

```

1  # value iteration example
2  #We have a 4x4 Gridworld where an agent moves in one of four directions: up, down, left, or right.
3  #The goal is to reach the terminal state (bottom-right corner) while maximizing cumulative rewards.
4  #The agent starts at any state (except the goal).
5  #Every move incurs a small penalty (-0.04), to encourage shorter paths.
6  #The agent cannot move outside the grid (it stays in place if it tries).
7  #The goal state (bottom-right corner) gives a reward of 1 and is terminal.
8  #The agent wants to find the best path to the goal.
9

```

```

1  import numpy as np
2
3  # Grid World parameters
4  grid_size = 4 # you can increase the grid size to increase the complexity of the problem
5  num_states = grid_size * grid_size
6  gamma = 0.9 # Discount factor
7  theta = 1e-4 # Convergence threshold
8  step_cost = -0.04 # Small penalty per move
9
10 # Actions and transitions
11 actions = ["UP", "DOWN", "LEFT", "RIGHT"]
12 action_deltas = {"UP": -grid_size, "DOWN": grid_size, "LEFT": -1, "RIGHT": 1}
13
14 # Initialize Value Function
15 V = np.zeros(num_states)
16 V[-1] = 1.0 # ✅ Ensure goal state starts with a value of 1.0
17
18 # Define rewards
19 rewards = np.full(num_states, step_cost) # Small cost everywhere
20 rewards[-1] = 1 # Goal state reward
21
22 print("Initial Rewards:")
23 print(rewards.reshape((grid_size, grid_size)))
24
25 # Value Iteration Algorithm
26 iteration = 0
27 while True:
28     delta = 0
29     V_new = np.copy(V)
30
31     print(f"\nIteration {iteration}:")
32     print(np.round(V.reshape((grid_size, grid_size)), 2))
33
34     for s in range(num_states):
35         if s == num_states - 1: # Goal state remains unchanged
36             continue
37
38         max_value = float('-inf')
39
40         for action in actions:
41             s_next = s + action_deltas[action]
42
43             # Ensure valid moves
44             if s_next < 0 or s_next >= num_states:
45                 continue
46             if action == "LEFT" and s % grid_size == 0:
47                 continue
48             if action == "RIGHT" and (s + 1) % grid_size == 0:
49                 continue
50
51             # ✅ Correct Bellman Update: Max over possible actions
52             value = rewards[s] + gamma * V[s_next]
53             max_value = max(max_value, value)
54
55         V_new[s] = max_value
56         delta = max(delta, abs(V_new[s] - V[s]))
57
58     V = V_new
59     iteration += 1
60     if delta < theta:
61         break # Convergence
62
63 # Final Value Function
64 print("\nFinal Optimal Value Function:")
65 print(np.round(V.reshape((grid_size, grid_size)), 2))
66 --

```

```

↩ Initial Rewards:
[[-0.04 -0.04 -0.04 -0.04]
 [-0.04 -0.04 -0.04 -0.04]
 [-0.04 -0.04 -0.04 -0.04]
 [-0.04 -0.04 -0.04 1.  ]]

Iteration 0:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 1.]]

Iteration 1:
[[-0.04 -0.04 -0.04 -0.04]
 [-0.04 -0.04 -0.04 -0.04]
 [-0.04 -0.04 -0.04 0.86]
 [-0.04 -0.04 0.86 1.  ]]

Iteration 2:
[[-0.08 -0.08 -0.08 -0.08]
 [-0.08 -0.08 -0.08 0.73]
 [-0.08 -0.08 0.73 0.86]
 [-0.08 0.73 0.86 1.  ]]

Iteration 3:
[[-0.11 -0.11 -0.11 0.62]
 [-0.11 -0.11 0.62 0.73]
 [-0.11 0.62 0.73 0.86]
 [ 0.62 0.73 0.86 1.  ]]

Iteration 4:
[[-0.14 -0.14 0.52 0.62]
 [-0.14 0.52 0.62 0.73]
 [ 0.52 0.62 0.73 0.86]
 [ 0.62 0.73 0.86 1.  ]]

Iteration 5:
[[-0.16 0.43 0.52 0.62]
 [ 0.43 0.52 0.62 0.73]
 [ 0.52 0.62 0.73 0.86]
 [ 0.62 0.73 0.86 1.  ]]

Iteration 6:
[[0.34 0.43 0.52 0.62]
 [0.43 0.52 0.62 0.73]
 [0.52 0.62 0.73 0.86]
 [0.62 0.73 0.86 1.  ]]

Final Optimal Value Function:
[[0.34 0.43 0.52 0.62]
 [0.43 0.52 0.62 0.73]
 [0.52 0.62 0.73 0.86]
 [0.62 0.73 0.86 1.  ]]

1  # policy iteration for the above example
2  import numpy as np
3
4  # Grid World parameters
5  grid_size = 4 # Can be increased to increase complexity
6  num_states = grid_size * grid_size
7  gamma = 0.9 # Discount factor
8  theta = 1e-4 # Convergence threshold
9  step_cost = -0.04 # Small penalty per move
10
11 # Actions and transitions
12 actions = ["UP", "DOWN", "LEFT", "RIGHT"]
13 action_deltas = {"UP": -grid_size, "DOWN": grid_size, "LEFT": -1, "RIGHT": 1}
14
15 # Initialize Value Function and Policy
16 V = np.zeros(num_states)
17 policy = np.random.choice(actions, size=num_states) # Random initial policy
18 V[-1] = 1.0 # Goal state has value 1.0
19
20 # Define rewards
21 rewards = np.full(num_states, step_cost)
22 rewards[-1] = 1 # Goal state reward
23
24 print("Initial Policy:")
25 print(policy.reshape((grid_size, grid_size)))
26

