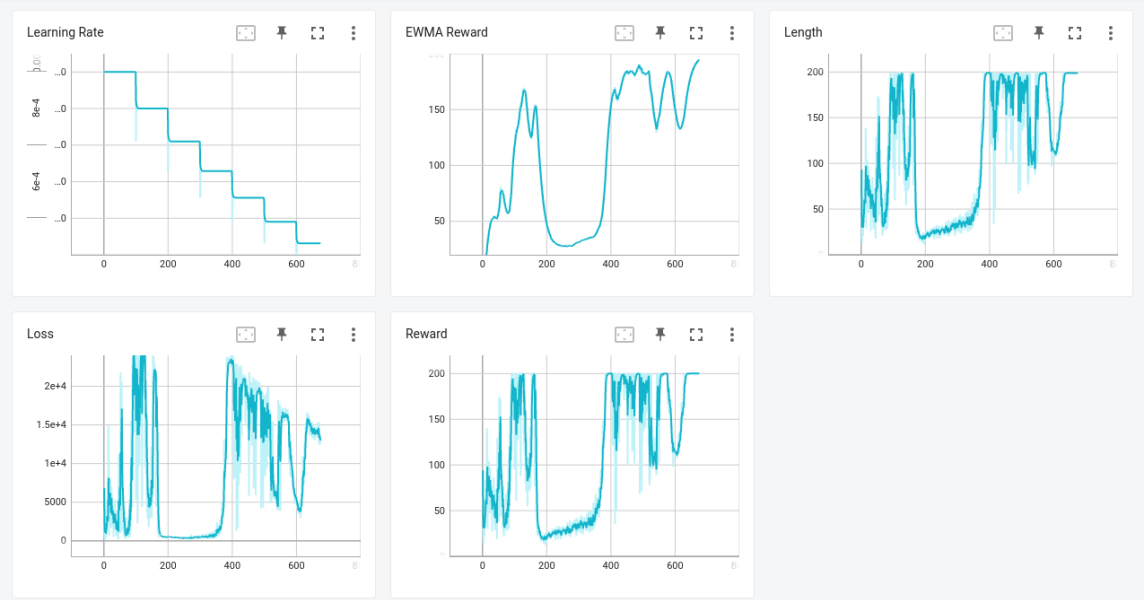
- Learning rate: 0.001
- Gamma: 0.999
- Scheduler step size: 100
- Scheduler gamma: 0.9
- Optimizer: `Adam`

```
self.shared_first = nn.Linear(self.observation_dim, self.hidden_size)
# initialize the shared layer
nn.init.kaiming_normal_(self.shared_first.weight)
# random weight initialization
self.shared_second = nn.Linear(self.hidden_size, self.hidden_size)
nn.init.kaiming_normal_(self.shared_second.weight)
self.action_head = nn.Linear(self.hidden_size, self.action_dim)
# initialize the action layer
nn.init.kaiming_normal_(self.action_head.weight)
self.value_head = nn.Linear(self.hidden_size, 1)
# initialize the value layer, since we only need one value, we set the
    output dimension to 1
nn.init.kaiming_normal_(self.value_head.weight)
```

Listing 1: weight initalize

```
def forward(self, state):
    """
        Forward pass of both policy and value networks
        - The input is the state, and the outputs are the corresponding
          action probability distirbution and the state value
        TODO:
            1. Implement the forward pass for both the action and the
    state value
    """

    ########## YOUR CODE HERE (3~5 lines) ##########
    input = F.relu(self.shared_first(state))
    input = F.relu(self.shared_second(input))
    action_prob = F.softmax(self.action_head(input), dim=-1)
    # softmax over the last dimension
    state_value = self.value_head(input)
    # no activation function for the value layer
    ########## END OF YOUR CODE ##########

    return action_prob, state_value
```

