RS-HL-12: Multi User Code

Hongseok Kim

UT Austin Oden Institute

07/20/24

I. Load the Dataset

```
clear;clc;
% Load the Satellite Contat Dataset
addpath('~/Desktop/Redstone_Project/RS_HL/RS_HL_10_TV_MDP_Functions')
load('/workspace/RS_Dataset/RS_HL_3_dataset.mat')
```

II. Destination Setting and Time Index Vector Setting

```
time_index_vector = 100:130;
start_time_index = 100:120;
%
% destination_1 = 10;
% destination_2 = 12;
% destination_3 = 18;
% destination_4 = 27;
% destination_5 = 40;


number_of_agents = 20;
number_of_destinations = 4;
state_vector = 1:48;


start_state = state_vector(randi(numel(state_vector), 1, number_of_agents));
destination_values = randsample(state_vector,number_of_destinations);
destination_state = destination_values(randi(numel(destination_values), 1,
number_of_agents));
start_time = start_time_index(randi(numel(start_time_index), 1,
number_of_agents));
```

II.1 Run the MDP simulation for each destination

```

number_of_destination = length(destination_values);
```

```
for destination_index = 1:number_of_destination

    MDP.(['MDP', num2str(destination_values(destination_index))])
= runMDP(sat_to_sat_contact_3d_matrix,
time_index_vector,destination_values(destination_index));

end
```

```
simulation set up complete!
Policy: 1 -> Value Iteration: 419
Policy: 2 -> Value Iteration: 14
Policy: 3 -> Value Iteration: 9
Policy: 4 -> Value Iteration: 5
Policy: 5 -> Value Iteration: 5
Policy: 6 -> Value Iteration: 8
Policy: 7 -> Value Iteration: 10
Policy: 8 -> Value Iteration: 10
Policy: 9 -> Value Iteration: 1
simulation set up complete!
Policy: 1 -> Value Iteration: 413
Policy: 2 -> Value Iteration: 14
Policy: 3 -> Value Iteration: 5
Policy: 4 -> Value Iteration: 5
Policy: 5 -> Value Iteration: 6
Policy: 6 -> Value Iteration: 10
Policy: 7 -> Value Iteration: 5
Policy: 8 -> Value Iteration: 5
Policy: 9 -> Value Iteration: 1
simulation set up complete!
Policy: 1 -> Value Iteration: 411
Policy: 2 -> Value Iteration: 14
Policy: 3 -> Value Iteration: 8
Policy: 4 -> Value Iteration: 5
Policy: 5 -> Value Iteration: 5
Policy: 6 -> Value Iteration: 4
Policy: 7 -> Value Iteration: 8
Policy: 8 -> Value Iteration: 4
Policy: 9 -> Value Iteration: 1
simulation set up complete!
Policy: 1 -> Value Iteration: 416
Policy: 2 -> Value Iteration: 14
Policy: 3 -> Value Iteration: 7
Policy: 4 -> Value Iteration: 5
Policy: 5 -> Value Iteration: 5
Policy: 6 -> Value Iteration: 8
Policy: 7 -> Value Iteration: 3
```

```
% MDP.(['MDP', num2str(18)]) = runMDP(sat_to_sat_contact_3d_matrix,
time_index_vector,18);
% MDP.(['MDP', num2str(28)]) = runMDP(sat_to_sat_contact_3d_matrix,
time_index_vector,28);
% MDP.(['MDP', num2str(42)]) = runMDP(sat_to_sat_contact_3d_matrix,
time_index_vector,42);
```

III. Configure each Agent's Setting

```
agents_input = [start_time', start_state', destination_state'];

% a_start = 31;
% a_destination = 18;
% a_time = 100;
%
%
% b_start = 31;
% b_destination= 28;
% b_time = 100;
%
% c_start = 31;
% c_destination = 42;
% c_time = 100;
%
% d_start = 31;
% d_destination = 18;
% d_time = 100;
%
% e_start = 31;
% e_destination = 18;
% e_time = 118;
%
%
% f_start = 14;
% f_destination = 28;
% f_time = 115;
%
% g_start = 10;
% g_destination = 42;
% g_time = 110;
%
% agents_input = [a_time, a_start, a_destination;
%                 b_time, b_start, b_destination;
%                 c_time, c_start, c_destination;
%                 d_time, d_start, d_destination;
%                 e_time, e_start, e_destination;
%                 f_time, f_start, f_destination;
%                 g_time, g_start, g_destination];
%
```

