CMPSCI 687 Homework 3 - Fall 2022

Due **October 27, 2022**, 11:55pm Eastern Time

1 Instructions

This homework assignment consists of a written portion and a programming portion. While you may discuss problems with your peers (e.g., to discuss high-level approaches), you must answer the questions on your own. In your submission, do explicitly list all students with whom you discussed this assignment. Submissions must be typed (handwritten and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code. You **must** use Python 3 for your homework code. You may not use any reinforcement learning or machine learning-specific libraries in your code, e.g., TensorFlow, PyTorch, or scikit-learn. You *may* use libraries like numpy and matplotlib, though. The automated system will not accept assignments after 11:55pm on October 27. The tex file for this homework can be found here.

Before starting this homework, please review this course's policies on plagiarism by reading Section 10 of the syllabus.

Part One: Written (50 Points Total)

1. (**4 Points**) An operator f is a contraction mapping if there exists some $\lambda \in [0,1)$ such that $d\big(f(x), f(y)\big) \leq \lambda d(x,y)$, where d is a distance metric. Notice, here, that we explicitly forbid that $\lambda = 1$, otherwise we would have no guarantees that the distance between $f(x)$ and $f(y)$ is smaller than the distance between x and y. Assume, alternatively, that we try to "fix" this problem by defining f to be a contraction mapping if there exists some $\lambda \in [0,1]$ such that $d\big(f(x), f(y)\big) < \lambda d(x,y)$. Notice that we now allow $\lambda = 1$, but we require $d\big(f(x), f(y)\big)$ to be strictly less than $\lambda d(x,y)$. What is the problem with this alternative definition of contraction mapping? What could go wrong if we were to use it?

   Solution 1. If $\lambda \in [0,1]$ such that $d(f(x), f(y)) < d(x,y)$. Assume a case where $\lambda = 0$, then the following will have to be true

   $$d(f(x), f(y)) < 0$$

   since $d()$ is a distance metric, it cannot be a negative term. Therefore this condition of **distance being strictly less** contradicts the basic definition of distance metrics.

2. (**8 Points**) Let M be an MDP with finite state space, finite action space, and $\gamma < 1$. Let π_1^* be an optimal policy that is *stochastic*. Assume that we only know π_1^*, and that we *do not know* $v^{\pi_1^*}$ and $q^{\pi_1^*}$. Describe a procedure for transforming this (arbitrary) stochastic optimal policy, π_1^*, into a *deterministic* policy, π_2^*, that is also optimal. Prove that the procedure you proposed guarantees that π_2^* is indeed optimal. Hint: you can start from the definition of the Bellman Equation for v^π (for an arbitrary policy π) and by recalling that $\sum_{x\in\mathcal{X}} \Pr(X = x) f(x) \leq \max f(x)$, where X is a random variable with outcomes in $\mathcal{X}$ and f is a function.

   Solution 2. For any π, without loss of generality we can write the value function for any state s as follows. Assume that there are 3 actions

   $$v^\pi(s) = \sum_{a\in\{a_1,a_2,a_3\}} \pi(s,a) q^\pi(s,a)$$

   $$v^\pi(s) = \pi(s,a_1)q^\pi(s,a_1) + \pi(s,a_2)q^\pi(s,a_2) + \pi(s,a_3)q^\pi(s,a_3)$$

   Similarly for π^*

   $$v^{\pi*}(s) = \pi^*(s,a_1)q^{\pi*}(s,a_1) + \pi^*(s,a_2)q^{\pi*}(s,a_2) + \pi^*(s,a_3)q^{\pi*}(s,a_3)$$

Now, if this is an optimal policy the value function will have the maximum value. Consider the case when $q^{\pi *}()$ has a different value for each action. In that case if $\pi *$ is optimal then it would be possible to assign more weightage to the highest $q^{\pi *}()$. If that is possible then the current policy is not optimal. But it is given that the policy is optimal, it means that all $q^{\pi *}()$ need to be equal. If all $q^{\pi *}()$ are equal, it means that the average return of a state on taking any action is equal. If all of those actions are giving the same return then we can assign the entire prbability mass to a single action, design the and without loss of generality we can have

$$\pi_2^*(s) = a_1$$

$$v^{\pi *}(s) = q^{\pi *}(s, a_1)$$

Now to prove that this policy is optimal, we use the Bellman equation,

$$v^{\pi_2^*}(s) = \sum_{a \in \{a_1, a_2, a_3\}} \pi(s,a) q^{\pi}(s,a)$$

Using the following property,

$$\sum_{x \in \mathcal{X}} \Pr(X = x) f(x) \leq \max f(x)$$

$$\begin{aligned} v^{\pi}(s) &= \sum_{a \in \{a_1, a_2, a_3\}} \pi(s,a) q^{\pi}(s,a) \\ &\leq \max_{a_1, a_2, a_3} q^{\pi}(s,a) \end{aligned}$$

But we have

$$q^{\pi}(s, a_1) = q^{\pi}(s, a_2) = q^{\pi}(s, a_3)$$

Therefore,

$$\begin{aligned} v^{\pi}(s) &\leq \max_{a_1, a_2, a_3} q^{\pi}(s,a) \\ &\leq q^{\pi}(s, a_1) \end{aligned}$$

But we showed above that

$$v^{\pi_2^*}(s) = q^{\pi_2^*}(s, a_1)$$

Therefore, this policy is the optimal policy.

