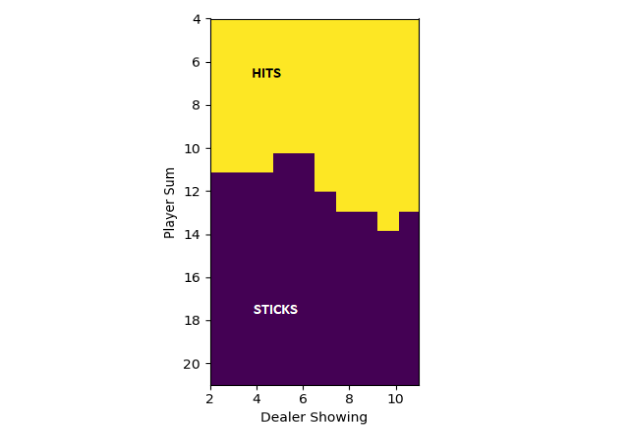
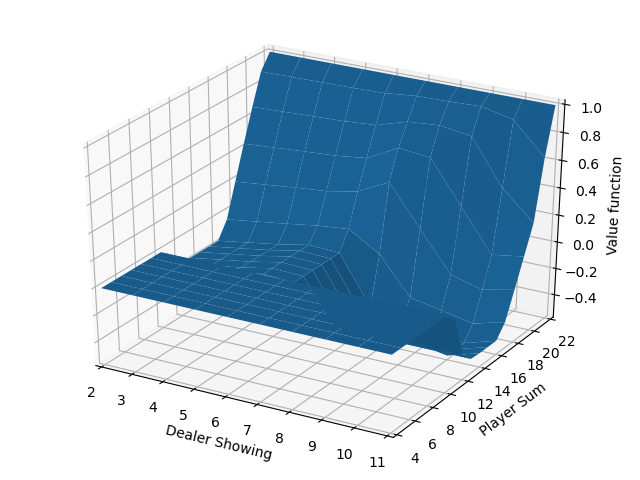
1. **Blackjack:**
2. The Markov decision process is defined as follows:  
   States space: each state represent the  where x is the sum of cards holds by the player and y is the exposed card of the dealer. After the dealer give cards we start from the appropriate state. The terminal state is where also the player and the dealer do not want to play anymore.The size of the state space is because there are 10 options for y (2-11) and 19 options for x (4-22) and another 10 states for natural win.Action space: for each state the possible actions are .  
   Transform function: if the player choose "sticks" the MDP transform stick with the current and this is the dealer turn.  
   but, if the player choose "hits" then the MDP transform to  where:  
   Rewards:the player gets reward only when the game ends: if the player starts the game when it is "neutral" and gets 1 if then loses and gets -1. For winning using hits gets 1 for a draw gets 0 and for loss gets   
   -1.  
   **The objective is to maximize the player's reward.  
   The bellman equation is:**
3. 

Code:  
**import** numpy **as** np

DEALER\_BUSTED **=** 22

PROBABILITY\_DICT **=** \

**{**

2**:** 1 **/** 13**,**

3**:** 1 **/** 13**,**

4**:** 1 **/** 13**,**

5**:** 1 **/** 13**,**

6**:** 1 **/** 13**,**

7**:** 1 **/** 13**,**

8**:** 1 **/** 13**,**

9**:** 1 **/** 13**,**

10**:** 4 **/** 13**,**

11**:** 1 **/** 13**,**

**}**

**def** \_\_calc**(**first\_card**,** dealer\_sum**,** prob**,** p**):**

**if** 17 **<=** dealer\_sum **<=** 21**:**

p**[**first\_card**][**dealer\_sum**]** **+=** prob

**return**

**if** dealer\_sum **>** 21**:**

p**[**first\_card**][**22**]** **+=** prob

**return**

**for** hit\_card**,** hit\_probability **in** PROBABILITY\_DICT**.**items**():**

\_\_calc**(**first\_card**,** dealer\_sum **+** hit\_card**,** prob **\*** hit\_probability**,** p**)**

**return**

**def** calc\_first\_card\_end\_sum\_probability**():**

first\_cards **=** **list(range(**2**,** 12**))**

end\_sum **=** **list(range(**17**,** 22**))**

p **=** np**.**zeros**(**shape**=(len(**first\_cards**)** **+** 2**,** **len(**end\_sum**)** **+** 18**))**

**for** first\_card **in** first\_cards**:**

\_\_calc**(**first\_card**,** first\_card**,** 1**,** p**)**

**return** p  
  
**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**from** dealer\_probabilty **import** calc\_first\_card\_end\_sum\_probability**,** PROBABILITY\_DICT**,** DEALER\_BUSTED

Nx **=** 22

Ny **=** 12

p **=** calc\_first\_card\_end\_sum\_probability**()**

r **=** np**.**zeros**(**shape**=(**Nx**,** Ny**))**

**for** y **in** **range(**2**,** Ny**):**

**for** x **in** **range(**4**,** Nx**):**

r**[**x**,** y**]** **=** p**[**y**][**DEALER\_BUSTED**]** **+** **sum(**p**[**y**,** 0**:**x**])** **-** **sum(**p**[**y**,** x **+** 1**:**DEALER\_BUSTED**])**

v\_f **=** np**.**zeros**(**shape**=(**Nx **+** 1**,** Ny**))**

actions **=** np**.**zeros**(**shape**=(**Nx **+** 1**,** Ny**))**

v\_f**[**Nx**][:]** **=** 1

v\_prev **=** v\_f

halt\_flag **=** **False**

**while** **not** halt\_flag**:**

**for** x **in** **range(**4**,** Nx**):**

**for** y **in** **range(**Ny**):**

sum\_for\_hits **=** 0

**for** hit\_card**,** hit\_probability **in** PROBABILITY\_DICT**.**items**():**

**if** x **+** hit\_card **<=** 21**:**

sum\_for\_hits **+=** hit\_probability **\*** v\_f**[**x **+** hit\_card**,** y**]**