IV. Configure the simulation structure setting

```
% Level 1: Initialize simulation structure
sim = struct();

for time_index = time_index_vector
    sim.(['time' num2str(time_index)]) = {};
end

% Level 2/3: Initialize Agent (Level 2) with States and Destination (Level 3)

number_of_agents = length(agents_input(:,1));

for agent_index = 1:number_of_agents
   sim.(['time' num2str(agents_input(agent_index,1))]).(['agent'
num2str(agent_index)]).('state') = agents_input(agent_index,2);
   sim.(['time' num2str(agents_input(agent_index,1))]).(['agent'
num2str(agent_index)]).('destination') = agents_input(agent_index,3);
end
```

V. Propagation of agents' state

```
collision_flag = false;

for time_index = time_index_vector

    % If there's no active agent, continue to next time step
    if isempty(sim.(['time' num2str(time_index)]))
       continue;
    end


    % Parse the number of active agents
    number_of_active_agents = length(fieldnames(sim.(['time'
num2str(time_index)])));

    % Make the status matrix represents current and next
    % [current_state, next_state, destination]
    status_matrix = zeros(3, number_of_active_agents);

    agents_list = fieldnames(sim.(['time' num2str(time_index)]));

    % Find the Next state from Current Agent-State
    for active_agent_index = 1:number_of_active_agents
        % Find the Current State and Destination of given agent
        current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
        destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');
```

```
        % Find the Next state from given MDP pi distribution
        pi_dist = MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).('policy_distribution');
        action_number = find(pi_dist(current_state,:) ~= 0);

        if length(action_number) > 1
        action_number = randsample(action_number,1);
        end

        next_state = MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('success').('next_state');
        reward = MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('success').('reward');
        state_value =  MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).(['state' num2str(current_state)]).('state_value');
        action_value  = MDP.(['MDP' num2str(destination)]).
(['time' num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('action_value');

        % Configure Proposed status matrix for the collision test
        status_matrix(1,active_agent_index) = current_state;
        status_matrix(2,active_agent_index) = next_state;
        status_matrix(3,active_agent_index) = destination;

        % Add State Value
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('state_value') = state_value;

        % Save the Original Reward and action value to prepare the update
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('original_action_number') = action_number;
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('original_action_value') = action_value;
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('original_reward') = reward;

        % Add the action number, reward, action value (may be changed)
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('action_number') = action_number;
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('action_value') = action_value;
        sim.(['time' num2str(time_index)]).(agents_list{active_agent_index}).
('reward') = reward;
    end


    % Break when time index reaches the end time
```

```
    if time_index == max(time_index_vector)
       break;
    end

    for active_agent_index = 1:number_of_active_agents

        % Don't update the agent already arrived to destination
        if status_matrix(1,active_agent_index) ==
status_matrix(3,active_agent_index)
            continue;
        end

        % Update the time+1 for next state
        sim.(['time' num2str(time_index+1)]).
(agents_list{active_agent_index}).('state') =
status_matrix(2,active_agent_index);
        sim.(['time' num2str(time_index+1)]).
(agents_list{active_agent_index}).('destination') =
status_matrix(3,active_agent_index);
    end


    % Avoid Collision Avoidance Algorithm


    % If next state Agent info is empty, continue
    if isempty(sim.(['time' num2str(time_index+1)]))
        continue;
    end

    % Parse the number of next state active agents
    next_state_number_of_active_agents = length(fieldnames(sim.(['time'
num2str(time_index+1)])));
    next_state_status_matrix = zeros(1, next_state_number_of_active_agents);
    next_state_agents_list = fieldnames(sim.(['time'
num2str(time_index+1)]));

    for next_state_agent_index = 1:next_state_number_of_active_agents
        next_state_status_matrix(next_state_agent_index)
= sim.(['time' num2str(time_index+1)]).
(next_state_agents_list{next_state_agent_index}).('state');
    end

    unique_elements = unique(next_state_status_matrix);

    % If there is no collision -> contintue
    if length(next_state_status_matrix) == length(unique_elements)
        continue;
    end
```

