HW4

Alireza Abbasi

403616204

Q5 - GRIDWORLD

١	٢	٣	۴
۵	۶	تله٧	٨
٩	١٠	١١	١٢
١٣	١۴	١۵	هدف

VI Algorithm:

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma V(s')\right]$

Q5 - GRIDWORLD

Code:

```
gridRows = 4;
gridCols = 4;
valueGrid = zeros(gridRows, gridCols);
convergenceThreshold = 0.001;
discountFactor = 0.9;
goalState = [4, 4];
trapState = [2, 3];

stateRewards = zeros(gridRows, gridCols);
stateRewards(goalState(1), goalState(2)) = 10;
stateRewards(trapState(1), trapState(2)) = -10;

actionOffsets = [0, -1; 0, 1; -1, 0; 1, 0];
numActions = size(actionOffsets, 1);

while true
    maxChange = 0;
    updatedValueGrid = valueGrid;

    for row = 1:gridRows
        for col = 1:gridCols
            if (row == goalState(1) && col == goalState(2)) || (row == trapState(1) && col == trapState(2))
                continue;
            end

            maxExpectedValue = -inf;
            for actionIdx = 1:numActions
                nextRow = row + actionOffsets(actionIdx, 1);
                nextCol = col + actionOffsets(actionIdx, 2);
```

Q5 - GRIDWORLD

Code:

```
                if nextRow < 1 || nextRow > gridRows || nextCol < 1 || nextCol > gridCols
                    nextRow = row;
                    nextCol = col;
                end

                actionValue = stateRewards(nextRow, nextCol) + discountFactor * valueGrid(nextRow, nextCol);
                maxExpectedValue = max(maxExpectedValue, actionValue);
            end

            updatedValueGrid(row, col) = maxExpectedValue;
            maxChange = max(maxChange, abs(valueGrid(row, col) - maxExpectedValue));
        end
    end

    if maxChange < convergenceThreshold
        break;
    end
    valueGrid = updatedValueGrid;
end

optimalPolicy = strings(gridRows, gridCols);
for row = 1:gridRows
    for col = 1:gridCols
        if (row == goalState(1) && col == goalState(2))
            optimalPolicy(row, col) = "Goal";
            continue;
        elseif (row == trapState(1) && col == trapState(2))
            optimalPolicy(row, col) = "Trap";
            continue;
```

Q5 - GRIDWORLD

Code:

```
    end

    maxExpectedValue = -inf;
    bestAction = "";
    for actionIdx = 1:numActions
        nextRow = row + actionOffsets(actionIdx, 1);
        nextCol = col + actionOffsets(actionIdx, 2);

        if nextRow < 1 || nextRow > gridRows || nextCol < 1 || nextCol > gridCols
            nextRow = row;
            nextCol = col;
        end

        actionValue = stateRewards(nextRow, nextCol) + discountFactor * valueGrid(nextRow, nextCol);
        if actionValue > maxExpectedValue
            maxExpectedValue = actionValue;
            switch actionIdx
                case 1
                    bestAction = "Left";
                case 2
                    bestAction = "Right";
                case 3
                    bestAction = "Up";
                case 4
                    bestAction = "Down";
            end
        end
    end
    optimalPolicy(row, col) = bestAction;
end
```

Q5 - GRIDWORLD

Result:

```
>> Q5_v1
Optimal Value Function:
    5.9049    6.5610    7.2900    8.1000
    6.5610    7.2900         0    9.0000
    7.2900    8.1000    9.0000   10.0000
    8.1000    9.0000   10.0000         0

Optimal Policy:
    "Right"    "Right"    "Right"    "Down"
    "Right"    "Down"     "Trap"     "Down"
    "Right"    "Right"    "Right"    "Down"
    "Right"    "Right"    "Right"    "Goal"
```

Q5 - GRIDWORLD

۱	۲	۳	۴
۵	۶	تله۷	۸
۹	۱۰	۱۱	۱۲
۱۳	۱۴	۱۵	هدف

PI Algorithm:

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Q5 - GRIDWORLD

Code:

```
gridRows = 4;
gridCols = 4;
valueMatrix = zeros(gridRows, gridCols);
actionPolicy = repmat("Left", gridRows, gridCols);

threshold = 0.01;
discountFactor = 0.9;
goalState = [4, 4];
trapState = [2, 3];

rewardMatrix = zeros(gridRows, gridCols);
rewardMatrix(goalState(1), goalState(2)) = 10;
rewardMatrix(trapState(1), trapState(2)) = -10;

possibleActions = ["Left", "Right", "Up", "Down"];
actionOffsets = [0, -1; 0, 1; -1, 0; 1, 0];

computeNextState = @(x, y, actionIdx) deal(...
    max(1, min(gridRows, x + actionOffsets(actionIdx, 1))), ...
    max(1, min(gridCols, y + actionOffsets(actionIdx, 2))));

while true
    while true
        maxDelta = 0;
        for row = 1:gridRows
            for col = 1:gridCols
                if isequal([row, col], goalState) || isequal([row, col], trapState)
                    continue;
                end
                oldValue = valueMatrix(row, col);
                actionIdx = find(possibleActions == actionPolicy(row, col));
```