3. (**5 Points**) In class, one of the definitions of optimal policy that we introduced is the following: π^* is an optimal policy if and only if $\pi^* \geq \pi$, for all $\pi \in \Pi$. Here, the operator $\geq$ is defined so that $\pi \geq \pi'$ iff $v^{\pi}(s) \geq v^{\pi'}(s)$, for $\pi' \in \Pi$ and for all $s \in \mathcal{S}$. Suppose π^* is the optimal policy for some MDP, e.g., Mountain Car. In Mountain Car, the initial state distribution, d_0, is defined such that the initial state of the car is $S_0 = (X_0, 0)$, where X_0 is the x coordinate of the car at time $t = 0$, drawn uniformly at random from the interval $[-0.6, -0.4]$. Suppose we *change* the initial locations where the car may be initialized—i.e., suppose we change the initial state distribution, d_0, of Mountain Car. Consider, for instance, an alternative formulation of this MDP in which the car's initial x coordinate, X_0, is drawn from a Gaussian distribution $\mathcal{N}(-0.3, 0.1)$. Without assuming anything about the new initial state distribution, is it possible to say that (in general) the optimal policy for the alternative formulation of Mountain Car is different than the optimal policy for the original problem? If so, prove that this is the case. If not, argue formally why the corresponding optimal policies may not differ.

Solution. π^* is an optimal policy if and only if $\pi^* \geq \pi$ iff $v^{\pi *}(s) \geq v^{\pi'}(s)$ for $\pi' \in \Pi$ and for all $s \in \mathcal{S}$. The above condition should be true for all $s \in \mathcal{S}$, this means that for any new initial state as well this condition will hold true and π^* will be an optimal policy. In the current question we are only changing the initial position. Since a new start position s' is from the global set $\mathcal{S}$,

$$v^{\pi *}(s') \geq v^{\pi'}(s')$$

for all $\pi' \in \Pi$. And if the above condition is true then, π^* is an optimal policy.

4. (**18 Points**) In class, we proved that the Bellman Operator is a contraction mapping and used this to show that Value Iteration converges to a unique fixed point. In the last homework, we investigated a function, $w^\pi(s,a,s')$, which represents the expected return if the agent starts in state s, executes action a, transitions to a particular next state s', and then follows policy π. In this problem you will prove that a new form of dynamic programming Policy Evaluation operator, defined over w^π, exists and is also a contraction mapping. This way, you will be showing that this Policy Evaluation procedure/operator can be used to compute w^π (for any policy π) and that it converges to a unique fixed point.

- In this question, the following identities/properties might be helpful:

  ⋆ **Property 1**:

$$\Big|\sum_{x\in\mathcal{X}} \Pr(X=x|Y=y,Z=z,\dots)f(x,y,z,\dots)\Big| \;\le\; \sum_{x\in\mathcal{X}} \Pr(X=x|Y=y,Z=z,\dots)\Big|f(x,y,z,\dots)\Big|, \quad (1)$$

  where $X, Y, Z, \dots$, are random variables with possible outcomes (respectively) in $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \dots$, and where f is a function.

  ⋆ **Property 2**:

$$\sum_{x\in\mathcal{X}} \Pr(X=x|Y=y,Z=z,\dots)f(x,y,z,\dots) \;\le\; \max_{x\in\mathcal{X}} f(x,y,z,\dots), \quad (2)$$

  where $X, Y, Z, \dots$, are random variables with possible outcomes (respectively) in $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \dots$, and where f is a function.

- Finally, let the Max-Norm over w^π be defined as $||w^\pi||_\infty = \max_s \max_a \max_{s'} |w^\pi(s,a,s')|$.

(Question 4a. 4 Points) In the last assignment, you showed how v^π can be written/expressed in terms of w^π. Your expression for v^π used only w^π, $\mathcal{S}, \mathcal{A}, p, R, d_0, \gamma$, and π. Based on this previous result, write a Bellman Equation for w^π using only $w^\pi, \mathcal{S}, \mathcal{A}, p, R, d_0, \gamma$, and π (but *not* v^π or q^π). Your Bellman Equation for w^π should resemble the form of the first Bellman Equation for v^π we studied in class: $v^\pi(s) = \sum_a \pi(s,a) \sum_{s'} p(s,a,s')\big(R(s,a) + \gamma v^\pi(s')\big)$. In particular, your Bellman Equation for w^π should have w^π on both sides of the equation, but it should not depend on q^π or v^π. Show your derivation of this new Bellman Equation step by step.

Solution 4a. From the previous HW we have,

$$v^\pi(s) = \sum_a \pi(s,a)\sum_{s'} p(s,a,s')w^\pi(s,a,s')$$

Also we have the following Bellman Eqns for $v^\pi(s)$,

$$v^\pi(s) = \sum_a \pi(s,a)\sum_{s'} p(s,a,s')\big(R(s,a) + \gamma v^\pi(s')\big)$$
$$v^\pi(s') = \sum_a \pi(s',a)\sum_{s''} p(s',a,s'')\big(R(s',a) + \gamma v^\pi(s'')\big)$$

Comparing the first 2 equations we can deduce the following,

$$w^\pi(s,a,s') = \big(R(s,a) + \gamma v^\pi(s')\big)$$

Substituting $v^\pi(s')$ from above,

$$w^\pi(s,a,s') = \Big(R(s,a) + \gamma \sum_{a'} \pi(s',a')\sum_{s''} p(s',a',s'')\big(R(s',a') + \gamma v^\pi(s'')\big)\Big)$$

Now

$$w^\pi(s',a,s'') = \big(R(s',a) + \gamma v^\pi(s'')\big)$$

so we have,

$$w^\pi(s,a,s') = R(s,a) + \gamma \sum_{a'} \pi(s',a')\sum_{s''} p(s',a',s'')\, w^\pi(s',a',s'')$$

Taking $\sum_{a'} \pi(s',a') \sum_{s''} p(s',a',s'')$ out since there is no dependence on s', a', s'',

$$w^\pi(s,a,s') = \sum_{a'} \pi(s',a') \sum_{s''} p(s',a',s'') \big(R(s,a) \,+\, \gamma\, w^\pi(s',a',s'')\big)$$