VI. If there exists Collision -> Activate Collision Avoidance Algorithm

```
    fprintf('collision occured at time index %d\n', time_index);
    % If there exist collision -> Start the infinite loop until the
    % problem resolved
    collision_flag = true;


    while collision_flag == true
     % Collect Action value vector of each agent's state

     action_value_struct = struct();
     vector_length_information = zeros(number_of_active_agents,1);

     for active_agent_index = 1:number_of_active_agents
        % Find the Current State and Destination of given agent
        current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
        destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');

        % Get the Action value Matrix from given state in given MDP
        action_value_vector = MDP.(['MDP' num2str(destination)]).
(['time' num2str(time_index)]).(['state' num2str(current_state)]).
('action_value_vector');
        vector_length = length(action_value_vector);
        action_value_matrix = [(1:vector_length)',action_value_vector];
        action_value_matrix = sortrows(action_value_matrix,2, 'descend');

        if vector_length > 4
            action_value_matrix = action_value_matrix(1:4,:);
        end


        action_value_struct.(agents_list{active_agent_index}).
('action_value_matrix') = action_value_matrix;
        vector_length_information(active_agent_index) =
length(action_value_matrix(:,1));
     end


     % Generate Cases For Each Action Value
     combination_matrix = [];

     for active_agent_index = 1:number_of_active_agents-1
        new_matrix = [];
```

```
        if active_agent_index == 1
           pre_matrix = (1:vector_length_information(active_agent_index))';
        else
           pre_matrix = combination_matrix;
        end

        length_of_pre_matrix = length(pre_matrix(:,1));

        for next_agent_index =
1:vector_length_information(active_agent_index+1)

        adding_vector = ones(length_of_pre_matrix,1)*next_agent_index;
        new_matrix_segment = [pre_matrix,adding_vector];
        new_matrix = [new_matrix;new_matrix_segment];

        end
        combination_matrix = new_matrix;
     end


     if number_of_active_agents == 1
        combination_matrix =  (1:vector_length_information(1))';
     end

     action_number_matrix =
zeros(length(combination_matrix(:,1)),number_of_active_agents);
     action_value_matrix =
zeros(length(combination_matrix(:,1)),number_of_active_agents);
     action_value_sum_vector = zeros(length(combination_matrix(:,1)),1);
     next_state_matrix =
zeros(length(combination_matrix(:,1)),number_of_active_agents);




     for case_index = 1:length(combination_matrix(:,1))
         for active_agent_index = 1:number_of_active_agents
             action_index =
combination_matrix(case_index,active_agent_index);

             action_value_info =  action_value_struct.
(agents_list{active_agent_index}).('action_value_matrix');

             action_number = action_value_info(action_index,1);
             action_value = action_value_info(action_index,2);

             action_number_matrix(case_index,active_agent_index) =
action_number;
             action_value_matrix(case_index,active_agent_index) =
action_value;
```

```
            current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
            destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');


            next_state_info =  MDP.(['MDP' num2str(destination)]).
(['time' num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('success').('next_state');
            next_state_matrix(case_index,active_agent_index) =
next_state_info;

        end
        action_value_sum_vector(case_index) =
sum(action_value_matrix(case_index,:));
    end

    case_evaluation_matrix =
[next_state_matrix,action_number_matrix,action_value_sum_vector];
    case_evaluation_matrix =
sortrows(case_evaluation_matrix,length(case_evaluation_matrix(1,:)),
'descend');

    for  case_index = 1:length(combination_matrix(:,1))

        next_state =
case_evaluation_matrix(case_index,1:number_of_active_agents);
        % Modify Status Matrix for corresponding next state vector
        status_matrix(2,:) = next_state;

        for active_agent_index = 1:number_of_active_agents

            % Don't update the agent already arrived to destination
            if status_matrix(1,active_agent_index) ==
status_matrix(3,active_agent_index)
                continue;
            end

            % Update states the time+1 for next state
            sim.(['time' num2str(time_index+1)]).
(agents_list{active_agent_index}).('state') =
status_matrix(2,active_agent_index);
        end

        % Parse the number of next state active agents
        next_state_number_of_active_agents = length(fieldnames(sim.(['time'
num2str(time_index+1)])));
```

```
        next_state_status_matrix = zeros(1,
next_state_number_of_active_agents);

        for next_state_agent_index = 1:next_state_number_of_active_agents
            next_state_status_matrix(next_state_agent_index)
= sim.(['time' num2str(time_index+1)]).
(next_state_agents_list{next_state_agent_index}).('state');
        end

        unique_elements = unique(next_state_status_matrix);

        % If there is no collision -> contintue
        if length(next_state_status_matrix) == length(unique_elements)
            selected_action_number_vector =
case_evaluation_matrix(case_index,number_of_active_agents+1:2*number_of_activ
e_agents);


           for active_agent_index = 1:number_of_active_agents

            action_number =
selected_action_number_vector(active_agent_index);
            % Parse the updated reward, state value and action value
            current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
            destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');
            reward = MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('success').('reward');



            % Modify the reward, state value and action value
            sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('reward') = reward;
            sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('action_number') = action_number;
            sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('action_value') = action_value;

           end


           fprintf('collision resolved at time index %d  at case index %d
\n', time_index, case_index);
           collision_flag = false;
           break;
        end
```

```
        end


        if collision_flag == true
            fprintf('we could not resolve the collision at time index %d\n',
time_index);
               break;
        end

    end

    if collision_flag == true

    fprintf('Simulation Terminated with fail');
    break;
    end
end
```

```
collision occured at time index 107
collision resolved at time index 107  at case index 14
collision occured at time index 108
collision resolved at time index 108  at case index 2
collision occured at time index 109
collision resolved at time index 109  at case index 2
collision occured at time index 112
collision resolved at time index 112  at case index 6
collision occured at time index 113
collision resolved at time index 113  at case index 5
```

```

if collision_flag == false
fprintf('Simulation Terminated with Success')
end
```

```
Simulation Terminated with Success
```

