1. Executive summary

We were tasked with developing and implementing a model that would control a thermostat to get the room to a target temperature. We were given the probabilities of the temperature changing at each state, and this varied with whether the heating was on or off. We solved this problem by using an MDP, and then using python to write a program that would solve the MDP and iterate through the costs of each state in order to obtain the optimal policy. At first it was difficult to figure out how to use the Bellman equation and dynamically program a method to return the optimal policy, but once we studied the slides and reviewed examples from class we were able to understand more fully how the equation worked and write the program to execute the expected steps. At the end, we were able to obtain the optimal policy that followed our estimated output with the costs being 5 for heating on and 1 for heating off. Our code was made to model this specific problem, but designed so that the values themselves were generic, and could be changed. The only parameters of the problem that can not be changed, are the increments and probabilities with which the temperature can change. Firstly our codes executes the value_iteration() method, and creates an array of length s, the number of states (in this case 19), with all indexes initialized to zero. The code then enteres a while loop where an empty deltas array is created, which will keep track of each states V value change during each value iteration. We then iterate through each state and action, calling the v_sigma method which uses the probability data given by the problem prompt, our specified costs (5 for ON, and 1 for OFF) in the v_sigma(), and the existing V(s) values to calculate the new potential V values. Between the two options for each state (ON or OFF), we choose the new V value with the lowest value, and add the difference between the old value, and the new value for that state to the deltas array. After the internal for loop runs, we iterate through the deltas array, and calculate the largest change across all of the states. If the largest change is below our convergence parameter (here specified as 0.1), the while loop will terminate and the policy will be computed, othewise, the while loop will repeat the process using the newly calculated V(s) values. Once the while loop terminates, indicating that the change in between iterations is negligible, we call the evaluate_policy() method which will use the V(s) values and a similar process to above to calculate the most efficient action for each state which in turn represents our optimal policy.

2. Objectives

The objective of this lab was to obtain the optimal policy for the thermostat regulation using the MDP model and implemented through dynamic programming. The temperature of the room could be any temperature between 16-25 in half degree increments. The temperature can fall by half a degree, stay the same, rise by half a degree, or rise by a whole degree. However, at some temperatures the probabilities of the temperature changing are different. Whether the heating is on or off also affects the probabilities. These all have to be taken into account when figuring out the equations in order to get the optimal policy. For the program, the cost values for each temperature would iterate until they converged, determined by a preset tolerance level. The final goal would be an output of the optimal policy where each temperature would either decide to have the heat on or off in order to reach the target temperature.

3. Formal description of the MDP model

States: {16, 16.5, 17, …, 24.5, 25} temperature of room

Actions: {h, c} heat on or off

Transition Function:

HEAT ON

	-.5	0	+.5	+1
16		.3	.5	.2
rest	.1	.2	.5	.2
24.5	.1	.2	.7	
25	.1	.9		

HEAT OFF

	-.5	0	+.5
16		.9	.1
rest	.7	.2	.1
25	.7	.3	

Cost:
Heating on: 5
Heating off: 1

4. Detailed cost model analysis

5. Optimal policy for a certain set of costs, chosen by the students.

{16.0: 'ON', 16.5: 'ON', 17.0: 'ON', 17.5: 'ON', 18.0: 'ON', 18.5: 'ON', 19.0: 'ON', 19.5: 'ON', 20.0: 'ON', 20.5: 'ON', 21.0: 'ON', 21.5: 'ON', 22.0: 'OFF', 22.5: 'OFF', 23.0: 'OFF', 23.5: 'OFF', 24.0: 'OFF', 24.5: 'OFF', 25.0: 'OFF'}

6. Project phases (design, implementation, testing, …)

First we drew out the MDP diagram and decided what the states, actions, and transition functions were. After that, we started creating the formal MDP model. At first, for our transition tables we listed all the states as temperatures of the room from 16-25 in .5 degree increments, but then we realized that each temperature can only do one of four actions: drop by .5 degrees, stay the same, raise by .5 degrees, or raise by 1 degree. The exceptions were the temperatures at 16, 24.5, and 25. This made it so that we could just create a four by four table instead. We made two tables, one for heating on and one for heating off as those affected the probabilities.

Next, we had to implement the problem into python and write a program that would solve the optimal MDP for us using dynamic programming. After figuring out the equation for the optimal policy, we created if statements for each temperature and had the program loop through until the values converged. The output is the optimal policy that decides whether the thermostat should turn the temperature on or off at each time interval in order to reach the target temperature using the cost values from the iteration. For testing, we knew that the optimal policy would be having the heat on when the temperature was below the target and turning it off if it was above or at the temperature. So we changed the initial on and off costs until the optimal policy that was output made sense.

7. Budget (financial estimate of how much the study and implementation would have cost if someone else had contracted it out)

The project took us around 20 hours to do, and estimating that software engineers in Spain are paid about 30 euros per hour, the total cost would be 600 euros to contract it out. Including actual hardware to test out the implementation might be around another 2-5 thousand euros, so the total cost would be about 2.6-5.6k euros to fully study and implement everything. Existing smart thermostats cost between 200-300 euros at market, so to utilize existing technology, we may want 10-20 thermostats to test in different environments. To create our own hardware, we would need to hire additional engineers to create hardware that we can test our own software on. Using the previous pay rate and an estimated 150 hours of work, this would represent 4500 euros towards development, and another 500 euros of material costs.

8. Conclusions (technical comments related to the development of the project and personal comments: difficulties, challenges, benefits, etc.).

At first the project seemed daunting, but after reading through and fully understanding what the project was asking for, it became easier to create a plan in order to implement the MDP. Creating the transition probability table was the first step, and then figuring out the equations to use was also a bit difficult. It needed a full understanding of the Bellman equation, so reviewing the slides and looking through examples was necessary to be able to apply the equation to this thermostat problem. After figuring out the equations to get the costs for each action at each temperature, we had to implement the loop and dynamic programming in order to get the final cost values in order to make decisions in the optimal policy. This also took some trial and error to figure out, but once we had the loops figured out and working all we had to do was play around with changing the costs until the optimal policy being output matched with the one that we expected. This lab was beneficial in creating a deeper understanding of how to solve a MDP problem, as well as actually implementing it into code to solve using dynamic programming. It took what we learned about in class and applied it to a real world problem that we were able to go through each step to solve by ourselves, using what we were taught throughout the semester.