2

RL - Assignment - 2

Theory part

CHELLA THIYAGARAJAN N - ME17B179

I pledge that I have not copied or given any unauthorized assistance on this assignment

Theory Questions

Problem 2 :

Given:

n - state discounted problem

Cost $= g(i,a)$

$\alpha \in (0,1)$

$P_{ij}(a) \rightarrow$ Transition probabilities

$$m_j = \min_{i=1,\dots,n} \; \min_{a \in A(i)} P_{ij}(a)$$

$$\tilde{P}_{ij}(a) = \frac{P_{ij}(a) - m_j}{1 - \sum_{k=1}^{n} m_k} \longrightarrow (1)$$

assume : $\sum_{k=1}^{n} m_k < 1$

a) for $\tilde{P}_{ij}(a)$ to be a transition probability

$$\sum_{j=1}^{n} \tilde{P}_{ij}(a) = 1$$

Apply summation on both sides of (1)

$$(1) \Rightarrow \sum_{j=1}^{n} \tilde{P}_{ij}(a) = \sum_{j=1}^{n} \left[\frac{P_{ij}(a) - m_j}{1 - \sum_{k=1}^{n} m_k} \right]$$

$$= \frac{\sum_{j=1}^{n} P_{ij}(a) - \sum_{j=1}^{n} m_j}{1 - \sum_{k=1}^{n} m_k} \longrightarrow (2)$$

$P_{ij}(a)$ is a transition probability, $\therefore \sum_{j=1}^{n} P_{ij}(a) = 1$

$$(2) \Rightarrow = \frac{1 - \sum_{j=1}^{n} m_j}{1 - \sum_{k=1}^{n} m_k}$$

$$= \frac{1 - \sum_{k=1}^{n} m_k}{1 - \sum_{k=1}^{n} m_k}, \text{ since } \sum_{j=1}^{n} m_j = \sum_{k=1}^{n} m_k$$

$$= 1$$

$$(1) \Rightarrow \sum_{j=1}^{n} \tilde{P}_{ij}(a) = 1 \quad //$$

Hence $\tilde{P}_{ij}$ are transition probabilities

Scanned with CamScanner

b) discount factor $= \alpha\left[1-\sum_{j=1}^{n} m_j\right]$

Show that $J^* = \tilde{J} + \dfrac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha}\, e$

e is n-vector of ones

$J \longrightarrow J^*$

$\bar{J} \longrightarrow \tilde{J}$

$J^* = TJ^*, \quad \tilde{J} = T\tilde{J}$

$TJ^*(i) = \sum_{j=1}^{n} P_{ij}\left[g(i,a) + \alpha J(j)\right]$

$= g(i,a) + \alpha \sum_{j=1}^{n} P_{ij} \cdot J^*(j) \longrightarrow (1)$

$T\tilde{J}(i) = g(i,a) + \alpha\left[1-\sum_{j=1}^{n} m_j\right] \sum_{j=1}^{n} \tilde{P}_{ij} \cdot \tilde{J}(j)$

$= g(i,a) + \alpha\left[1-\sum_{j=1}^{n} m_j\right] \sum_{j=1}^{n} \left[\dfrac{P_{ij}(a) - m_j}{1-\sum_{k=1}^{n} m_k}\right] \tilde{J}(j)$

$= g(i,a) + \alpha \sum_{j=1}^{n} P_{ij} \tilde{J}(j) - \alpha \sum_{j=1}^{n} m_j \tilde{J}(j) \longrightarrow (2)$

b) Consider the term $\dfrac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} \longrightarrow (3)$

(3) is a constant as $\tilde{J}$ is the optimal value and m_j is constant for every j

add (3) both sides of (2)

$$T\tilde{J}(i) + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} = g(i,a) + \alpha \sum_{j=1}^{n} P_{ij} \tilde{J}(j) - \alpha \sum_{j=1}^{n} m_j \tilde{J}(j) + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha}$$

$$\tilde{J}(i) + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} = g(i,a) + \alpha \sum_{j=1}^{n} P_{ij} \tilde{J}(j) + \frac{\alpha^2 \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha}$$

$$= g(i,a) + \alpha \left[\sum_{j=1}^{n} P_{ij} \tilde{J}(j) + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} \right]$$

we know $\sum_{j=1}^{n} P_{ij} = 1$ using this

$$= g(i,a) + \alpha \left[\sum_{j=1}^{n} P_{ij} \tilde{J}(j) + \frac{\alpha \sum_{j=1}^{n} P_{ij} \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} \right]$$

since (3) is a constant

$$\tilde{J}(i) + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} = g(i,a) + \alpha \sum_{j=1}^{n} P_{ij} \left[\tilde{J}(j) + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} \right] \longrightarrow (4)$$

