Project: 1

1.) Approach:

a.) Background:

For the given environment we use *Value Iteration* wherein, updates on the value function $V_k(s)$ are made by greedifying the w.r.t the actions $a \epsilon A(s)$. This is an elegant way of eliminating “the policy evaluation” step from the policy iteration algorithm. Additionally, we still guarantee convergence without making policy evaluations after every iteration. Following is the value iteration algorithm based on the book by Sutton and Barto- “Reinforcement Learning: An Introduction.”

```
Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in S^+$)
Repeat
    $\Delta \leftarrow 0$ For each $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\,[r + + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\,[r + + \gamma V(s')]$
```

The equation still provides us with the optimal policy mainly because the Bellman Optimality conditions for the value function are still satisfied (due to greedifying on each iteration which results in $\boldsymbol{V_k(s) \leq V_{k+1}(s)}$). Finally, the algorithm converges after an infinite number of iterations, however, in smaller discrete environments, it usually does so in a finite number of updates. This is taken care of by using a difference check method, i.e., we stop the updations if the difference in the value function outputs ($|v - V(s)|$) is less than a predefined tolerance value.

b.) Implementation:

We implement the *Value Iteration* algorithm as explained above for the *“FrozenLake-v0”* environment on OpenAI-gym. In this project, we turn the (is_slippery) hidden Markov transitions off to focus on the basic algorithm used. The environment is grid-based and can be visualized as follows:

State Space			
0	1	2	3
4	5	6	7
8	9	10	11
11	12	13	14

Action Space	
LEFT	0
DOWN	1
RIGHT	2
UP	3

Each of the state space cells can have the value H/F/G/S indicating a “hole”, “frozen section”, “goal”, or “Start” respectively. The agent can take 4 actions and will transition accordingly, hitting the walls brings it back to the same cell with 0 rewards. The only transition that results in a +1 reward is when it reaches the cell *“G”*.

c.) **Algorithm:**

Below, we define the pseudo-code for our implementation, it is granular enough and can be extended to other similar grid-world problems.

Pseudo-code: Value Iteration

Inputs:

- The frozen lake environment.
- Discount factor (γ) …(0.9 value was used for the experiments)

Returns:

- State value function ~V^*
- State-action value function ~Q^*

```
Initialize: $V(s)$, $v(s)$ arrays with the size = $|S|$
            $Q(s, a)$ array with size = $|S|$ x $|A(s)|$ ….( $A(s)$ in this case is = 4 for all the states)
            $\Pi(s)$ arrays with the size = $|S|$

while (iteration ≤ Maximum allowed iterations):
   for $s$ in range($|S|$):
   |   for $a$ in range($|A(s)|$):
   |   |   for probability ($p$), new state ($s\_$), reward ($r$) in transition[s][a]:
   |   |       $Q(s, a) \mathrel{+}= p*(r + \gamma * v[s\_])$
   |   Maximizing action: $a_{max} = \text{argmax}(Q(s, *))$
   |   $V[s] = Q(s, a_{max})$
   |   $\Pi(s) = a_{max}$
   |
   $v = V$
return $V(s)$, $Q(s, a)$, $\Pi(s)$
```

In the given question from the report, we need to find the policy separately, however, we can achieve this in the value iteration function itself as shown above as well. In any case, given the optimal value function, the policy can separately be found out as follows:

Pseudo-code: Policy Extraction

Inputs:

- State value function ~V^*.
- Discount factor (γ)

Returns:

- Policy ~$\Pi^*(s)$

```
Initialize: $\Pi(s)$ arrays with the size = $|S|$

   for $s$ in range($|S|$):
   |   for $a$ in range($|A(s)|$):
   |   |   for probability ($p$), new state ($s\_$), reward ($r$) in transition[s][a]:
   |   |       $Q(s, a) \mathrel{+}= p*(r + \gamma * V[s\_])$
   |   Maximizing action: $a_{max} = \text{argmax}(Q(s, *))$
   |   $\Pi(s) = a_{max}$
   return $\Pi(s)$
```

2.) <u>Experimental Results</u>:

The delta approach was not used in this case, however, a constant number of max iterations = 7 was used. The values didn't change much after that threshold. Additionally, the board chosen was of the default category, the state space, value function, and the optimal policy are given below:

State Space			
S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

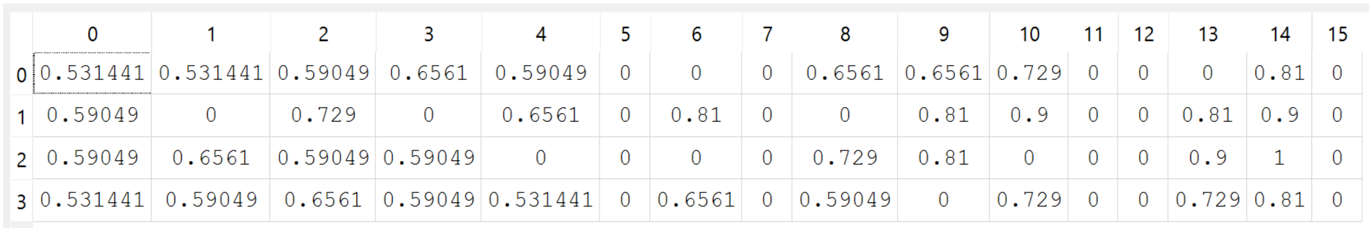
State Space

V(s)*			
0.59	0.65	0.72	0.65
0.65	0	0.81	0
0.72	0.81	0.90	0
0	0.90	1	0

Optimal State Values

Π(s)*			
↓→	→	←	↓
↓	–	↓	–
→	↓→	↓	–
–	→	→	G

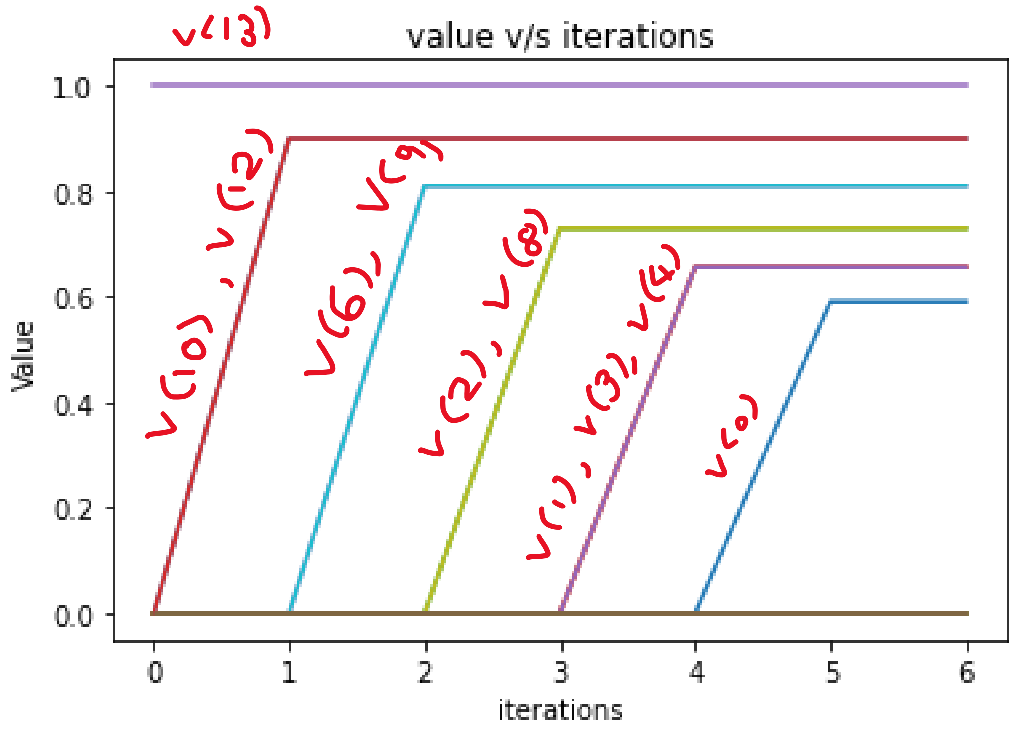
Optimal Policy



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.531441	0.531441	0.59049	0.6561	0.59049	0	0	0	0.6561	0.6561	0.729	0	0	0	0.81	0
1	0.59049	0	0.729	0	0.6561	0	0.81	0	0	0.81	0.9	0	0	0.81	0.9	0
2	0.59049	0.6561	0.59049	0.59049	0	0	0	0	0.729	0.81	0	0	0	0.9	1	0
3	0.531441	0.59049	0.6561	0.59049	0.531441	0	0.6561	0	0.59049	0	0.729	0	0	0.729	0.81	0

Optimal Q-values

Experimentally, it was seen that the optimal values, in this case, were reached after nearly the 6th iteration only. The below graph depicts the value function outputs for all the states. All the 0 value states remained the same throughout and can be seen at the x-axis itself.



3.) Discussion:

Upon implementing the Value Iteration Algorithm, we were able to solve the frozen lake problem, however, we need to discuss the important observations, advantages, and disadvantages of the same.

a.) Key observations.

- The worst-case time complexity of the Value Iteration is ~$|S^+|*|A(s)|*k$. Additionally, the value of ‘k’ is the maximum number of iterations and it depends upon the environment and would vary in different cases.
- The worst-case space complexity of the Value Iteration is ~$|S^+|*|A(s)|$. In addition to this, the algorithm will either require extensive experimentation beforehand to figure out the max iterations required or if the delta–theta method is used then the value of theta is to be determined which can be different based on the reward system and the environment as a whole.
- In the Bruteforce case, however, the time complexity will be ~$|A(s)|^{|S|}$.

b.) Advantages of VI.

- Much faster than the Bruteforce method, easier to implement than Policy iteration.
- Efficient for the smaller state and action-space problems.
- The algorithm must converge to the global minimum eventually for deterministic environments.

c.) Disadvantages of VI.

- Suffers from the curse of dimensionality as the state-space will grow exponentially w.r.t the features used.
- High space complexity as the algorithm maintains a state-action value for each pair.
- Cannot be used with continuous domain states/actions directly (some sort of value function approximation needs to be used e.g., tile coding or NNs).
- Suffers greatly from the hidden Markov transitions for stochastic actions (observed experimentally by setting is_slippery = True in the programming section).
- Assumes the knowledge of the environment to work correctly (Fully observable MDP), thus we can’t use it in areas where the knowledge of the full environment is not known.

In addition to the above, this lays the foundation for future concepts like the TD(0) algorithm, which is a combination of the Monte Carlo approach and Dynamic Programming.