(Question 4b. 14 Points) Recall that we showed, in class, how the Bellman Optimality Equation for v^*,

$$v^*(s) = \max_a \sum_{s'} p(s,a,s') \left(R(s,a) \,+\, \gamma v^*(s')\right),$$

could be turned into an *update equation*, used by the Value Iteration algorithm, to iteratively estimate v^*:

$$v_{i+1}(s) := \max_a \sum_{s'} p(s,a,s') \left(R(s,a) \,+\, \gamma v_i(s')\right).$$

To show that this update equation converges to v^*, we showed that the operator that implements it, $\mathcal{T}(v_i) := \max_a \sum_{s'} p(s,a,s') \left(R(s,a) \,+\, \gamma v_i(s')\right)$, is a contraction mapping. Similarly, consider now the Bellman Equation for w^π that you derived in the previous question. Turn it into an update equation that takes as input a current estimate, w_i, of w^π, and returns an updated estimate, w_{i+1}. Let $\Upsilon(w_i)$ be the operator that implements this update equation; that is, $\Upsilon(w_i) := w_{i+1}$. Prove, step-by-step, that Υ is a contraction mapping under the L^∞ norm (i.e., the same Max-Norm used in our proof that the Bellman Operator is a contraction). You may find it helpful to use the two properties described at the beginning of this question.

Solution 4b. Similar to the value iteration we can show the Bellman operator for w^π,

$$w^\pi(s,a,s') = \sum_{a'} \pi(s',a') \sum_{s''} p(s',a',s'') \big(R(s,a) \,+\, \gamma\, w^\pi(s',a',s'')\big)$$

Policy update step,

$$\pi(s') = arg \max_{a'} \sum_{s''} p(s',a',s'') \big(R(s,a) \,+\, \gamma\, w^\pi(s',a',s'')\big)$$

Therefore,

$$w^\pi(s,a,s') = \max_{a'} \sum_{s''} p(s',a',s'') \big(R(s,a) \,+\, \gamma\, w^\pi(s',a',s'')\big)$$

Now we need to show that this Bellman operator is a contractions mapping for $\gamma < 1$

$$d(w,w') := \max_s \max_a \max_{s'} |w(s,a,s') - w'(s,a,s')|$$

For Brevity we will write

$$\max_s \max_a \max_{s'} = \max_{s,a,s'}$$

$$||w - w'||_\infty = \max_{s,a,s'} |w(s,a,s') - w'(s,a,s')|$$

By Definition of max norm,

$$\begin{aligned} ||\Upsilon w - \Upsilon w'||_\infty &= \max_{s,a,s'} |\Upsilon w(s,a,s') - \Upsilon w'(s,a,s')| \\ &= \max_{s,a,s'} |\max_{a'} \sum_{s''} p(s',a',s'') \big(R(s,a) \,+\, \gamma\, w(s',a',s'')\big) - \\ &\qquad \max_{a'} \sum_{s''} p(s',a',s'') \big(R(s,a) \,+\, \gamma\, w'(s',a',s'')\big)| \end{aligned}$$

Using the following property,

$$|max\ f(x) - max\ g(x)| \le max|f(x) - g(x)|$$

$$\begin{aligned}
||\Upsilon w - \Upsilon w'||_\infty &= max_{s,a,s'} | \max_{a'} \sum_{s''} p(s',a',s'')\big(R(s,a) \,+\, \gamma\, w(s',a',s'')\big) - \\
&\quad \max_{a'} \sum_{s''} p(s',a',s'')\big(R(s,a) \,+\, \gamma\, w'(s',a',s'')\big)| \\
&\leq \max_{s,a,s'} \max_{a'} | \sum_{s''} p(s',a',s'')\big(R(s,a) \,+\, \gamma\, w(s',a',s'')\big) - \\
&\quad \sum_{s''} p(s',a',s'')\big(R(s,a) \,+\, \gamma\, w'(s',a',s'')\big)|
\end{aligned}$$

$p(s',a',s'')R(s,a)$ cancels out factoring γ,

$$||\Upsilon w - \Upsilon w'||_\infty = \gamma \max_{s,a,s'} \max_{a'} | \sum_{s''} p(s',a',s'')\big(w(s',a',s'') - w'(s',a',s'')\big)|$$

Using Property 1,

$$||\Upsilon w - \Upsilon w'||_\infty \leq \gamma \max_{s,a,s'} \max_{a'} \sum_{s''} p(s',a',s'')|w(s',a',s'') - w'(s',a',s'')|$$

Using Property 2,

$$\begin{aligned}
||\Upsilon w - \Upsilon w'||_\infty &\leq \gamma \max_{s,a,s'} \max_{a'} \max_{s''} |w(s',a',s'') - w'(s',a',s'')| \\
&= \gamma \max_{s} \max_{a} \max_{s'} \max_{a'} \max_{s''} |w(s',a',s'') - w'(s',a',s'')|
\end{aligned}$$

Removing $\max_s \max_a$ because s and a don't appear,

$$\begin{aligned}
||\Upsilon w - \Upsilon w'||_\infty &\leq \gamma \max_{s'} \max_{a'} \max_{s''} |w(s',a',s'') - w'(s',a',s'')| \\
&= \gamma ||w - w'||_\infty
\end{aligned}$$

Therefore we proved that,

$$||\Upsilon w - \Upsilon w'||_\infty \leq \gamma ||w - w'||_\infty$$

By definition of Max norm,

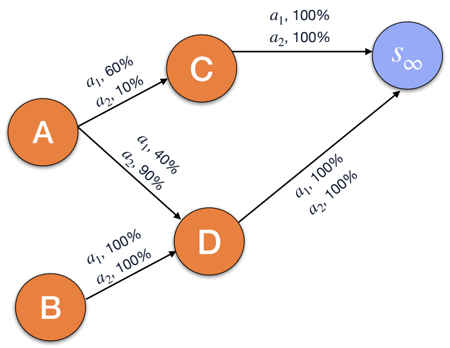
$$d\big(\Upsilon(w) - \Upsilon(w')\big) \leq \gamma d\big(w, w'\big)$$

Hence it is a contraction mapping.