④ is similar to ①

~~where $\tilde{J}^*(j) = \tilde{J}(j) + \alpha$~~

where $J^*(i) = \tilde{J}(i) + \dfrac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha}$

for all $i \in [1, 2, \dots, n]$

$$J^* = \tilde{J} + \frac{\alpha \sum_{j=1}^{n} m_j \tilde{J}(j)}{1-\alpha} e$$

2

1)

$$f(x_1) \le f(x_2) \quad , \quad x_1 \le x_2$$

$$ce \le f(x') - x' \le de \longrightarrow ⊛$$

(i) $$-|c| e \le f(x') - x' \Leftarrow ⊛$$

$$\frac{-|c| e \cdot (1-\alpha)}{(1-\alpha)} \le f(x') - x'$$

$$\frac{-|c| e}{1-\alpha} + \frac{\alpha |c| e}{1-\alpha} \le f(x') - x'$$

$$x' - \frac{|c| e}{1-\alpha} \le f(x') - \frac{\alpha |c| e}{1-\alpha}$$

$\longrightarrow$ ①

(ii) $\circledast \Rightarrow f(x') - x' \le |d|e$

$$f(x') - x' \le \frac{|d|e\,(1-\alpha)}{(1-\alpha)}$$

$$f(x') - x' \le \frac{|d|e}{1-\alpha} - \frac{\alpha|d|e}{1-\alpha}$$

$$f(x') + \frac{\alpha|d|e}{1-\alpha} \le \frac{|d|e}{1-\alpha} + x' \longrightarrow (2)$$

(iii) $$\|f_{k+m} - f_k\|_\xi \le \frac{\alpha^k}{1-\alpha}\|f_1 - f_0\|_\xi$$

$\hookrightarrow$ formula of Contraction mapping

Case 1

Let $k=0$, $m \to \infty$

$$\|f_m - f_0\|_\xi \le \frac{\alpha^0}{1-\alpha}\|f_1 - f_0\|_\xi$$

Let $f_0 = x'$

$$\|f^m(x') - x'\|_\xi \le \frac{1}{1-\alpha}\|f(x') - x'\|_\xi$$

as $m \to \infty$ $f^m(x') \to x^*$

$$\|x^* - x'\|_\xi \le \frac{1}{1-\alpha}\|f(x') - x'\|_\xi$$

$$-\frac{1}{1-\alpha}\|f(x)-x'\| \le x^* - x' \le \frac{1}{1-\alpha}\|f(x') - x'\|_\xi$$

$$\frac{-1}{1-\alpha} \| f(x') - x' \|_\xi \le x^* - x' \le \frac{1}{1-\alpha} \| f(x') - x' \|_\xi$$

from (*)

$$- \frac{|c|e}{1-\alpha} \le x^* - x' \le \frac{|d|e}{1-\alpha}$$

$$\frac{-|c|e}{1-\alpha} + x' \le x^* \le \frac{|d|e}{1-\alpha} + x' \quad \to (3)$$

Case 2

Let $k = 1$, $m \to \infty$, which means $m+1 \to \infty$

and $f^{m+1}(x') \to x^*$, Let $f_0 = x'$

$$\| f_{k+m} - f_k \|_\xi \le \frac{\alpha^k}{1-\alpha} \| f_1 - f_0 \|_\xi$$

$$\| f^{m+1}(x') - f(x') \|_\xi \le \frac{\alpha}{1-\alpha} \| f(x') - \cancel{f} x' \|_\xi$$

$$\| x^* - f(x') \|_\xi \le \frac{\alpha}{1-\alpha} \| f(x') - x' \|_\xi$$

$$\frac{-\alpha}{1-\alpha} \| f(x') - x' \|_\xi \le x^* - f(x') \le \frac{\alpha}{1-\alpha} \| f(x') - x' \|_\xi$$

from (*)

$$- \frac{\alpha |c|e}{1-\alpha} \le x^* - f(x') \le \frac{\alpha |d|e}{1-\alpha}$$

$$f(x') - \frac{\alpha |c|e}{1-\alpha} \le x^* \le f(x') + \frac{\alpha |d|e}{1-\alpha} \quad \to (4)$$