Q5 - GRIDWORLD

Code:

```
                [nextRow, nextCol] = computeNextState(row, col, actionIdx);
                valueMatrix(row, col) = rewardMatrix(nextRow, nextCol) + ...
                                        discountFactor * valueMatrix(nextRow, nextCol);
                maxDelta = max(maxDelta, abs(oldValue - valueMatrix(row, col)));
            end
        end
        if maxDelta < threshold
            break;
        end
    end

    isPolicyStable = true;
    for row = 1:gridRows
        for col = 1:gridCols
            if isequal([row, col], goalState)
                actionPolicy(row, col) = "Goal";
                continue;
            elseif isequal([row, col], trapState)
                actionPolicy(row, col) = "Trap";
                continue;
            end

            previousAction = actionPolicy(row, col);
            actionValues = zeros(1, numel(possibleActions));
            for actionIdx = 1:numel(possibleActions)
                [nextRow, nextCol] = computeNextState(row, col, actionIdx);
                actionValues(actionIdx) = rewardMatrix(nextRow, nextCol) + ...
                                          discountFactor * valueMatrix(nextRow, nextCol);
            end
            [maxValue, optimalActionIdx] = max(actionValues);
            actionPolicy(row, col) = possibleActions(optimalActionIdx);
```

Q5 - GRIDWORLD

Code:

```
            [nextRow, nextCol] = computeNextState(row, col, actionIdx);
            valueMatrix(row, col) = rewardMatrix(nextRow, nextCol) + ...
                                    discountFactor * valueMatrix(nextRow, nextCol);
            maxDelta = max(maxDelta, abs(oldValue - valueMatrix(row, col)));
        end
    end
    if maxDelta < threshold
        break;
    end
end

isPolicyStable = true;
for row = 1:gridRows
    for col = 1:gridCols
        if isequal([row, col], goalState)
            actionPolicy(row, col) = "Goal";
            continue;
        elseif isequal([row, col], trapState)
            actionPolicy(row, col) = "Trap";
            continue;
        end

        previousAction = actionPolicy(row, col);
        actionValues = zeros(1, numel(possibleActions));
        for actionIdx = 1:numel(possibleActions)
            [nextRow, nextCol] = computeNextState(row, col, actionIdx);
            actionValues(actionIdx) = rewardMatrix(nextRow, nextCol) + ...
                                      discountFactor * valueMatrix(nextRow, nextCol);
        end
        [maxValue, optimalActionIdx] = max(actionValues);
        actionPolicy(row, col) = possibleActions(optimalActionIdx);

        if previousAction ~= actionPolicy(row, col)
            isPolicyStable = false;
        end
    end
end

if isPolicyStable
    break;
end
end

disp("Optimal Value Matrix:");
disp(valueMatrix);
disp("Optimal Action Policy:");
disp(actionPolicy);
```

Q5 - GRIDWORLD

Result:

```
>> Q5_P1
Optimal Value Matrix:
    5.9049    6.5610    7.2900    8.1000
    6.5610    7.2900         0    9.0000
    7.2900    8.1000    9.0000   10.0000
    8.1000    9.0000   10.0000         0

Optimal Action Policy:
    "Right"    "Right"    "Right"    "Down"
    "Right"    "Down"     "Trap"     "Down"
    "Right"    "Right"    "Right"    "Down"
    "Right"    "Right"    "Right"    "Goal"
```

Q6 - FROZENLAKE

PI Algorithm- results:

```
Policy Iteration Results:

Optimal Policy (4x4 Grid):
['↓', '→', '↓', '←']
['↓', '←', '↓', '←']
['→', '↓', '↓', '←']
['←', '→', '→', '←']

Optimal Value Function (4x4 Grid):
['0.95', '0.96', '0.97', '0.96']
['0.96', '0.00', '0.98', '0.00']
['0.97', '0.98', '0.99', '0.00']
['0.00', '0.99', '1.00', '0.00']
```

Q6 - FROZENLAKE

VI Algorithm- results:

```
Value Iteration Results:

Optimal Policy (4x4 Grid):
['↓', '→', '↓', '←']
['↓', '←', '↓', '←']
['→', '↓', '↓', '←']
['←', '→', '→', '←']

Optimal Value Function (4x4 Grid):
['0.95', '0.96', '0.97', '0.96']
['0.96', '0.00', '0.98', '0.00']
['0.97', '0.98', '0.99', '0.00']
['0.00', '0.99', '1.00', '0.00']
```

Q6 - FROZENLAKE

On-policy Monte Carlo - results:

```
Episode 5000/50000
Success Rate: 38.12%
Average Reward per Episode: 0.3812

Current Policy:
['↓', '→', '↓', '←']
['↓', '←', '↓', '↑']
['→', '→', '↓', '↓']
['↑', '→', '→', '→']

Episode 10000/50000
Success Rate: 41.24%
Average Reward per Episode: 0.4124

Current Policy:
['↓', '→', '↓', '←']
['↓', '←', '↓', '↑']
['→', '↓', '↓', '↓']
['↑', '→', '→', '→']

Episode 15000/50000
Success Rate: 53.16%
Average Reward per Episode: 0.5316
...
['↓', '→', '↓', '←']
['↓', '←', '↓', '↑']
['→', '↓', '↓', '↓']
['↑', '→', '→', '→']
```

Q6 - FROZENLAKE

Off-policy Monte Carlo - results:

```
Episode 5000/50000
Success Rate: 36.64%
Average Reward per Episode: 0.3664

Current Policy:
['↓', '←', '←', '←']
['↓', '←', '↓', '→']
['→', '→', '↓', '↓']
['←', '→', '→', '↑']

Episode 10000/50000
Success Rate: 39.44%
Average Reward per Episode: 0.3944

Current Policy:
['↓', '←', '←', '←']
['↓', '←', '↓', '→']
['→', '→', '↓', '↓']
['←', '→', '→', '↑']

Episode 15000/50000
Success Rate: 51.58%
Average Reward per Episode: 0.5158
...
['↓', '←', '←', '←']
['↓', '←', '↓', '→']
['→', '→', '↓', '↓']
['←', '→', '→', '↑']
```