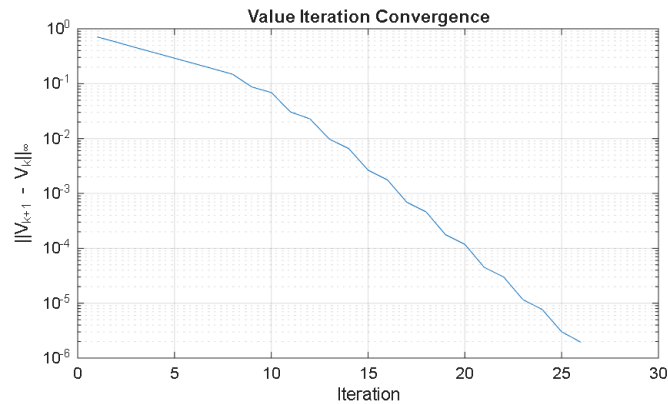


**3.2.i)**

Using: Algorithm 6 compute the value function. Hint: You should obtain a value function as shown in Figure 3.3.

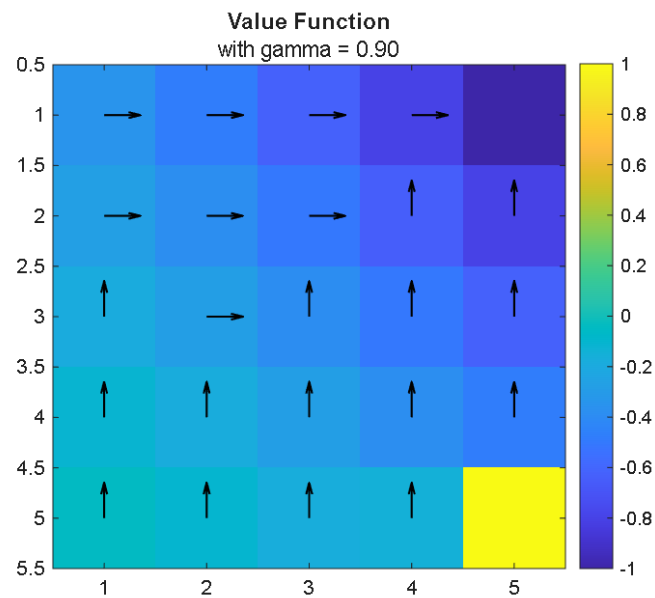
Solution:

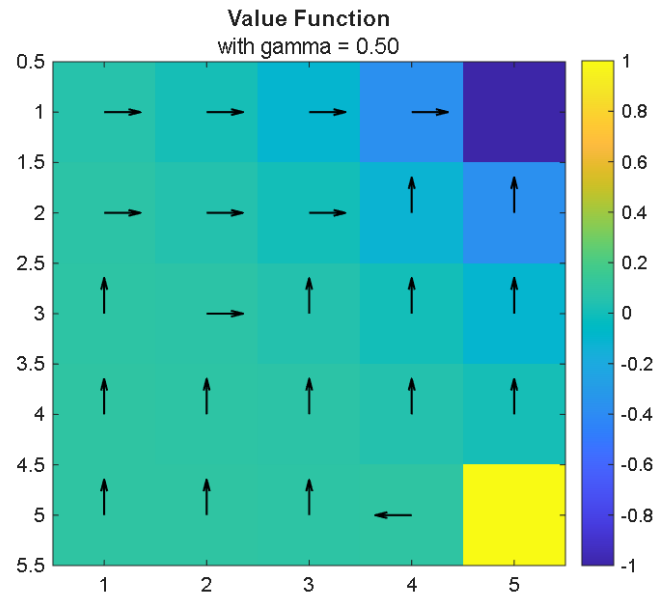
Figure 1: Convergence of value iteration algorithm for  $\gamma = 0.9$

**3.2.ii)**

Compute: the optimal policy. Display the optimal action at each state by showing an arrow in each cell.

Solution:





We can see as  $\gamma$  is decreased the policy becomes more risk averse because in the square to the left of the pit the policy moves in the opposite direction so there is no chance of slipping into it. For higher  $\gamma$  it takes this risk at the reward of potentially getting to the goal faster.