Listing 2: Network architecture

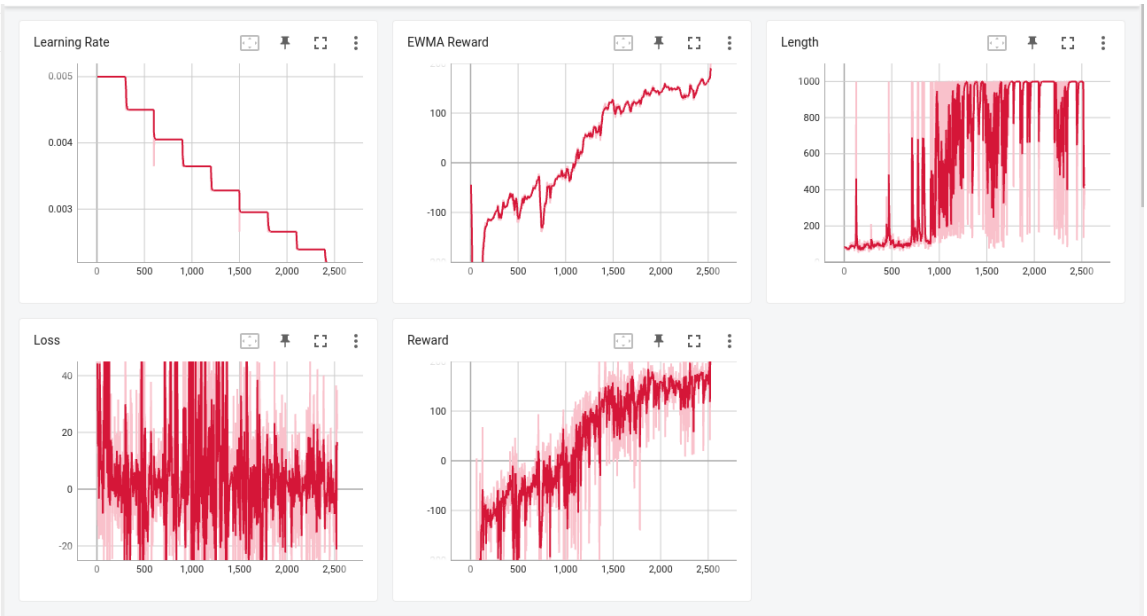


(b)

Hyperparameters

- Learning rate: 0.005
- Gamma: 0.99
- Scheduler step size: 300
- Scheduler gamma: 0.9
- Optimizer: `Adam`

Same network architecture as **(a)**. But the accumulating reward will add divided by 200.



(c)

Hyperparameters

- Learning rate: 0.005
- Gamma: 0.999
- Scheduler step size: 300
- Scheduler gamma: 0.9
- Optimizer: `Adam`

Same network architecture as **(a)**.

```python
class GAE:
    def __init__(self, gamma, lambda_, num_steps):
        self.gamma = gamma
        self.lambda_ = lambda_
        self.num_steps = num_steps
        # set num_steps = None to adapt full batch

    def __call__(self, rewards, values, done):

    #Implement Generalized Advantage Estimation (GAE) for your value prediction
    #TODO (1): Pass correct corresponding inputs (rewards, values, and done) into the function arguments
    #TODO (2): Calculate the Generalized Advantage Estimation and return the obtained value


        ########## YOUR CODE HERE (8-15 lines) ##########
        # Initialize the lists and variables
        advantages = []
        advantage = 0
        next_value = 0

        for reward, value, done in zip(reversed(rewards), reversed(values), reversed(done)):
            td_error = reward + self.gamma * next_value * (1 - done) - value
            # calculate the temporal difference error
            advantage = td_error + self.gamma * self.lambda_ * advantage * (1 - done)
            # calculate the advantage
            next_value = value # update the next value
            advantages.insert(0, advantage)
            # insert the advantage to the front of the list

        advantages = torch.tensor(advantages)
        # convert the list to a tensor
        advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-5) # normalize the advantages

        return advantages
```

Listing 3: Generalized Advantage Estimation (GAE)