5. (**15 Points**) We showed in class that the Policy Iteration algorithm can—in the worst case— converge to π^* only in the limit, since the "standard" Policy Evaluation step may require an infinite number of iterations to converge. In this question, you will show that a *different* way of performing the Policy Evaluation step can be used and allow for Policy Iteration to converge in a *finite, bounded*, number of iterations. First, recall that in slides 55–58 of Lecture 06, we discussed how to perform policy evaluation by creating a system of linear equations with $|\mathcal{S}|$ equations and $|\mathcal{S}|$ unknowns. In particular, this system has one equation per state: the Bellman Equation for the corresponding state. As an example, consider an MDP with 2 states and 3 actions; the resulting system of equations would be:

$$\begin{aligned}
v^\pi(s_1) &= \sum_{a\in\{a_1,a_2,a_3\}} \pi(s_1,a) \sum_{s'\in\{s_1,s_2\}} p(s_1,a,s')\left(R(s_1,a) + \gamma v^\pi(s')\right)\\
v^\pi(s_2) &= \sum_{a\in\{a_1,a_2,a_3\}} \pi(s_2,a) \sum_{s'\in\{s_1,s_2\}} p(s_2,a,s')\left(R(s_2,a) + \gamma v^\pi(s')\right)\\
v^\pi(s_3) &= \sum_{a\in\{a_1,a_2,a_3\}} \pi(s_3,a) \sum_{s'\in\{s_1,s_2\}} p(s_3,a,s')\left(R(s_3,a) + \gamma v^\pi(s')\right).
\end{aligned}$$

If p and R are known, one could expand each equation above and obtain formulas whose form resembles (for instance) something like $v^\pi(s_1) = 0.3 + 0.12\gamma v^\pi(s_1) + 0.42\gamma v^\pi(s_2) - 0.9\gamma v^\pi(s_3)$. Consider the following MDP:



Assume the following transition function, p, where all transition probabilities not indicated in the table below are 0; the following reward function, R; and the following stochastic policy, π:

$p(A,a_1,C)=0.6$	$p(A,a_1,D)=0.4$
$p(A,a_2,C)=0.1$	$p(A,a_2,D)=0.9$
$p(B,a_1,D)=1.0$	$p(B,a_2,D)=1.0$
$p(C,a_1,s_\infty)=1.0$	$p(C,a_2,s_\infty)=1.0$
$p(D,a_1,s_\infty)=1.0$	$p(D,a_2,s_\infty)=1.0$

$R(A,a_1)=7$	$R(A,a_2)=4$
$R(B,a_1)=0$	$R(B,a_2)=1$
$R(C,a_1)=9$	$R(C,a_2)=-4$
$R(D,a_1)=-2$	$R(D,a_2)=6$

$\pi(A,a_1)=0.3$	$\pi(A,a_2)=0.7$
$\pi(B,a_1)=0.4$	$\pi(B,a_2)=0.6$
$\pi(C,a_1)=0.1$	$\pi(C,a_2)=0.9$
$\pi(D,a_1)=0.25$	$\pi(D,a_2)=0.75$

(Question 5a. 5 Points) Start from the Bellman Equation for v^π and use the probabilities and rewards specified above to show, step by step, how to construct a system of equations that, when solved, produces the values of $v^\pi(A), v^\pi(B), v^\pi(C)$, and $v^\pi(D)$.

Solution 5a. Applying Bellman equation

$$v^\pi(s) = \sum_a \pi(s,a)\sum_{s'} p(s,a,s')\left(R(s,a) + \gamma v^\pi(s')\right)$$

For state A we can substitute the transition probabilities, the policy probabilities and the reward as follows,

$$\begin{aligned}
v^\pi(A) &= \sum_{a\in\{a_1,a_2\}} \pi(A,a) \sum_{s'\in\{C,D\}} p(A,a,s')\left(R(A,a) + \gamma v^\pi(s')\right)\\
&= 0.3*0.6*(7+\gamma v^\pi(C)) + 0.3*0.4*(7+\gamma v^\pi(D)) +\\
&\quad 0.7*0.1*(4+\gamma v^\pi(C)) + 0.7*0.9*(4+\gamma v^\pi(D))\\
&= 4.9 + 0.25\gamma v^\pi(C) + 0.75 v^\pi(D)
\end{aligned}$$

For B,

$$\begin{aligned}
v^\pi(B) &= \sum_{a\in\{a_1,a_2\}} \pi(B,a) \sum_{s'\in\{D\}} p(B,a,s')\big(R(B,a) \,+\, \gamma v^\pi(s')\big)\\
&= 0.4 * 1.0 * (0 + \gamma v^\pi(D)) + 0.6 * 1.0 * (1 + \gamma v^\pi(D))\\
&= 0.6 + \gamma v^\pi(D)
\end{aligned}$$

For C, (by definition $v^\pi(s_\infty) = 0$)

$$\begin{aligned}
v^\pi(C) &= \sum_{a\in\{a_1,a_2\}} \pi(C,a) \sum_{s'\in\{s_\infty\}} p(C,a,s')\big(R(C,a) \,+\, \gamma v^\pi(s')\big)\\
&= 0.1 * 1.0 * (9 + \gamma v^\pi(s_\infty)) + 0.9 * 1.0 * (\text{-}4 + \gamma v^\pi(s_\infty))\\
&= -2.7
\end{aligned}$$