Scanned with CamScanner

from ①, ②, ③ & ④

we can say that

$$x' - \left[\frac{|c|e}{1-\alpha}\right] \le f(x') - \left[\frac{|c|\alpha e}{1-\alpha}\right] \le x^*$$

$$\le f(x') + \left[\frac{|d|\alpha e}{1-\alpha}\right] \le x' + \left[\frac{|d| e}{1-\alpha}\right]$$

Hence Proved

---

3). a) At state x_k and at time instant k

$$P(H) = \beta, \quad P(T) = 1-\beta$$

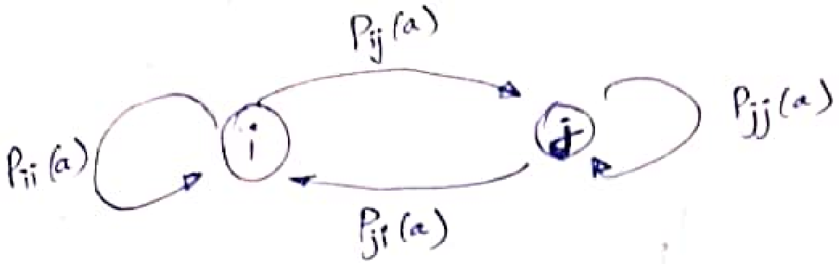
if Heads : $x_k \to x_{k+1}$

else Tails : $x_k \to T$

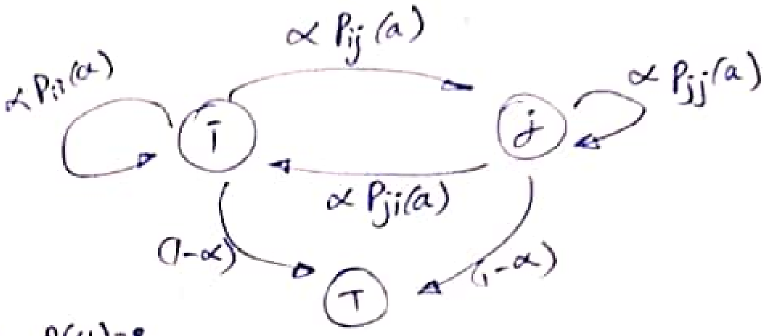
Originial discounted MDP :

discount factor $= \alpha$

Idea : In the S.S.P w.p α pick a next state according to transition probabilities of discounted MDP. & w.p $(1-\alpha)$ move to "T" eg.,



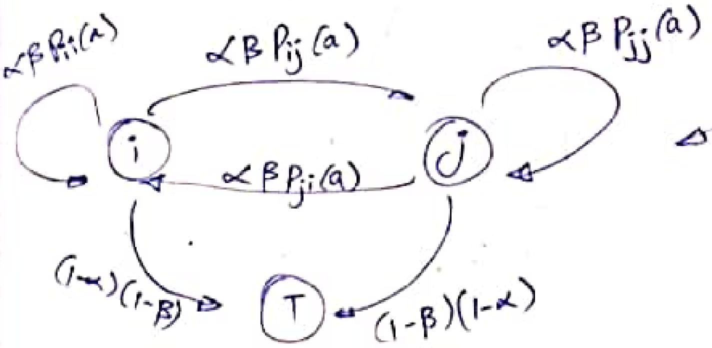
discounted - MDP to equivalent SSP



if Head : $x_k \to x_{k+1}$, $P(H)=\beta$

else : $x_k \to T$, $P(T)=1-\beta$

New Varient of equivalent SSP



↳ This is our new Varient discounted MDP

- Original Discounted MDP

$$\hookrightarrow J(x) = \max_{a \in A} \sum P(x,a,x')\left[g(x,a,x') + \alpha J(x')\right]$$