```

```

27 # Policy Iteration
28 policy_stable = False
29 iteration = 0
30
31 while not policy_stable:
32     # Policy Evaluation
33     while True:
34         delta = 0
35         V_new = np.copy(V)
36
37         for s in range(num_states):
38             if s == num_states - 1: # Skip goal state
39                 continue
40
41             action = policy[s]
42             s_next = s + action_deltas[action]
43
44             # Ensure valid moves
45             if s_next < 0 or s_next >= num_states:
46                 continue
47             if action == "LEFT" and s % grid_size == 0:
48                 continue
49             if action == "RIGHT" and (s + 1) % grid_size == 0:
50                 continue
51
52             # Bellman Equation for Policy Evaluation
53             V_new[s] = rewards[s] + gamma * V[s_next]
54             delta = max(delta, abs(V_new[s] - V[s]))
55
56         V = V_new
57         if delta < theta:
58             break # Convergence
59
60     # Policy Improvement
61     policy_stable = True
62     for s in range(num_states):
63         if s == num_states - 1: # Skip goal state
64             continue
65
66         best_action = None
67         best_value = float('-inf')
68
69         for action in actions:
70             s_next = s + action_deltas[action]
71
72             # Ensure valid moves
73             if s_next < 0 or s_next >= num_states:
74                 continue
75             if action == "LEFT" and s % grid_size == 0:
76                 continue
77             if action == "RIGHT" and (s + 1) % grid_size == 0:
78                 continue
79
80             value = rewards[s] + gamma * V[s_next]
81             if value > best_value:
82                 best_value = value
83                 best_action = action
84
85         if policy[s] != best_action:
86             policy_stable = False # Policy changed, need another iteration
87             policy[s] = best_action
88
89     iteration += 1
90     print(f"\nIteration {iteration}:")
91     print("Value Function:")
92     print(np.round(V.reshape((grid_size, grid_size)), 2))
93     print("Policy:")
94     print(policy.reshape((grid_size, grid_size)))
95
96 print("\nFinal Optimal Value Function:")
97 print(np.round(V.reshape((grid_size, grid_size)), 2))
98
99 print("\nFinal Optimal Policy:")
100 print(policy.reshape((grid_size, grid_size)))
101

```



Initial Policy:

```
[[ 'UP' 'DOWN' 'RIGHT' 'LEFT' ]  
 [ 'UP' 'UP' 'DOWN' 'RIGHT' ]  
 [ 'DOWN' 'LEFT' 'RIGHT' 'RIGHT' ]  
 [ 'UP' 'LEFT' 'RIGHT' 'RIGHT' ]]
```

Iteration 1:

Value Function:

```
[[ 0.  -0.4 -0.4 -0.4 ]  
 [-0.04 -0.4 -0.08 0. ]  
 [-0.4 -0.4 -0.04 0. ]  
 [-0.4 -0.4 0.86 1.  ]]
```

Policy:

```
[[ 'DOWN' 'LEFT' 'DOWN' 'DOWN' ]  
 [ 'UP' 'LEFT' 'RIGHT' 'DOWN' ]  
 [ 'UP' 'RIGHT' 'DOWN' 'DOWN' ]  
 [ 'UP' 'RIGHT' 'RIGHT' 'RIGHT' ]]
```

Iteration 2:

Value Function:

```
[[ -0.4 -0.4 0.52 0.62 ]  
 [-0.4 -0.4 0.62 0.73 ]  
 [-0.4 0.62 0.73 0.86 ]  
 [-0.4 0.73 0.86 1.  ]]
```

Policy:

```
[[ 'DOWN' 'RIGHT' 'DOWN' 'DOWN' ]  
 [ 'UP' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'RIGHT' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'RIGHT' 'RIGHT' 'RIGHT' 'RIGHT' ]]
```

Iteration 3:

Value Function:

```
[[ -0.4 0.43 0.52 0.62 ]  
 [-0.4 0.52 0.62 0.73 ]  
 [ 0.52 0.62 0.73 0.86 ]  
 [ 0.62 0.73 0.86 1.  ]]
```

Policy:

```
[[ 'RIGHT' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'DOWN' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'DOWN' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'RIGHT' 'RIGHT' 'RIGHT' 'RIGHT' ]]
```

Iteration 4:

Value Function:

```
[[ 0.34 0.43 0.52 0.62 ]  
 [ 0.43 0.52 0.62 0.73 ]  
 [ 0.52 0.62 0.73 0.86 ]  
 [ 0.62 0.73 0.86 1.  ]]
```

Policy:

```
[[ 'DOWN' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'DOWN' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'DOWN' 'DOWN' 'DOWN' 'DOWN' ]  
 [ 'RIGHT' 'RIGHT' 'RIGHT' 'RIGHT' ]]
```

Iteration 5:

Value Function:

```
[[ 0.34 0.43 0.52 0.62 ]  
 [ 0.43 0.52 0.62 0.73 ]]
```