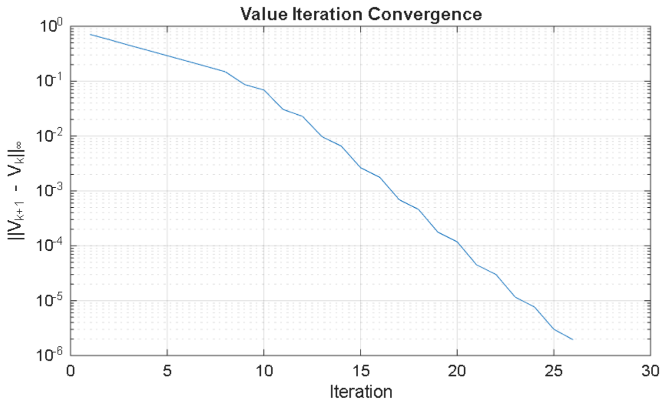
3.2.i)

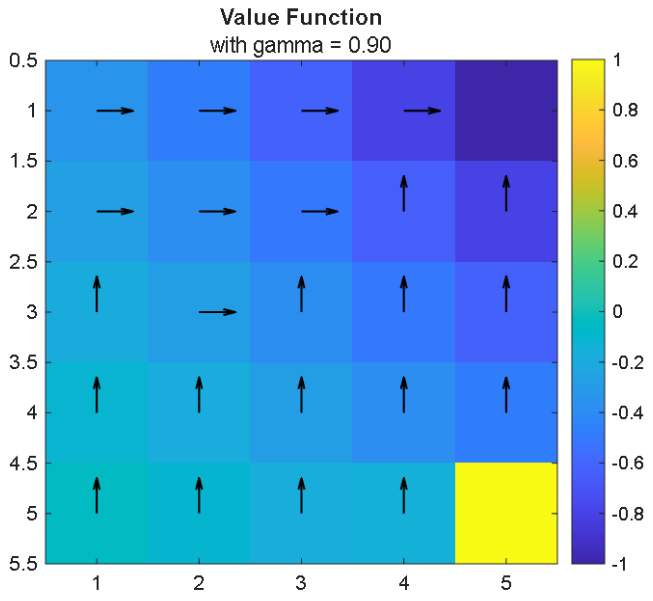
Using: Algorithm 6 compute the value function. Hint: You should obtain a value function as shown in Figure 3.3.
Solution:

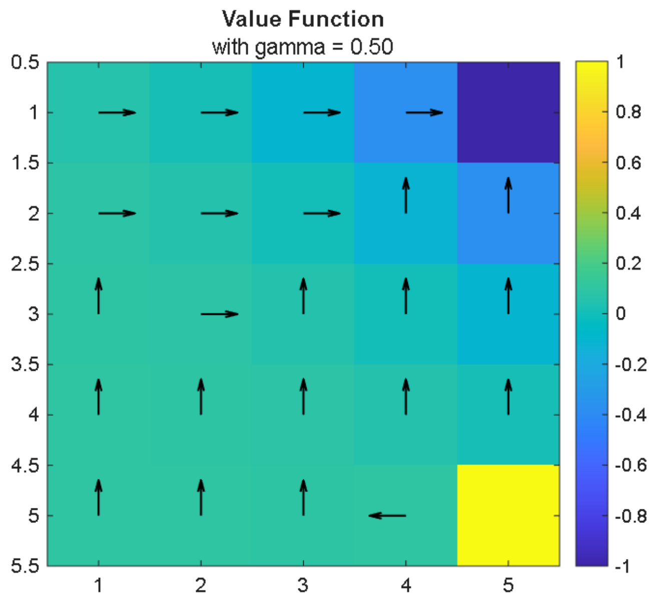
Figure 1: Convergence of value iteration algorithm for $\gamma = 0.9$



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 γ 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 γ it takes this risk at the reward of potentially getting to the goal faster.

Code Snippet 1: Q3.2 Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AAE590 Stochastic Control
% Machine Replacement Dynamic Programming
% Author: Travis Hastreiter
% Created On: 16 September, 2025
% Description: Solves machine replacement problem using dynamic
% programming.
% Most Recent Change: 16 September, 2025
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

C_pit = 1;
C_goal = -1;
C_step = 0.05;
p = 0.2;

C = C_step * ones([5, 5]);

% Create absorbing states
X_abs = zeros([5, 5]);
% Goal
goal_i = 5;
X_abs(goal_i) = 1;
C(goal_i) = -1;
% Pit
pit_i = 25;
X_abs(pit_i) = 1;
C(pit_i) = 1;

% Create regular states
X_reg = ~X_abs;

% Get regular and absorbing state indices
X_reg_i = find(X_reg);
X_abs_i = find(X_abs);

% Create U
U = 1 : 4; % { Left ,  Right ,  Up ,  Down }

% Create tau
tau = zeros(5 ^ 2, 5 ^ 2, 4);

% Absorbing state
tau(X_abs_i(1), X_abs_i(1), :) = 1;
tau(X_abs_i(2), X_abs_i(2), :) = 1;

%% Left
% Left is wall, not at corner
for i = 2 : 5
    tau(i, i, 1) = 1 - p;
end

% Left is wall, at corner
for i = [1, 5]
```

```
    tau(i, i, 1) = 1 - p / 2;
end

% Left isn't wall
for i = 6 : 5 * 5
    tau(i - 5, i, 1) = 1 - p;
end

% Up isn't wall
for i = reshape((2 : 5) + 5 * (0:4)', 1, [])
    tau(i - 1, i, 1) = p / 2;
end