Scanned with CamScanner

New Variant of Original MDP

$$\hookrightarrow J(x) = \max_{a \in A} \sum \beta \cdot P(x,a,x')\left[g(x,a,x') + \alpha J(x')\right] + (1-\beta)\left[\cancel{J(T)}^{0}\right]$$

$$= \max_{a \in A} \sum P(x,a,x')\left[\beta g(x,a,x') + \alpha\beta J(x')\right]$$ due to cost-free & self absorbing property

The Cost function of the new Variant of the MDP

is $\beta \cdot g(x,a,x')$

The discount factor of this new Variant of MDP

is $\alpha\beta$

Let the New Cost function $\beta \cdot g(x,a,x') = h(x,a,x')$

Let the New discount factor $\alpha\beta = \gamma$

then:

$$J(x) = \max_{a \in A} \sum P(x,a,x')\left[h(x,a,x') + \gamma J(x')\right]$$

The above equation is a Viable equation of a discounted MDP problem, Therefore The New Variant Can also be described as discounted MDP framework

b) As proved in section (a), the discount factor of the new variant of the MDP is $\alpha\beta$.

if the discount factor $\alpha = 1$

then cost function of the new variant $h(x,a,x')$ remains the same.

the discount factor of the new variant $\alpha\beta$ becomes β.

$\therefore$ now

$$J(x) = \max_{a \in A} \sum P(x,a,x') \left[h(x,a,x') + \beta J(x') \right]$$

The above equation is a viable equation of a discounted MDP problem, Therefore the MDP variant is also continued to be a discounted type, with discount factor β.

Scanned with CamScanner

5. ~~State~~ Type 1 or Type 2

Type 1 ↓ work for salary

Type 2 ↳ works for passion

Single employee probabilities :

$$P(1, \text{recieve}, 2) = 0.75$$

$$P(1, \text{no recieve}, 1) = 0.75$$

$$P(2, \text{recieve}, 2) = 0.8$$

$$P(2, \text{no recieve}, 2) = 0.4$$

Single employee rewards :

$$g(1, \text{recieve}) = -5000 + 7500$$

$$g(1, \text{no recieve}) = -5000 + 2500$$

$$g(2, \text{recieve}) = -5000 + 20000$$

$$g(2, \text{no recieve}) = -5000 + 10000$$

$$J(i) = \max_{a \in A} \sum P(i,a,j)\left[g(i,a) + \alpha J(j)\right]$$

here $\alpha = 0.9$

There are 100 employees in our Company

actions : ['a1', 'a2'] → 'a2' incentive for type II

↳ 'a1' incentive for type I

Let x be the number of total Type I employees.

Let's define states as numbers of employees in Type I at each stage.

$$x \in [0, 1, 2, \ldots, 100]$$

Rewards at each stage:

$$g[\underbrace{x}_{\text{state}}, \underbrace{'a_1'}_{\text{action}}] = [-5000 + 7500]x + 10000[100-x]$$

$$g[\underbrace{x}_{\text{state}}, \underbrace{'a_2'}_{\text{action}}] = [100-x][-5000+20000] + 2500x$$

Probabilities follow binomial distribution for 100 employees:

$$P(x_i, \cancel{e}, \cancel{a_1}\ 'a_1', x_{i+1})$$

$$= \sum_{k=\max(0,(x_{i+1}+x_i-100))}^{\min(x_i, x_{i+1})} \left[{}^{x_i}C_k (0.25)^k (0.75)^{(x_i-k)} \right] \left[{}^{100-x_i}C_{x_{i+1}-k} (0.6)^{x_{i+1}-k} (0.4)^{100+k-x_i-x_{i+1}} \right]$$

$$P(x_i, 'a_2', x_{i+1})$$

$$= \sum_{k=\max(0,(x_{i+1}+x_i-100))}^{\min(x_i, x_{i+1})} \left[{}^{x_i}C_k (0.75)^k (0.25)^{(x_i-k)} \right] \left[{}^{100-x_i}C_{x_{i+1}-k} (0.2)^{x_{i+1}-k} (0.4)^{100+k-x_i-x_{i+1}} \right]$$

To Calculate Values of discounted MDP

$$J^*(i) = \max_{a \in A} E\left[\sum_{k=0}^{\infty} (0.9)^k g(x,a) \mid x_0 = 0.5\right]$$