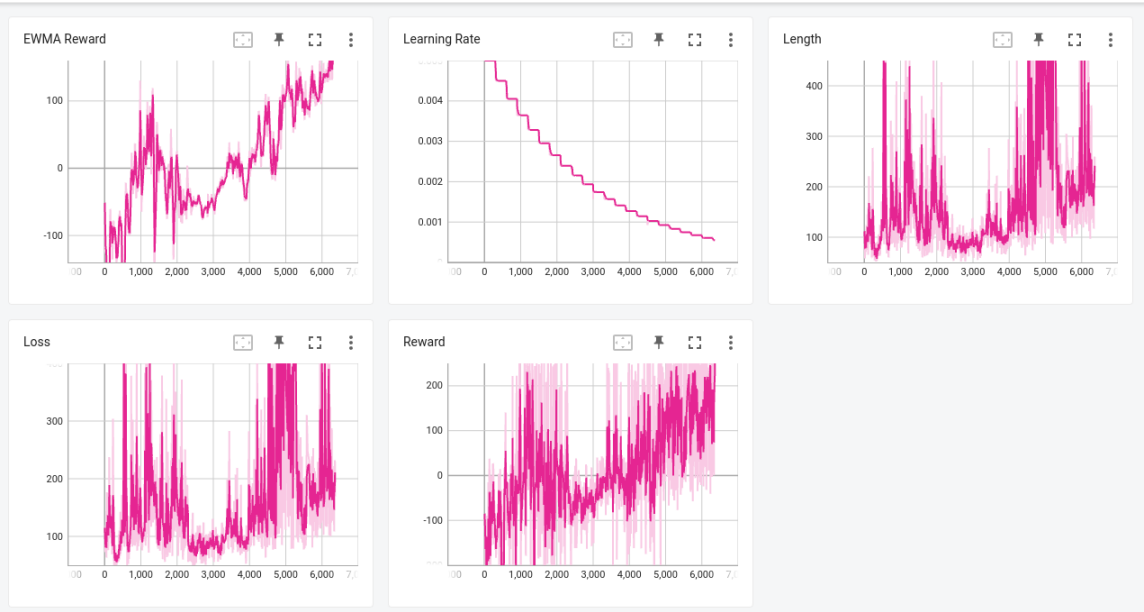
```python
def calculate_loss(self, gamma=0.999, advantages=None):
    """
        Calculate the loss (= policy loss + value loss) to perform backprop later
        TODO:
            1. Calculate rewards-to-go required by REINFORCE with the help of self.rewards
            2. Calculate the policy loss using the policy gradient
            3. Calculate the value loss using either MSE loss or smooth L1 loss
    """

    # Initialize the lists and variables

    R = 0
    saved_actions = self.saved_actions
    policy_losses = []
    value_losses = []

    vals = []

    for a, v in saved_actions:
        vals.append(v[0])
    ########## YOUR CODE HERE (8-15 lines) ##########
    for (log_prob, val), advantage in zip(saved_actions, advantages(self.rewards, vals, self.done)):
        policy_losses.append(-log_prob * advantage.detach())
        value_losses.append(advantage.pow(2))

    loss = torch.stack(policy_losses).sum() + torch.stack(value_losses).sum()
    ########## END OF YOUR CODE ##########

    return loss
```

Listing 4: Loss for GAE

I used the algorithm about REINFORCE with GAE to implement the code
Initalize θ_0 and step size η
Sampling trajectory $\tau \sim P_{\mu}^{\pi_\theta}$ and make the update as

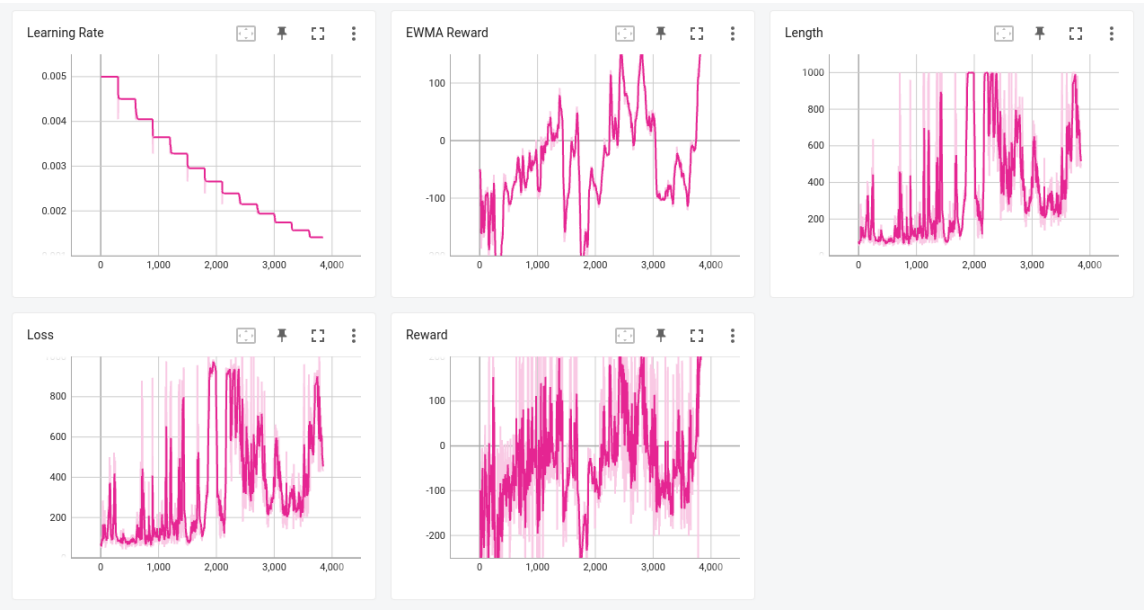
$$\theta_{k+1} = \theta_k + \eta(\sum_{t=0}^{\infty} \gamma^t \hat{A}_t^{GAE(\gamma,\lambda)} \nabla_\theta \log \pi_\theta(a_t \mid s_t))$$