VI. result display

```
result_matrix = zeros(length(time_index_vector),number_of_agents);
reward_matrix =  zeros(length(time_index_vector),number_of_agents);
state_value_matrix = zeros(length(time_index_vector),number_of_agents);
action_value_matrix = zeros(length(time_index_vector),number_of_agents);
cumulative_reward_matrix = zeros(length(time_index_vector),number_of_agents);

for time_index = time_index_vector

    if isempty(sim.(['time' num2str(time_index)]))
     continue;
    end

    agents_list = fieldnames(sim.(['time' num2str(time_index)]));
    number_of_agents = length(agents_list);
```

```
    for agent_index = 1:number_of_agents
        agent_name = cell2mat(agents_list(agent_index));
        agent_no = regexp(agent_name, '\d+','match');
        agent_number = str2double(agent_no{1});

        result_matrix(time_index - min(time_index_vector) + 1,agent_number)
= sim.(['time' num2str(time_index)]).(agent_name).('state');
        reward_matrix(time_index - min(time_index_vector) + 1,agent_number)
=  sim.(['time' num2str(time_index)]).(agent_name).('reward');
        state_value_matrix(time_index - min(time_index_vector)
+ 1,agent_number) = sim.(['time' num2str(time_index)]).(agent_name).
('state_value');
        action_value_matrix(time_index - min(time_index_vector)
+ 1,agent_number) = sim.(['time' num2str(time_index)]).(agent_name).
('action_value');
        cumulative_reward_matrix(time_index - min(time_index_vector) +
1,agent_number) = sum(reward_matrix(1:time_index - min(time_index_vector) +
1,agent_number));

    end

end

result = [time_index_vector' , result_matrix]
```

```
result = 31×21
   100     0     0     0     0     0     0     0     0     0     0     0     0 ...
   101     0     0    28     0     0     0     0     0     0     0     0     0
   102     0     0    29     0     0     0     0     0     0     0     0     2
   103     0     0    30     0     0     0     0     0     0    27     0     3
   104     0     0    31     0     0     0     0     0     0    28     0     4
   105     0     0     0     0     0     0     0     0     0    29     0     5
   106     0     0     0     0     0     0     0     0     0    30     0     6
   107     0     0     0     0     0     0     0     0     0    31     0     7
   108     0     0     0     0     0     0     0     0     0     0    31     8
   109     0     0     0     0     0     0     0     0     0     0    32     9
     ⋮
```

```
reward = [time_index_vector', reward_matrix]
```

```
reward = 31×21
   100     0     0     0     0     0     0     0     0     0     0     0     0 ...
   101     0     0    -1     0     0     0     0     0     0     0     0     0
   102     0     0    -1     0     0     0     0     0     0     0     0    -1
   103     0     0    75     0     0     0     0     0     0    -1     0    -1
   104     0     0     0     0     0     0     0     0     0    -1     0    -1
   105     0     0     0     0     0     0     0     0     0    -1     0    -1
   106     0     0     0     0     0     0     0     0     0    75     0    -1
   107     0     0     0     0     0     0     0     0     0     0     0    -1
   108     0     0     0     0     0     0     0     0     0     0    -1    -1
   109     0     0     0     0     0     0     0     0     0     0    -1   100
     ⋮
```

```
cumulative_reward = [time_index_vector', cumulative_reward_matrix]
```