To Calculate policy

~~$\pi^*(i) = E[\;]$~~

$$\pi^*(i) = \underset{a \in A}{\text{argmax}}\, E\left[g(i,a) + J(i+1)\cdot 0.9\right]$$

b) Code and results are attached as a pdf file

Policy Iteration :

$$\text{Optimal policy} : \pi^* = \begin{cases} 'a_2' , & x \leq 55 \\ 'a_1' , & x > 55 \end{cases}$$

c) Code and results are attached as a pdf file

4. Finite Horizon MDP:

Let there be n stages totally

Values : $J_i(s)$

Policy : $\pi_i(s)$

where i represents stage

Policy Iteration algorithm:

Initialize : initialize random policy for all states and stages

Initialize : $J_i(S) = 0$, $\forall s \in S$, $i \in [0, 1, \dots, n]$

```
Repeat
   Repeat
      δ = 0
      For i from (range(n,0,-1)):
         For each s ∈ S:
            j = J_i(s)
            if (i == n):
               J_i(s) = g_n(s)
            else:
               J_i(s) = Σ_{s'∈S} P_ss'(π_i(s)) [ g_i(s, π_i(s), s') + J_{i+1}(s') ]
            δ = max(δ, |j - J_i(s)|)
   until (δ < 1e-8)
   Done = 1
   For i from (range(n, 0, -1)):
      For each s ∈ S
         b = π_i(s)
         π_i(s) = argmax_{a∈A} Σ_{s'∈S} P_ss'(a) [ g_i(s, a, s') + J_{i+1}(s') ]
         if (b != π_i(s)):
            Done = 0
```

Last line of Code

```
}
until (Done == 1)
```

Chella Thiyagarajan N

ME17B179 5th Question part b and c

```
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm.notebook import tqdm
```

```
import operator as op
from functools import reduce

def comb(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return numer // denom
```

```
def get_probab(state, action, new_state):
  probab = 0
  for k in range(max(0, (state+new_state)-100), min(state, new_state)+1):
    if action == 'a1':
      probab += (comb(state, k)*((0.25)**k)*(0.75**(state-k))) * \
                  (comb(100-state, new_state-k)*((0.6)**(new_state-k))*(0.4**((100-state)-(new_state-k))))
    else:
      probab += (comb(state, k)*((0.75)**k)*(0.25**(state-k))) * \
                  (comb(100-state, new_state-k)*((0.2)**(new_state-k))*(0.8**((100-state)-(new_state-k))))
  return probab
```

```
def get_reward(state, action):
  if action == 'a1':
    reward = (-5000+7500)*state + 10000*(100-state)
  else:
    reward = (-5000+20000)*(100-state) + 2500*state
  return reward
```

```
# Initialization
values = [0]*101
actions = ['a1', 'a2']
greedy_policy = dict()
for state in range(0, 101):
  reward = []
  for action in actions:
    reward.append(get_reward(state, action))
  greedy_policy[state] = actions[np.argmax(np.array(reward))]
policy = greedy_policy.copy()
gamma = 0.9
```

```
%%time
# Policy Iteration

all_values = []
not_done = True
while(not_done):
  vi_not_done = True
  while(vi_not_done):
    delta = 0
    for state in range(0, 101):
      j = values[state]

      value = 0
      for new_state in range(0, 101):
        value += get_probab(state, policy[state], new_state)*(get_reward(state, policy[state]) + gamma*values[new_state])
      values[state] = value

      delta = max(delta, abs(j - values[state]))
    if(delta < 1e-8):
      vi_not_done = False
  not_done = False
  for state in range(0, 101):
    b = policy[state]

    action_values = []
    for action in actions:
      reward = 0
      for new_state in range(0, 101):
        reward += get_probab(state, action, new_state)*(get_reward(state, action) + gamma*values[new_state])
      action_values.append(reward)
    policy[state] = actions[np.argmax(np.array(action_values))]

    if(b != policy[state]):
      not_done = True
  all_values.append(values.copy())
```

```
CPU times: user 6min 51s, sys: 832 ms, total: 6min 51s
Wall time: 6min 53s
```

```
# Policy That we got from Policy Iteration
policy
```

```
42: 'a2',
43: 'a2',
44: 'a2',
45: 'a2',
46: 'a2',
47: 'a2',
48: 'a2',
49: 'a2',
50: 'a2',
51: 'a2',
52: 'a2',
53: 'a2',
54: 'a2',
55: 'a2',
56: 'a1',
57: 'a1',
58: 'a1',
59: 'a1',
60: 'a1',
61: 'a1',
62: 'a1',
63: 'a1',
64: 'a1',
65: 'a1',
66: 'a1',
67: 'a1',
68: 'a1',
69: 'a1',
70: 'a1',
71: 'a1',
72: 'a1',
73: 'a1',
74: 'a1',
75: 'a1',
76: 'a1',
77: 'a1',
78: 'a1',
79: 'a1',
80: 'a1',
81: 'a1',
82: 'a1',
83: 'a1',
84: 'a1',
85: 'a1',
86: 'a1',
87: 'a1',
88: 'a1',
89: 'a1',
90: 'a1',
91: 'a1',
92: 'a1',
93: 'a1',
94: 'a1',
95: 'a1',
96: 'a1',
97: 'a1',
98: 'a1',
99: 'a1',
100: 'a1'}
```

```
# Initialization
values = [0]*101
actions = ['a1', 'a2']
policy = {x:0 for x in range(0,101)}
```

```
%%time
# Value Iteration

all_values = []
not_done = True
while(not_done):
  delta = 0
  H = dict()
  for state in range(0,101):

    action_values = []
    for action in actions:
      reward = 0
      for new_state in range(0, 101):
        reward += get_probab(state, action, new_state)*(get_reward(state, action) + gamma*values[new_state])
      action_values.append(reward)
    H[state] = np.max(np.array(action_values))
    policy[state] = actions[np.argmax(np.array(action_values))]

    delta = max(delta, abs(values[state] - H[state]))
  for state in range(0, 101):
    values[state] = H[state]
  if(delta < 0.1):
    not_done = False
  all_values.append(values.copy())
```

```
CPU times: user 6min 30s, sys: 752 ms, total: 6min 30s
Wall time: 6min 31s
```

```
# Policy That we got from Value Iteration
policy
```