For D,(by definition $v^\pi(s_\infty) = 0$)

$$\begin{aligned}
v^\pi(D) &= \sum_{a\in\{a_1,a_2\}} \pi(D,a) \sum_{s'\in\{s_\infty\}} p(D,a,s')\big(R(D,a) \,+\, \gamma v^\pi(s')\big)\\
&= 0.25 * 1.0 * (\text{-}2 + \gamma v^\pi(s_\infty)) + 0.75 * 1.0 * (6 + \gamma v^\pi(s_\infty))\\
&= 4 + 0
\end{aligned}$$

Therefore the equations are,

$$\begin{aligned}
v^\pi(A) - 0.25\gamma v^\pi(C) - 0.75 v^\pi(D) &= 4.9\\
v^\pi(B) - \gamma v^\pi(D) &= 0.6\\
v^\pi(C) &= 4.5\\
v^\pi(D) &= 4
\end{aligned}$$

(Question 5b. 5 Points) Similarly to what was shown in Lecture 06, present/write the system of equations above (which solves for v^π) in matrix form:

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet\\ \bullet & \bullet & \bullet & \bullet\\ \bullet & \bullet & \bullet & \bullet\\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} \times \begin{bmatrix} v^\pi(A)\\ v^\pi(B)\\ v^\pi(C)\\ v^\pi(D) \end{bmatrix} = \begin{bmatrix} \bullet\\ \bullet\\ \bullet\\ \bullet \end{bmatrix}, \tag{3}$$

where each entry marked with the symbol $\bullet$ indicates a value you should specify.

Solution 5b. The equations described above can be re-written as follows,

$$\begin{bmatrix} 1 & 0 & -0.25\gamma & -0.75\gamma\\ 0 & 1 & 0 & -\gamma\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} v^\pi(A)\\ v^\pi(B)\\ v^\pi(C)\\ v^\pi(D) \end{bmatrix} = \begin{bmatrix} 4.9\\ 0.6\\ -2.7\\ 4 \end{bmatrix}, \tag{4}$$

(Question 5c. 5 Points) It can be shown that the Policy Iteration algorithm, while converging towards π^*, never evaluates/considers the same policy twice (Theorem 4, Section 4.2 of the Lecture Notes). Assume we are interested in deterministic policies, and that it takes $O(n^3)$ operations to solve a system of linear equations with n equations and n unknowns. What is the *maximum* number of operations that Policy Iteration could require to identify an optimal policy? Write your answer (using Big O notation) in terms of quantities associated with the components of an MDP $(\mathcal{S}, \mathcal{A}, p, R, d_0, \gamma)$, and *explain your reasoning.*

Solution 5c. Since the Policy Iteration will not evaluate a deterministic policy twice and it converges when we get the same policy after the evaluation and improvement step. Its known that policy evaluation step takes $O(n^3)$ operations since Matrix inversion takes $O(n^3)$ and matrix multiplication takes $O(n^3)$ operations. For the policy improvement step we calculate the max of $\sum_{s'} p(s,a,s')(R(s,a)+\gamma v(s'))$ over $|A|$ actions, therefore there are $|S| * |A|$ number of operations. And we do this for each state s to get the action for rach state $\pi_{k+1}(s) = arg\max_a \sum_{s'} p(s,a,s')(R(s,a)+\gamma v(s'))$, there fore the total number of operations is $|S|^2 * |A|$, if we take $|S| = n$, the total number of operations for policy iterations is $n^2 * |A|$. Now since the policy iteration doesn't evaluate the same policy twice, the worst case would be when it computes each policy at-least once. And for finite state and finite actions MDP the maximum number of deterministic policies can be $|A|^{|S|} = |A|^n$. Using this we can find the upper bound of the total number of operations in Policy Iterations (Assuming that all actions are available in all the states)

$$O(|A|^n(n^3 + n^2 * |A|))$$

Part Two: Programming (50 Points Total)

In this question you will implement the Value Iteration algorithm and use it to find the optimal value function and the optimal policy for the 687-GridWorld domain, discussed in class. **Notice that you may not use existing RL code for this problem—you must implement the learning algorithm and the environment entirely on your own and from scratch.** The original 687-GridWorld domain can be described as follows:

- **State**: This problem consists of a 5×5 grid world where each state $s = (r, c)$ describes the current coordinates/location of the agent. In particular, $r \in [0, 4]$ is the current row where the agent is located and $c \in [0, 4]$ is the current column where the agent is located. Refer to Figure 1 for an example. In this figure, the topmost row is row zero, and the leftmost column is column zero—i.e., *State1* corresponds to $s = (0, 0)$ and *State16* corresponds to $s = (3, 1)$.
- **Actions**: There are four actions: AttemptUp (AU), AttemptDown (AD), AttemptLeft (AL), and AttemptRight (AR).
- **Dynamics**: This is a *stochastic* MDP:
  - With 80% probability, the agent moves in the specified direction.
  - With 5% probability, the agent gets confused and veers to the right with respect to the intended direction.
  - With 5% probability, the agent gets confused and veers to the left with respect to the intended direction.
  - With 10% probability, the agent temporarily breaks and does not move.
  - The grid world is surrounded by walls. If the agent hits a wall, it stays in its current state.
  - There are two *Obstacle states* in this domain: one in state $(2, 2)$ and one in state $(3, 2)$. If the agent hits an obstacle, it stays in its current state. The agent cannot enter an Obstacle state.
  - There is a *Water state* located at $(4, 2)$.
  - There is a *Goal state* located at $(4, 4)$.
- **Rewards**: The reward is always 0, except when transitioning to (entering) the Goal state, in which case the reward is 10; or when transitioning to (entering) the Water state, in which case the reward is -10. Notice that, in order to model this type of reward function, you will need to use a function in the form $R(S_t, A_t, S_{t+1})$, instead of $R(S_t, A_t)$. This requires a small modification to the Value Iteration update equation: $v_{i+1}(s) := \max\limits_{a \in \mathcal{A}} \sum_{s'} p(s, a, s')\Big(R(s, a, s') + \gamma v_i(s')\Big)$.
- **Terminal State**: The Goal state is terminal. Any actions executed in this state always transition to s_∞ with reward 0.
- **Discount**: Unless otherwise specified, $\gamma = 0.9$.
- **Initial State**: $S_0 = (0, 0)$, deterministically.