cumulative_reward = 31×21

100	0	0	0	0	0	0	0	0	0	0	0	0	···
101	0	0	-1	0	0	0	0	0	0	0	0	0	
102	0	0	-2	0	0	0	0	0	0	0	0	-1	
103	0	0	73	0	0	0	0	0	0	-1	0	-2	
104	0	0	73	0	0	0	0	0	0	-2	0	-3	
105	0	0	0	0	0	0	0	0	0	-3	0	-4	
106	0	0	0	0	0	0	0	0	0	72	0	-5	
107	0	0	0	0	0	0	0	0	0	72	0	-6	
108	0	0	0	0	0	0	0	0	0	0	-1	-7	
109	0	0	0	0	0	0	0	0	0	0	-2	93	
⋮													

```
state_value = [time_index_vector', state_value_matrix]
```

state_value = 31×21

100.0000	0	0	0	0	0	0	0	···
101.0000	0	0	50.5000	0	0	0	0	
102.0000	0	0	59.0000	0	0	0	0	
103.0000	0	0	67.5000	0	0	0	0	
104.0000	0	0	0	0	0	0	0	
105.0000	0	0	0	0	0	0	0	
106.0000	0	0	0	0	0	0	0	
107.0000	0	0	0	0	0	0	0	
108.0000	0	0	0	0	0	0	0	
109.0000	0	0	0	0	0	0	0	
⋮								

```
action_value = [time_index_vector', action_value_matrix]
```