## Code Snippet 1: Q3.2 Code

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % AAE590 Stochastic Control
3  % Machine Replacement Dynamic Programming
4  % Author: Travis Hastreiter
5  % Created On: 16 September, 2025
6  % Description: Solves machine replacement problem using dynamic
7  % programming.
8  % Most Recent Change: 16 September, 2025
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11 C_pit = 1;
12 C_goal = -1;
13 C_step = 0.05;
14 p = 0.2;
15
16 C = C_step * ones([5, 5]);
17
18 % Create absorbing states
19 X_abs = zeros([5, 5]);
20 % Goal
21 goal_i = 5;
22 X_abs(goal_i) = 1;
23 C(goal_i) = -1;
24 % Pit
25 pit_i = 25;
26 X_abs(pit_i) = 1;
27 C(pit_i) = 1;
28
29 % Create regular states
30 X_reg = ~X_abs;
31
32 % Get regular and absorbing state indices
33 X_reg_i = find(X_reg);
34 X_abs_i = find(X_abs);
35
36 % Create U
37 U = 1 : 4; % { Left , Right , Up , Down }
38
39 % Create tau
40 tau = zeros(5 ^ 2, 5 ^ 2, 4);
41
42 % Absorbing state
43 tau(X_abs_i(1), X_abs_i(1), :) = 1;
44 tau(X_abs_i(2), X_abs_i(2), :) = 1;
45
46 %% Left
47 % Left is wall, not at corner
48 for i = 2 : 5
49     tau(i, i, 1) = 1 - p;
50 end
51
52 % Left is wall, at corner
53 for i = [1, 5]

```

```

54     tau(i, i, 1) = 1 - p / 2;
55 end
56
57 % Left isn't wall
58 for i = 6 : 5 * 5
59     tau(i - 5, i, 1) = 1 - p;
60 end
61
62 % Up isn't wall
63 for i = reshape((2 : 5) + 5 * (0:4)', 1, [])
64     tau(i - 1, i, 1) = p / 2;
65 end
66
67 % Up or down is wall, not at left corner
68 for i = reshape([1, 5] + 5 * (1:4)', 1, [])
69     tau(i, i, 1) = p / 2;
70 end
71
72 % Down isn't wall
73 for i = reshape((1 : 4) + 5 * (0:4)', 1, [])
74     tau(i + 1, i, 1) = p / 2;
75 end
76
77 %% Right
78 % Right is wall, not at corner
79 for i = 22 : 24
80     tau(i, i, 2) = 1 - p;
81 end
82
83 % Right is wall, at corner
84 for i = [21, 25]
85     tau(i, i, 2) = 1 - p / 2;
86 end
87
88 % Right isn't wall
89 for i = 1 : 5 * 4
90     tau(i + 5, i, 2) = 1 - p;
91 end
92
93 % Up isn't wall
94 for i = reshape((2 : 5) + 5 * (0 : 4)', 1, [])
95     tau(i - 1, i, 2) = p / 2;
96 end
97
98 % Up or down is wall, not at right corner
99 for i = reshape([1, 5] + 5 * (0 : 3)', 1, [])
100     tau(i, i, 2) = p / 2;
101 end
102
103 % Down isn't wall
104 for i = reshape((1 : 4) + 5 * (0 : 4)', 1, [])
105     tau(i + 1, i, 2) = p / 2;
106 end
107

```

```
108 %% Up
109 % Up is wall, not at corner
110 for i = 1 + 5 * (1 : 3)
111     tau(i, i, 3) = 1 - p;
112 end
113
114 % Up is wall, at corner
115 for i = [1, 21]
116     tau(i, i, 3) = 1 - p / 2;
117 end
118
119 % Up isn't wall
120 for i = reshape((2 : 5) + 5 * (0 : 4)', 1, [])
121     tau(i - 1, i, 3) = 1 - p;
122 end
123
124 % Right isn't wall
125 for i = reshape((1 : 5) + 5 * (0 : 3)', 1, [])
126     tau(i + 5, i, 3) = p / 2;
127 end
128
129 % Right or left is wall, not at upper corner
130 for i = reshape((2 : 5) + 5 * [0, 4]', 1, [])
131     tau(i, i, 3) = p / 2;
132 end
133
134 % Left isn't wall
135 for i = reshape((1 : 5) + 5 * (1 : 4)', 1, [])
136     tau(i - 5, i, 3) = p / 2;
137 end
138
139 %% Down
140 % Down is wall, not at corner
141 for i = 5 + 5 * (1 : 3)
142     tau(i, i, 4) = 1 - p;
143 end
144
145 % Down is wall, at corner
146 for i = [5, 25]
147     tau(i, i, 4) = 1 - p / 2;
148 end
149
150 % Down isn't wall
151 for i = reshape((1 : 4) + 5 * (0 : 4)', 1, [])
152     tau(i + 1, i, 4) = 1 - p;
153 end
154
155 % Right isn't wall
156 for i = reshape((1 : 5) + 5 * (0 : 3)', 1, [])
157     tau(i + 5, i, 4) = p / 2;
158 end
159
160 % Right or left is wall, not at lower corner
161 for i = reshape((1 : 4) + 5 * [0, 4]', 1, [])
```