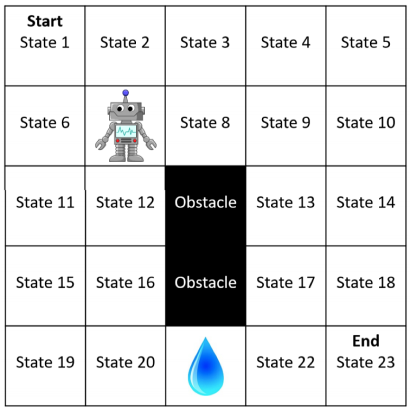
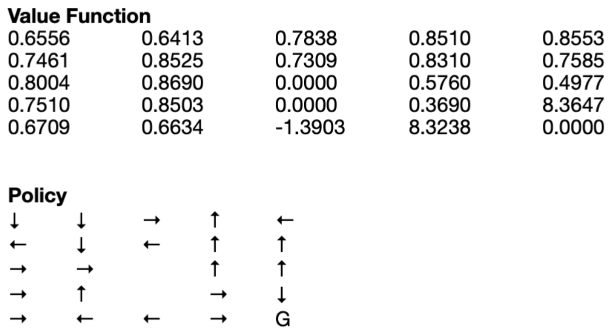


Figure 1: The original 687-GridWorld.

2. Questions (Programming)

General instructions:

- You should initialize the value of all states to zero.
- Whenever displaying values (e.g., the value of a state), use 4 decimal places; e.g, $v(s) = 9.4312$.
- In the following questions, the Value Iteration algorithm should stop whenever $\Delta < 0.0001$.
- Unless otherwise specified, you should implement the "standard" version of Value Iteration: one that, at each iteration, performs a *full backup*; and that is *not* in-place (i.e., the updated value function, v_{i+1}, at iteration $i+1$, should be computed entirely based on v_i).
- Whenever showing the value function and policy learned by your algorithm, present them in a format resembling that depicted in Figure 2.



```
Value Function
0.6556     0.6413     0.7838     0.8510     0.8553
0.7461     0.8525     0.7309     0.8310     0.7585
0.8004     0.8690     0.0000     0.5760     0.4977
0.7510     0.8503     0.0000     0.3690     8.3647
0.6709     0.6634     -1.3903    8.3238     0.0000

Policy
↓    ↓    →    ↑    ←
←    ↓    ←    ↑    ↑
→    →         ↑    ↑
→    ↑         →    ↓
→    ←    ←    →    G
```

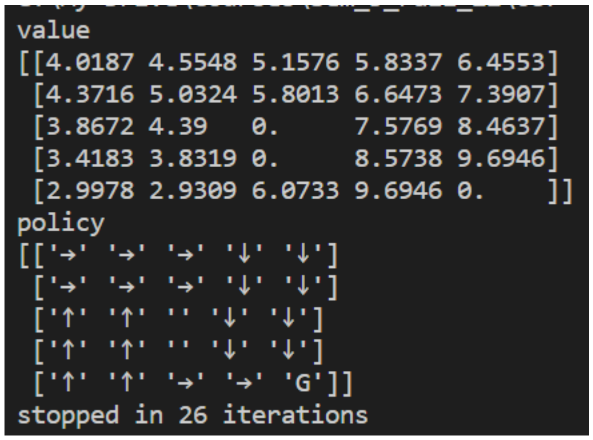
Figure 2: Example of the output format your code should generate.

1. (**14 Points**) Run the Value Iteration algorithm in the 687-GridWorld as defined above until its stopping criterion is met. *(a)* Show the final value function and the final policy identified by the algorithm. *(b)* How many iterations did it take for the algorithm to stop? *(c)* Describe, in English, the behavior being implemented by the learned policy.

   Solution 1a. Fig 3 shows the value function and the optimal policy obtained.

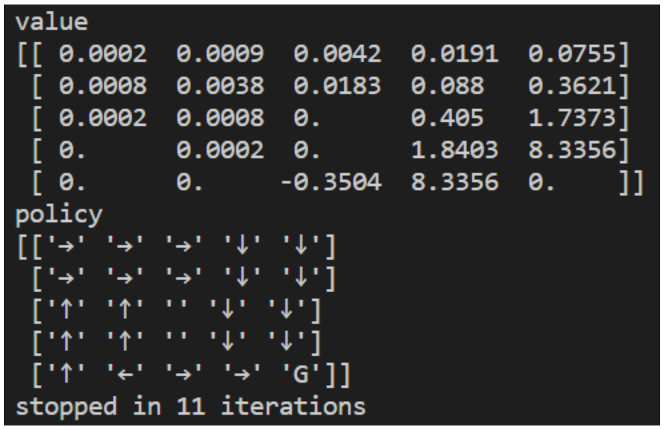
   Solution 1b. Policy converged in 26 iterations.