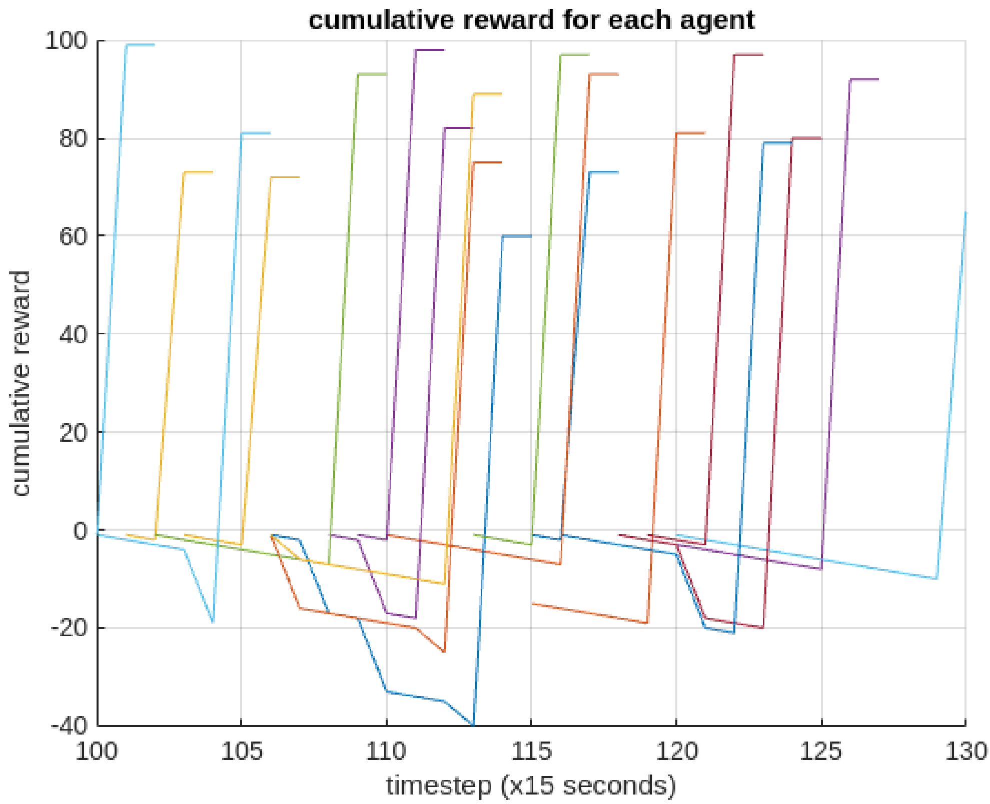
action_value = 31×21

100.0000	0	0	0	0	0	0	0	···
101.0000	0	0	50.5000	0	0	0	0	
102.0000	0	0	59.0000	0	0	0	0	
103.0000	0	0	67.5000	0	0	0	0	
104.0000	0	0	0	0	0	0	0	
105.0000	0	0	0	0	0	0	0	
106.0000	0	0	0	0	0	0	0	
107.0000	0	0	0	0	0	0	0	
108.0000	0	0	0	0	0	0	0	
109.0000	0	0	0	0	0	0	0	
⋮								

VII. Result Graph

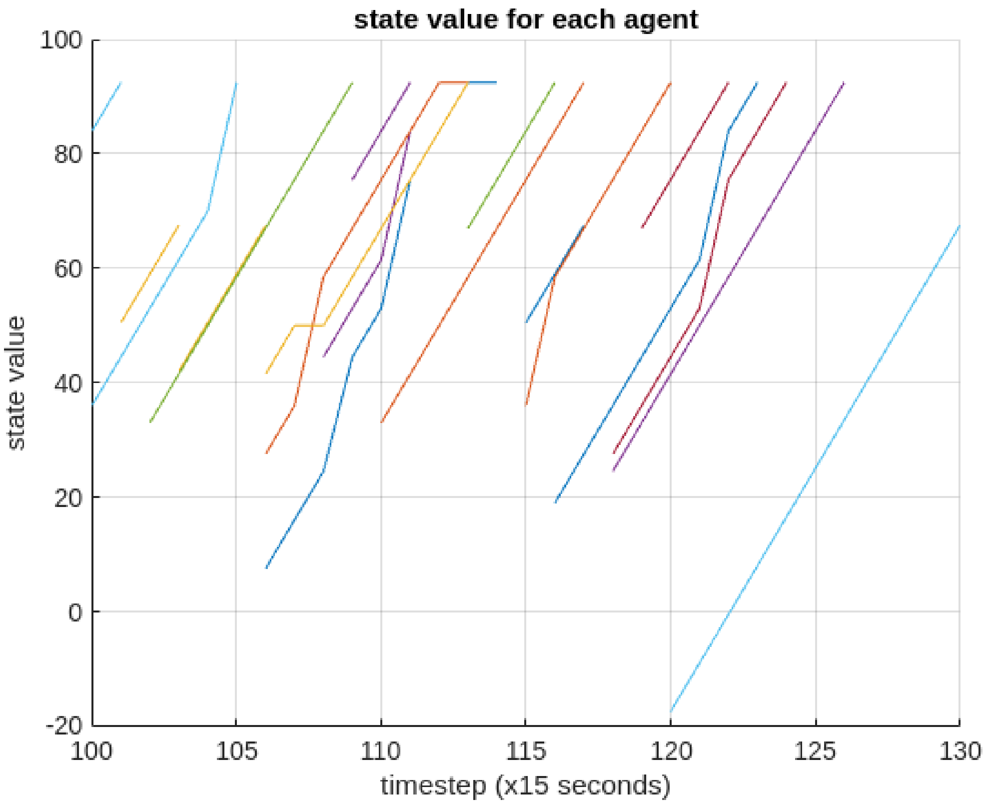
```
number_of_agents = length(agents_input(:,1));

figure;
hold on
for agent_index = 1:number_of_agents

    cumulative_reward_each_agent = [time_index_vector',
cumulative_reward_matrix(:,agent_index)];
```

```
    cumulative_reward_each_agent =
cumulative_reward_each_agent(cumulative_reward_each_agent(:,2) ~= 0, :);

   plot(cumulative_reward_each_agent(:,1),cumulative_reward_each_agent(:,2))

end
hold off
grid on
title('cumulative reward for each agent')
xlabel('timestep (x15 seconds)')
ylabel('cumulative reward')
```

