## Question2：

I programmed in python3 to implement the MDP. As the formula in the lecture note:

We have a state space S.

For , action set A(x)

For , , we have immediate reward .

For , , we have transition to state s’ with probability .

For ：





And by value iteration,



Hence,



In my approach, I use the following convergence condition to accelerate the convergence,

.

And the representation of data is showed below.

State Set

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| New | Used1 | Used2 | Used3 | Used4 | Used5 | Used6 | Used7 | Used8 | Dead |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Action Set

|  |  |
| --- | --- |
| Used | Replace |
| 0 | 1 |

Transition Set

|  |  |
| --- | --- |
| Transition\_used | Transition\_replace |

Transition used, (-infinite denotes can’t be reach)

|  |  |  |  |
| --- | --- | --- | --- |
| state | 0 | 1-8 | 9 |
| (probability, next state) | (1, 1) | (0.1 \* i, i + 1), (1 – 0.1 \* i, i) | (negative infinite, 0) |

Transition replace

|  |  |  |
| --- | --- | --- |
| state | 0 | 1-9 |
| (probability, next state) | (negative infinite, 0) | (1, 0) |

Reward Set

|  |  |
| --- | --- |
| Reward\_used | Reward\_replace |

Reward used, (0 denotes can’t be reach and no reward)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 100 | 90 | 80 | 70 | 60 | 50 | 40 | 30 | 20 | 0 |

Reward replace, (0 denotes can’t be reach and no reward)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | -250 | -250 | -250 | -250 | -250 | -250 | -250 | -250 | -250 |

So when I get U by value iteration, I can easily use it to find the best action to do on the exactly state.

|  |  |  |
| --- | --- | --- |
|  | state |  |
| New | 0 | 800.5305499262429, |
| Used1 | 1 | 778.3673954680371, |
| Used2 | 2 | 643.2212348213058, |
| Used3 | 3 | 556.1225096942011, |
| Used4 | 4 | 502.8349428957191, |
| Used5 | 5 | 475.8449436861682, |
| Used6 | 6 | 470.4773889386281, |
| Used7 | 7 | 470.4773889386281, |
| Used8 | 8 | 470.4773889386281, |
| Dead | 9 | 470.4773889386281 |

1. **0 is used, 1 is replace.**

|  |  |  |
| --- | --- | --- |
|  | state | action |
| New | 0 | 0 |
| Used1 | 1 | 0 |
| Used2 | 2 | 0 |
| Used3 | 3 | 0 |
| Used4 | 4 | 0 |
| Used5 | 5 | 0 |
| Used6 | 6 | 1 |
| Used7 | 7 | 1 |
| Used8 | 8 | 1 |
| Dead | 9 | 1 |