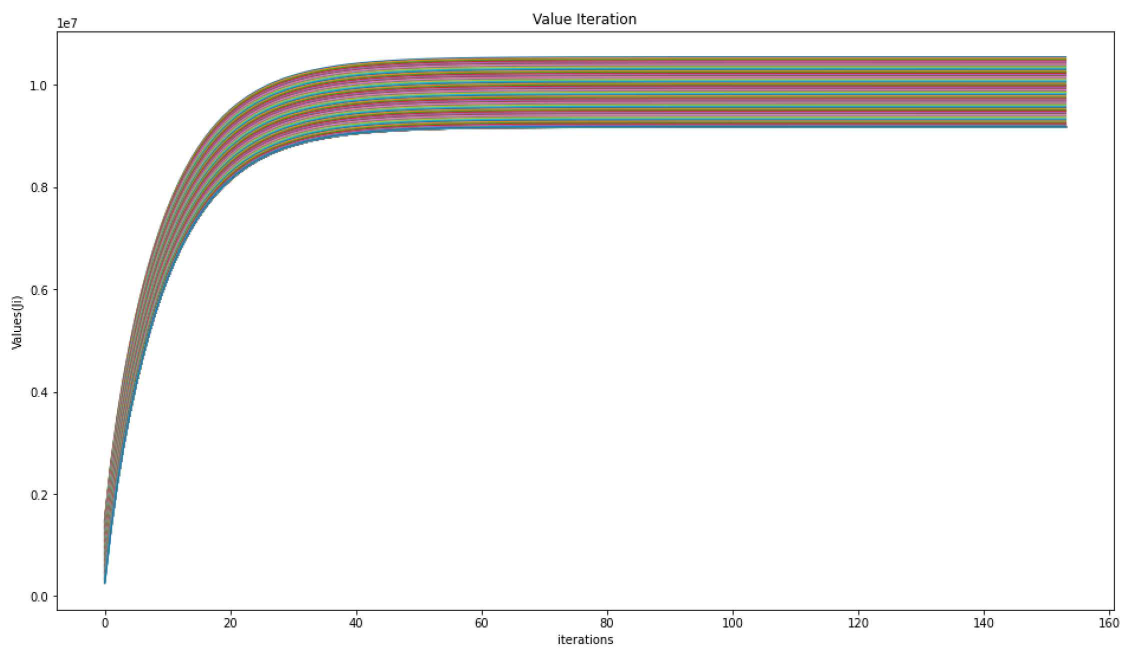
```
43: 'a2',
44: 'a2',
45: 'a2',
46: 'a2',
47: 'a2',

48: 'a2',
49: 'a2',
50: 'a2',
51: 'a2',
52: 'a2',
53: 'a2',
54: 'a2',
55: 'a2',
56: 'a1',
57: 'a1',
58: 'a1',
59: 'a1',
60: 'a1',
61: 'a1',
62: 'a1',
63: 'a1',
64: 'a1',
65: 'a1',
66: 'a1',
67: 'a1',
68: 'a1',
69: 'a1',
70: 'a1',
71: 'a1',
72: 'a1',
73: 'a1',
74: 'a1',
75: 'a1',
76: 'a1',
77: 'a1',
78: 'a1',
79: 'a1',
80: 'a1',
81: 'a1',
82: 'a1',
83: 'a1',
84: 'a1',
85: 'a1',
86: 'a1',
87: 'a1',
88: 'a1',
89: 'a1',
90: 'a1',
```