   Solution 1c. The learned policy tries to move towards the "Goal" state in the smallest number of steps possible, avoiding the obstacles and the water state. In the rightmost 2 columns the agent tries to move straight downwards.



```
value
[[4.0187 4.5548 5.1576 5.8337 6.4553]
 [4.3716 5.0324 5.8013 6.6473 7.3907]
 [3.8672 4.39   0.     7.5769 8.4637]
 [3.4183 3.8319 0.     8.5738 9.6946]
 [2.9978 2.9309 6.0733 9.6946 0.    ]]
policy
[['→' '→' '→' '↓' '↓']
 ['→' '→' '→' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '→' '→' 'G']]
stopped in 26 iterations
```

Figure 3: Value function and Policy



```
value
[[ 0.0002  0.0009  0.0042  0.0191  0.0755]
 [ 0.0008  0.0038  0.0183  0.088   0.3621]
 [ 0.0002  0.0008  0.      0.405   1.7373]
 [ 0.      0.0002  0.      1.8403  8.3356]
 [ 0.      0.     -0.3504  8.3356  0.    ]]
policy
[['→' '→' '→' '↓' '↓']
 ['→' '→' '→' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '←' '→' '→' 'G']]
stopped in 11 iterations
```

Figure 4: Value function and Policy

In the top rows it moves towards right. In the first 2 columns for the bottom rows the agent tries to avoid the water state and the obstacle and moves upwards.

2. (**10 Points**) Run the same experiment as above, but now using $\gamma = 0.25$. *(a)* Show the final value function and the final policy identified by the algorithm. *(b)* Did the value function identified when $\gamma = 0.25$ change with respect to the value function identified when $\gamma = 0.9$? How about the policy? If any of these quantities changed, explain, intuitively, why you think that happened. *(c)* Finally, describe how many iterations it took for the algorithm to stop. Did it take more iterations or fewer iterations, compared to when $\gamma = 0.9$? Explain, intuitively, why do you think that is the case.

   Solution 2a Fig 4 shows the value function and the optimal policy.

   Solution 2b. The value function and the policy changed. The value function changed because it is dependent on gamma directly. decreasing the gamma gives lower importance to states far away from the current state and gives more importance to the close by neighbouring states. This is why the states away from the Goal state get almost a zero value, the water state gets a negative value because it gives a high negative reward. And the states, next to the goal state gets higher value. If we look closely at the policy, we can see that the states next to the water state are trying to go away from the water state, which shows that a lower gamma gives more importance to the near by states, and it tries to avoid going into the water state by playing safe and moving away from the water state by going in the opposite direction so that the probability of going in to the water state goes down.

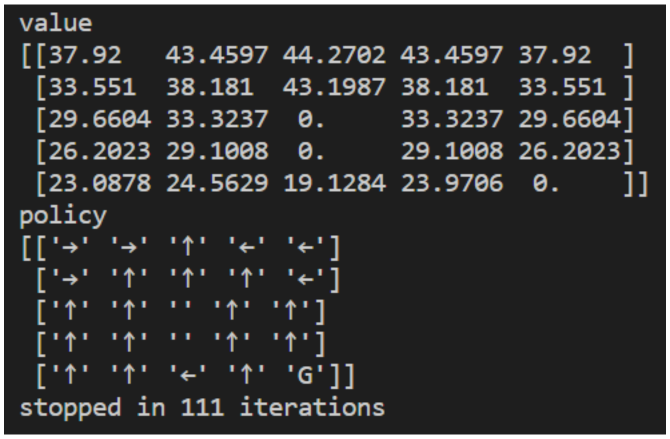
   Solution 2c. It takes 11 iterations to converge. Which is lesser than $\gamma = 0.9$. This happens because the policy only gives importance to the close by states and when that happens the value function doesn't changes much during the evaluation step. This happens because the values of the near by states doesn't change much.

3. (**10 Points**) In this question, you will once again use the Value Iteration algorithm with $\gamma = 0.9$. You will now create a state in which the agent finds gold. To do so, change the reward given to the agent when entering the state $s = (0, 2)$. Whenever that happens, the reward should be 5.0. This should not be a terminal state. *(a)* Show the final value function and the final policy identified by this algorithm. *(b)* Describe, in English, the behavior being implemented by the learned policy, and explain intuitively why this is the best policy for this variant of the domain.

   Solution 3a. Fig 5 shows the value function and the optimal policy.

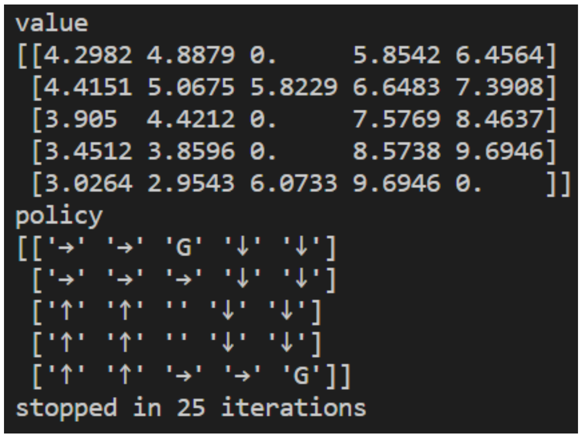
   Solution 3b. This is the best policy since, the state (0,2) is not terminal and the agent wants to go into that state infinitely. Therefor the original Goal state is not lucrative anymore. In all the states the agent is moving towards the state (0,2). Even in the state (0,2) the agent moves upwards and collides to the wall and comes back to the same state.