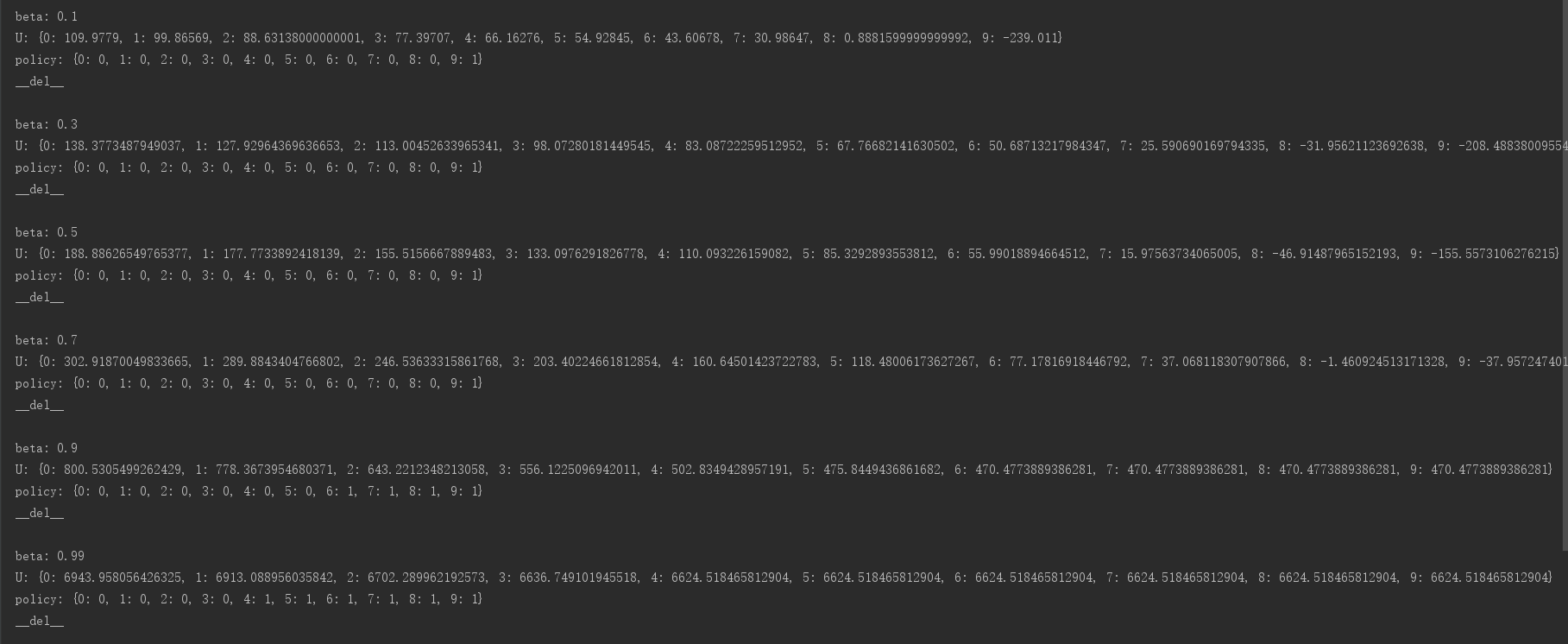
Assume we just let the cost to be integer, so I modify the code to the file mdp\_c.py to output the U of this problem.

From observation, I found that when the cost is 169, all the *U\*(state)* are slightly larger than the U I have got in a) with a delta = 0.8 approximately. And when the cost is 170, the U is smaller than the U in a).

So, I conclude that the highest price supposed to be 169 so that the used machine would be a rational choice.

I modify the code to mdp\_d.py

Just use different values of beta to run the program, so I get: when beta increases, the number of states best to *replace* is increases.

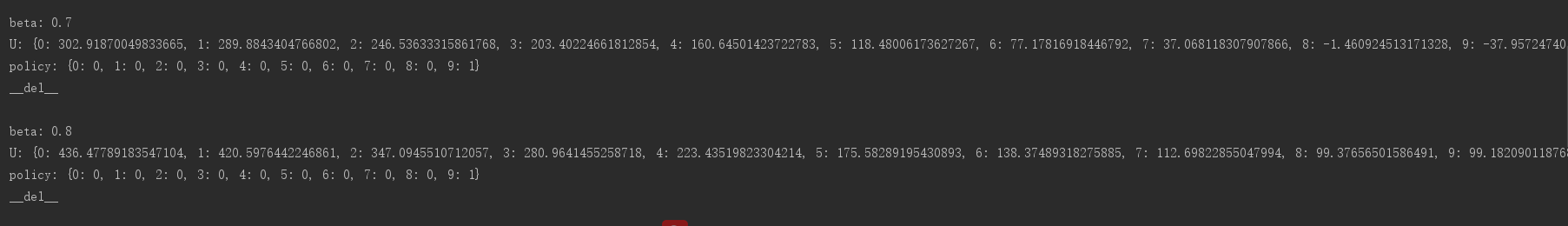


So, from the outcome I could assume that there is a policy,

|  |  |  |
| --- | --- | --- |
|  | state | action |
| New | 0 | Used |
| Used1 | 1 | Used |
| Used2 | 2 | Used |
| Used3 | 3 | Used |
| Used4 | 4 | Used |
| Used5 | 5 | Used |
| Used6 | 6 | Used |
| Used7 | 7 | Used |
| Used8 | 8 | Used |
| Dead | 9 | Replace |

That using this policy we can have the optimal until beta is 0.7. But for beta larger than 0.7, we may not have a optimal policy for all beta.

**Bonus)**



For beta = 0.8, I found all U is bigger than 0, and beta smaller than 0.7 there are always negative Us, so I think the long term discounted value ***x*** may be between 0.7 and 0.8. When it is small than ***x***, we will get net loss and otherwise we can get a net gain.