**else:**

sum\_for\_hits **+=** hit\_probability **\*** **-**1

v\_f**[**x**,** y**]** **=** **max(**r**[**x**,** y**],** sum\_for\_hits**)**

actions**[**x**,** y**]** **=** np**.**argmax**([**r**[**x**,** y**],** sum\_for\_hits**])**

halt\_flag **=** np**.**array\_equal**(**v\_f**,** v\_prev**)**

v\_prev **=** v\_f

fig **=** plt**.**figure**()**

ax **=** plt**.**axes**(**projection**=**'3d'**)**

y **=** np**.**linspace**(**4**,** 22**,** 19**)**

x **=** np**.**linspace**(**2**,** 11**,** 10**)**

X, Y = np.meshgrid(x, y)

ax.plot\_surface(X=X, Y=Y, Z=v\_f[4:, 2:])

ax.set\_xlabel("Dealer Showing")

ax.set\_ylabel('Player Sum')

ax.set\_zlabel('Value function')

ax.set\_xlim([2, 11])

ax.set\_ylim([4, 22])

plt.show()

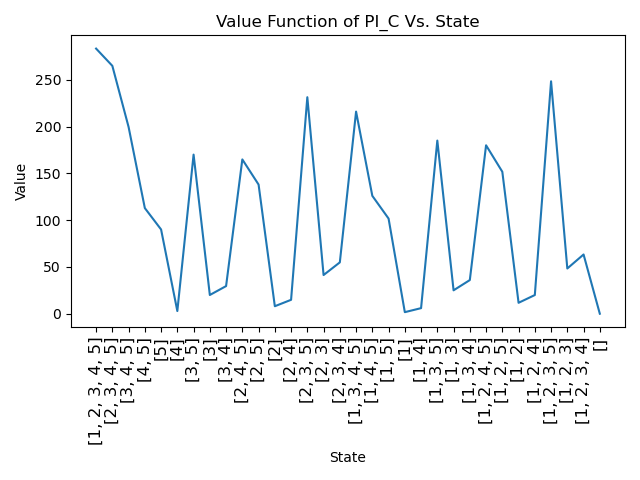
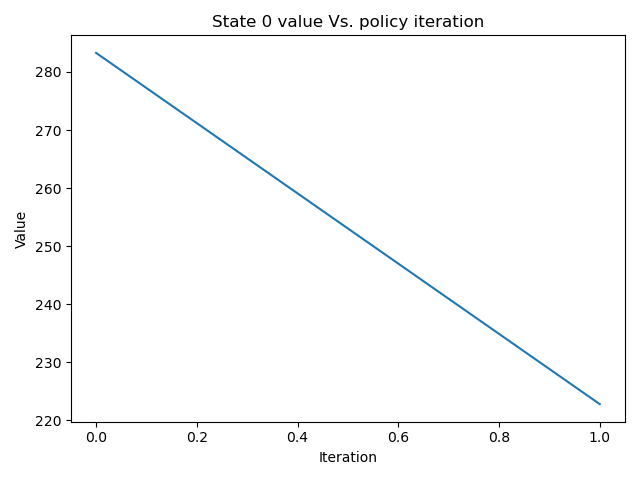
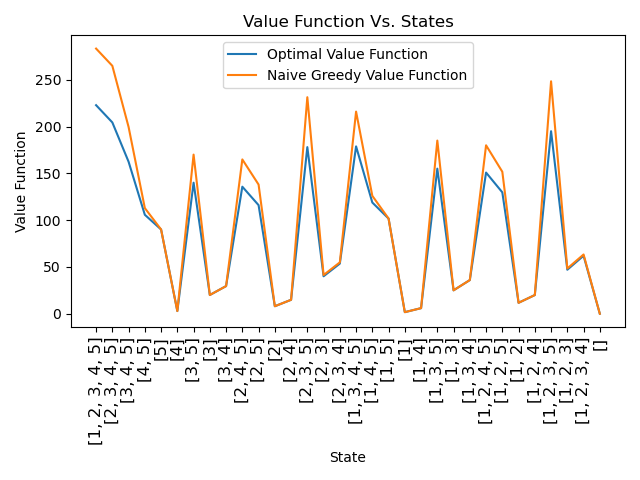
\_, ax = plt.subplots()

ax.imshow(actions[4:, 2:], extent=[2, 11, 21, 4])

ax.set\_xlabel("Dealer Showing")

ax.set\_ylabel('Player Sum')

plt.show()

1. **The**  **revisited:**
2. The number of states of the problem is:  because for each job there are two options or it already done or it is in the system.  
   The number different actions is one for each different job. Not all are available in all states, for example for the initial state, that includes all the jobs, all are available but after one of the jobs done only N-1 are available, and so on.
3. ***Code is available at the end of document.***
4. 
5.   
   After one iteration we get the optimal policy and one more step to make sure the policy is not changed from last one, means one iteration.
6.   
   As we can see from the graph for all the states the optimal policy values are smaller than the greedy one, for state with only on job left the value is equal.
7. ***Code is available at the end of document.***