4. (**8 Points**) Repeat the experiment above but now make state $s = (0, 2)$ a terminal state (similarly to the Goal state). Any actions executed in this state always transition to s_∞ with reward 0. *(a)* Show the final value function and the final policy identified by the algorithm. *(b)* Describe, in English, the behavior being implemented by the learned policy,



```
value
[[37.92   43.4597 44.2702 43.4597 37.92  ]
 [33.551  38.181  43.1987 38.181  33.551 ]
 [29.6604 33.3237  0.     33.3237 29.6604]
 [26.2023 29.1008  0.     29.1008 26.2023]
 [23.0878 24.5629 19.1284 23.9706  0.    ]]
policy
[['→' '→' '↑' '←' '←']
 ['→' '↑' '↑' '↑' '←']
 ['↑' '↑' '' '↑' '↑']
 ['↑' '↑' '' '↑' '↑']
 ['↑' '↑' '←' '↑' 'G']]
stopped in 111 iterations
```

Figure 5: Value function and Optimal Policy



```
value
[[4.2982 4.8879 0.     5.8542 6.4564]
 [4.4151 5.0675 5.8229 6.6483 7.3908]
 [3.905  4.4212 0.     7.5769 8.4637]
 [3.4512 3.8596 0.     8.5738 9.6946]
 [3.0264 2.9543 6.0733 9.6946 0.    ]]
policy
[['→' '→' 'G' '↓' '↓']
 ['→' '→' '→' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '→' '→' 'G']]
stopped in 25 iterations
```

Figure 6: Value function and optimal policy

and explain intuitively why this is the best policy for this variant of the domain. *(c)* Finally, experiment with larger values of γ. What is the first larger value of γ that causes the optimal policy to change—e.g., that causes the agent to avoid the new rewarding state? Describe intuitively why changing γ in this way results in such a different behavior.

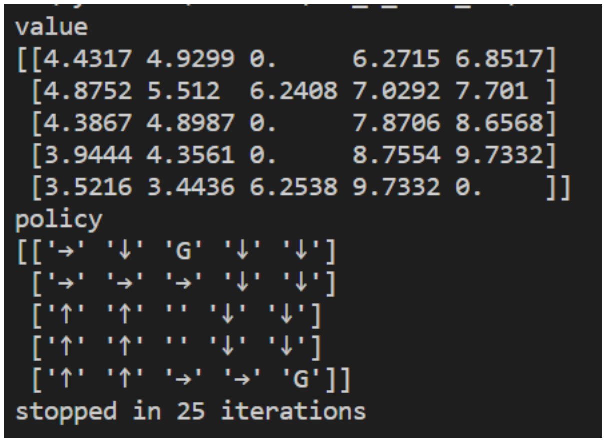
Solution 4a. Fig 6 shows the value function and the optimal policy.

Solution 4b. In this case, since the state (0,2) is a terminal state, it doesn't necessarily go into the state (0,2) always because there is a state with a better reward (4,4). Although there is still a possibility that the agent might go into the (0,2) state from the (0,1) state. But from all other states the agent tries to go to the state (4,4). Also since, $\gamma = 0.9$, which means that the farther states are given high importance, and their rewards are considered while deciding the action, and that is why for the states near (0,2), the agent doesn't necessarily takes an action towards (0,2), rather moves towards the (4,4) state.

Solution 4c. The first largest value that changes the policy is $\gamma = 0.9133$. This value causes the agent to avoid the state (0,2) because it has a lower reward than the (4,4) state, and this happens because increasing gamma give higher weightage to the far away states, and the agent takes actions that will give a higher reward than taking a greedy approach. Fig 7 shows the value function and the optimal policy

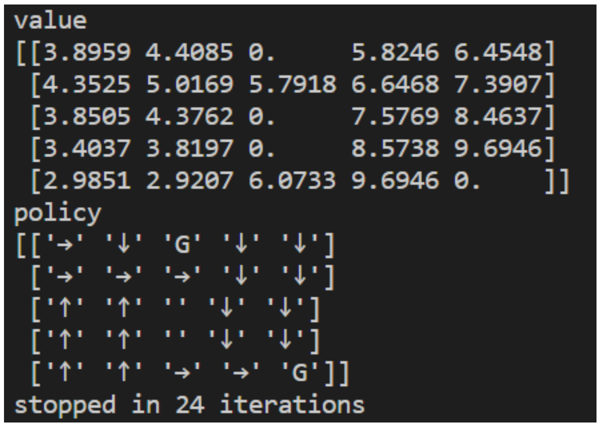
5. (**8 Points**) Repeat the experiment above (using $\gamma = 0.9$) but now experiment with other rewards for entering $s = (0, 2)$; i.e., rewards different than 5.0. *(a)* What is the first (smaller) reward value that causes the optimal policy to change—for instance, that causes the agent to avoid the new rewarding state? Describe intuitively why changing the reward in this way results in this new, different behavior. *(b)* Show the final value function and the final policy identified by the algorithm.

Solution 5a. The first smaller reward is 4.4844, that causes the optimal policy to change, and the agent



```
value
[[4.4317 4.9299 0.     6.2715 6.8517]
 [4.8752 5.512  6.2408 7.0292 7.701 ]
 [4.3867 4.8987 0.     7.8706 8.6568]
 [3.9444 4.3561 0.     8.7554 9.7332]
 [3.5216 3.4436 6.2538 9.7332 0.    ]]
policy
[['→' '↓' 'G' '↓' '↓']
 ['→' '→' '→' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '→' '→' 'G']]
stopped in 25 iterations
```

Figure 7: Value function and optimal policy



```
value
[[3.8959 4.4085 0.     5.8246 6.4548]
 [4.3525 5.0169 5.7918 6.6468 7.3907]
 [3.8505 4.3762 0.     7.5769 8.4637]
 [3.4037 3.8197 0.     8.5738 9.6946]
 [2.9851 2.9207 6.0733 9.6946 0.    ]]
policy
[['→' '↓' 'G' '↓' '↓']
 ['→' '→' '→' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '' '↓' '↓']
 ['↑' '↑' '→' '→' 'G']]
stopped in 24 iterations
```

Figure 8: Value function and optimal policy

avoids the new state. This happens because the reward from entering that state as decreased and at this reward the agent doesn't find the state very lucrative to enter at this reward. Also when we look at the value function in Fig 8 we see that the value of the neighboring states also decreased and therefore the agent moves around this state to go to the state (4,4).

Solution 5b. Fig 8 shows the Value function and the optimal policy.