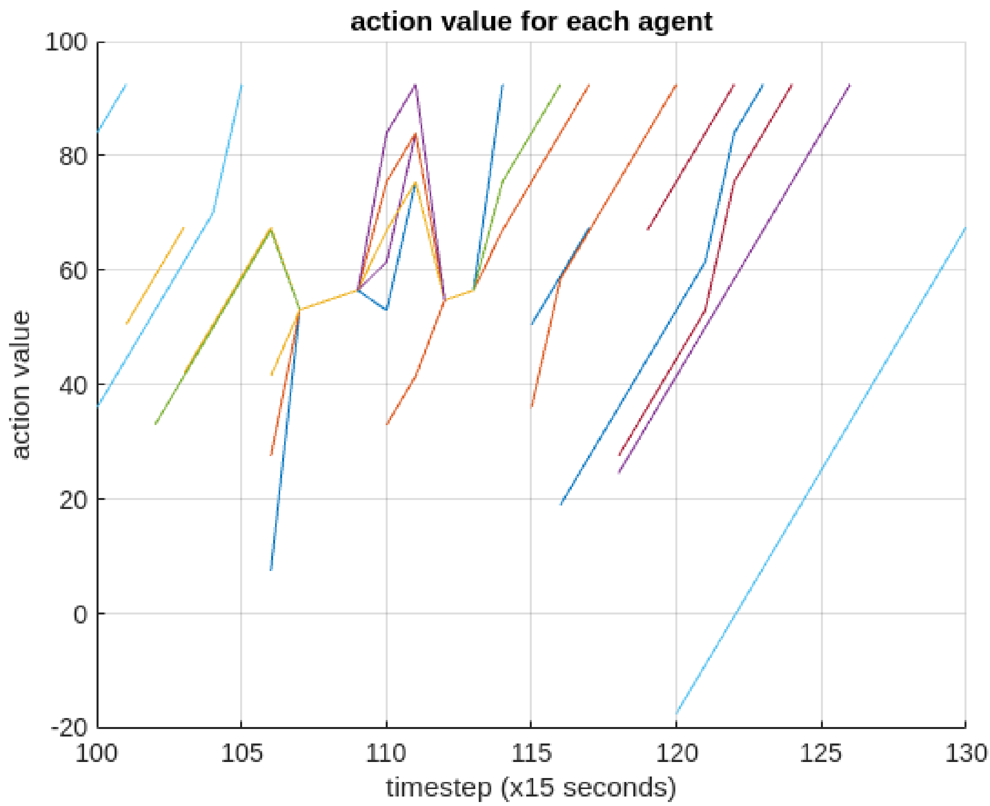



```
figure;
hold on
for agent_index = 1:number_of_agents

    state_value_each_agent = [time_index_vector',
state_value_matrix(:,agent_index)];
    state_value_each_agent =
state_value_each_agent(state_value_each_agent(:,2) ~= 0, :);

    plot(state_value_each_agent(:,1),state_value_each_agent(:,2))
end
hold off
grid on
title('state value for each agent')
```

```
xlabel('timestep (x15 seconds)')
ylabel('state value')
```




```
figure;
hold on
for agent_index = 1:number_of_agents

    action_value_each_agent = [time_index_vector',
action_value_matrix(:,agent_index)];
    action_value_each_agent =
action_value_each_agent(action_value_each_agent(:,2) ~= 0, :);
    plot(action_value_each_agent(:,1),action_value_each_agent(:,2))
end
hold off
grid on
title('action value for each agent')
xlabel('timestep (x15 seconds)')
ylabel('action value')
```


action value for each agent
action value
timestep (x15 seconds)
100
80
60
40
20
0
-20
100
105
110
115
120
125
130