% Up or down is wall, not at left corner
for i = reshape([1, 5] + 5 * (1:4)', 1, [])
    tau(i, i, 1) = p / 2;
end

% Down isn't wall
for i = reshape((1 : 4) + 5 * (0:4)', 1, [])
    tau(i + 1, i, 1) = p / 2;
end

%% Right
% Right is wall, not at corner
for i = 22 : 24
    tau(i, i, 2) = 1 - p;
end

% Right is wall, at corner
for i = [21, 25]
    tau(i, i, 2) = 1 - p / 2;
end

% Right isn't wall
for i = 1 : 5 * 4
    tau(i + 5, i, 2) = 1 - p;
end

% Up isn't wall
for i = reshape((2 : 5) + 5 * (0 : 4)', 1, [])
    tau(i - 1, i, 2) = p / 2;
end

% Up or down is wall, not at right corner
for i = reshape([1, 5] + 5 * (0 : 3)', 1, [])
    tau(i, i, 2) = p / 2;
end

% Down isn't wall
for i = reshape((1 : 4) + 5 * (0 : 4)', 1, [])
    tau(i + 1, i, 2) = p / 2;
end

```

```
%% Up
% Up is wall, not at corner
for i = 1 + 5 * (1 : 3)
    tau(i, i, 3) = 1 - p;
end

% Up is wall, at corner
for i = [1, 21]
    tau(i, i, 3) = 1 - p / 2;
end

% Up isn't wall
for i = reshape((2 : 5) + 5 * (0 : 4)', 1, [])
    tau(i - 1, i, 3) = 1 - p;
end

% Right isn't wall
for i = reshape((1 : 5) + 5 * (0 : 3)', 1, [])
    tau(i + 5, i, 3) = p / 2;
end

% Right or left is wall, not at upper corner
for i = reshape((2 : 5) + 5 * [0, 4]', 1, [])
    tau(i, i, 3) = p / 2;
end

% Left isn't wall
for i = reshape((1 : 5) + 5 * (1 : 4)', 1, [])
    tau(i - 5, i, 3) = p / 2;
end

%% Down
% Down is wall, not at corner
for i = 5 + 5 * (1 : 3)
    tau(i, i, 4) = 1 - p;
end

% Down is wall, at corner
for i = [5, 25]
    tau(i, i, 4) = 1 - p / 2;
end

% Down isn't wall
for i = reshape((1 : 4) + 5 * (0 : 4)', 1, [])
    tau(i + 1, i, 4) = 1 - p;
end

% Right isn't wall
for i = reshape((1 : 5) + 5 * (0 : 3)', 1, [])
    tau(i + 5, i, 4) = p / 2;
end

% Right or left is wall, not at lower corner
for i = reshape((1 : 4) + 5 * [0, 4]', 1, [])
```

```
    tau(i, i, 4) = p / 2;
end

% Left isn't wall
for i = reshape((1 : 5) + 5 * (1 : 4)', 1, [])
    tau(i - 5, i, 4) = p / 2;
end

%% Solve
% Algorithm parameters
gamma = 0.5;
max_iter = 100;
eps = 2e-6;

[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);
converged_i

figure
plot(J_diff)
xlabel( Iteration )
ylabel( ||V_{k+1} - V_{k}||_\infty )
title( Value Iteration Convergence )
grid on
yscale( log )

J_cost(:, :, end)
J_grid = reshape(J(:, end), [5, 5])
grid_i = zeros(5, 5);
grid_i(X_reg_i) = mu(:, end);
reshape(grid_i', [5,5])

 % { Left ,  Right ,  Up ,  Down }
action_x = [0, 0, -1, 1];
action_y = [-1, 1, 0, 0];

figure
image(J_grid','CDataMapping','scaled'); hold on
action_base_x = zeros([1, numel(X_reg_i)]);
action_base_y = zeros([1, numel(X_reg_i)]);

action_head_x = zeros([1, numel(X_reg_i)]);
action_head_y = zeros([1, numel(X_reg_i)]);

for index = 1 : numel(X_reg_i)
    i = X_reg_i(index);
    x_i = mod(i - 1, 5) + 1;
    y_i = floor((i - 1) / 5) + 1;

    action_base_x(index) = x_i;
    action_base_y(index) = y_i;

    if mu(index, end) ~= 0
```

```
        action_head_x(index) = action_x(mu(index, end));
        action_head_y(index) = action_y(mu(index, end));
    end
end
quiver(action_base_x, action_base_y, action_head_x, action_head_y, 0.3, filled , LineWidth = 1, Color = k ) ; hold off
colorbar()
title ( Value Function )
subtitle(sprintf ( with gamma = %.2f , gamma))

%% Algorithm 6
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)
    % Assign J(X_abs) to C_exit, arbitrary values to X_reg
    J = C;

    for iter = 1 : max_iter
        % Step value function
        for x_i = X_reg_i
            J_cost(:, :, iter) = C(x_i, :)' + gamma * sum(J(:, iter) .* tau(:, x_i, :), 1);

            [J(x_i, iter + 1), mu(:, iter + 1)] = min(J_cost(:, :, iter), [], 2);
            J(X_abs_i, iter + 1) = C(X_abs_i); % maintain correct absorbing state cost
        end

        % Check stopping condition
        J_diff(iter) = norm(J(:, iter + 1) - J(:, iter), inf);
        if J_diff(iter) <= eps
            converged_i = iter;
            break;
        end
    end
end
```