```

162     tau(i, i, 4) = p / 2;
163 end
164
165 % Left isn't wall
166 for i = reshape((1 : 5) + 5 * (1 : 4)', 1, [])
167     tau(i - 5, i, 4) = p / 2;
168 end
169
170 %% Solve
171 % Algorithm parameters
172 gamma = 0.5;
173 max_iter = 100;
174 eps = 2e-6;
175
176 [J, mu, J_cost, J_diff, converged_i] =
    value_iteration_discounted_cost_finiteMDP(X_reg_i, X_abs_i, tau, C(:),
    gamma, eps, max_iter);
177 converged_i
178
179 figure
180 plot(J_diff)
181 xlabel( Iteration )
182 ylabel( ||V_{k+1} - V_{k}||_{\infty} )
183 title( Value Iteration Convergence )
184 grid on
185 yscale( log )
186
187 J_cost(:, :, end)
188 J_grid = reshape(J(:, end), [5, 5])
189 grid_i = zeros(5, 5);
190 grid_i(X_reg_i) = mu(:, end);
191 reshape(grid_i', [5,5])
192
193 % { Left , Right , Up , Down }
194 action_x = [0, 0, -1, 1];
195 action_y = [-1, 1, 0, 0];
196
197 figure
198 image(J_grid, 'CDataMapping', 'scaled'); hold on
199 action_base_x = zeros([1, numel(X_reg_i)]);
200 action_base_y = zeros([1, numel(X_reg_i)]);
201
202 action_head_x = zeros([1, numel(X_reg_i)]);
203 action_head_y = zeros([1, numel(X_reg_i)]);
204
205 for index = 1 : numel(X_reg_i)
206     i = X_reg_i(index);
207     x_i = mod(i - 1, 5) + 1;
208     y_i = floor((i - 1) / 5) + 1;
209
210     action_base_x(index) = x_i;
211     action_base_y(index) = y_i;
212
213     if mu(index, end) ~= 0

```

```

214     action_head_x(index) = action_x(mu(index, end));
215     action_head_y(index) = action_y(mu(index, end));
216 end
217 end
218 quiver(action_base_x, action_base_y, action_head_x, action_head_y, 0.3,
        filled , LineWidth = 1, Color= k ); hold off
219 colorbar()
220 title( Value Function )
221 subtitle(sprintf( with gamma = %.2f , gamma))
222
223 %% Algorithm 6
224 function [J, mu, J_cost, J_diff, converged_i] =
        value_iteration_discounted_cost_finiteMDP(X_reg_i, X_abs_i, tau, C,
        gamma, eps, max_iter)
225     % Assign J(X_abs) to C_exit, arbitrary values to X_reg
226     J = C;
227
228     for iter = 1 : max_iter
229         % Step value function
230         for x_i = X_reg_i
231             J_cost(:, :, iter) = C(x_i, :) + gamma * sum(J(:, iter) .*
                tau(:, x_i, :), 1);
232
233             [J(x_i, iter + 1), mu(:, iter + 1)] = min(J_cost(:, :, iter),
                [], 2);
234             J(X_abs_i, iter + 1) = C(X_abs_i); % maintain correct
                absorbing state cost
235         end
236
237         % Check stopping condition
238         J_diff(iter) = norm(J(:, iter + 1) - J(:, iter), inf);
239         if J_diff(iter) <= eps
240             converged_i = iter;
241             break;
242         end
243     end
244 end

```