where $\hat{A}_t^{GAE(\gamma,\lambda)}$ is constructed from $V(s)$ with TD return



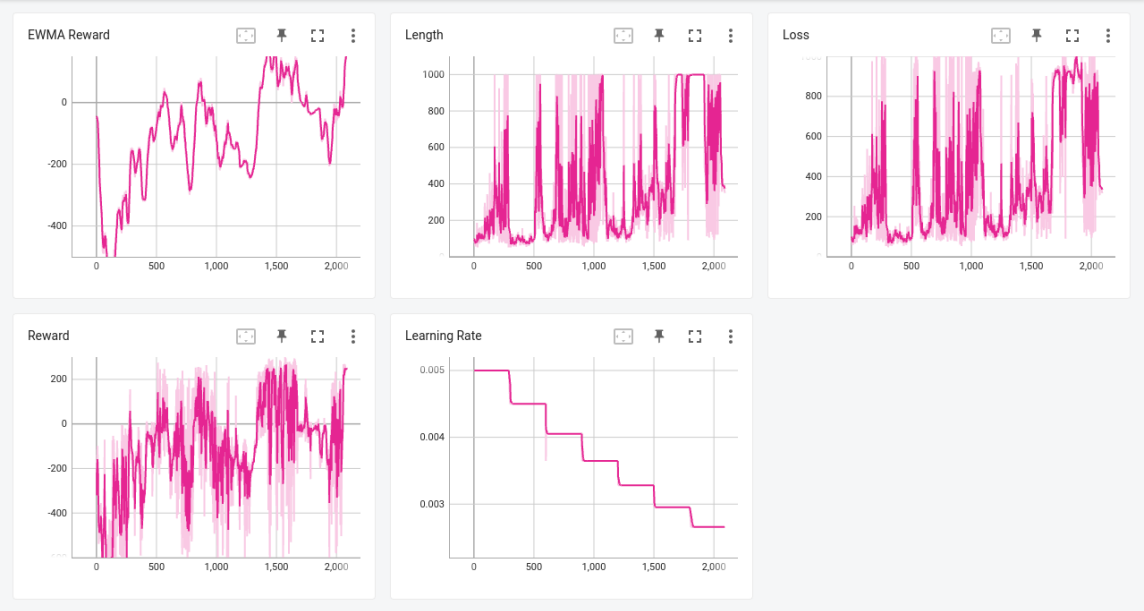
$\lambda = 0.99$

The model learns the task after around 6000 episodes when $\lambda = 0.99$.



$\lambda = 0.98$

The model learns the task after around 4000 episodes when $\lambda = 0.98$.



$\lambda = 0.97$

The model learns the task after around 2000 episodes when $\lambda = 0.97$.

First, we can look at the formula about GAE estimator:

$$\hat{A}^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda(\hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \dots) = \sum_{\ell=0}^{\infty} (\gamma\lambda)^\ell \delta_{t+\ell}$$

As we can see, when we using larger λ, the convergence range will get larger.

When the $\lambda = 1$, we will use MC return, and when $\lambda = 0$ we will use TD return.