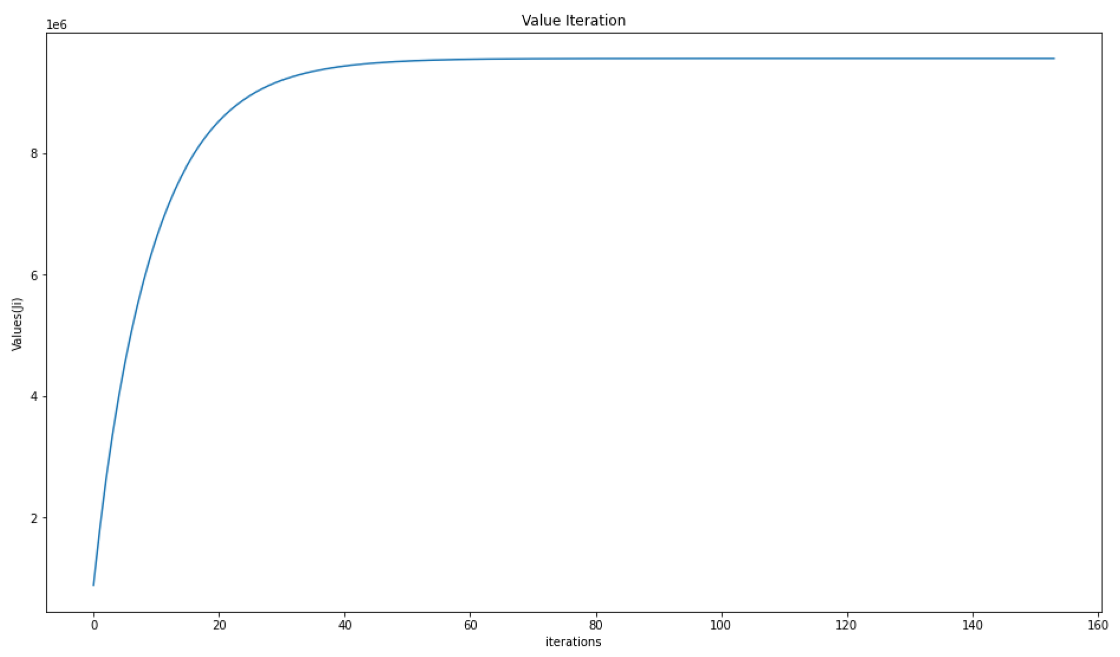
```
     91: 'a1',
     92: 'a1',
     93: 'a1',
     94: 'a1',
     95: 'a1',
     96: 'a1',
     97: 'a1',
     98: 'a1',
     99: 'a1',
     100: 'a1'}
```

```
# plot values of all 100 states vs iterations
plt.figure(figsize=(16,9))
iter_values = {x:[] for x in range(0,101)}
for values in all_values:
  for state,value in enumerate(values):
    iter_values[state].append(value)
for state in range(0,101):
  plt.plot(range(len(iter_values[state])), iter_values[state], label=state)

plt.xlabel('iterations')
plt.ylabel('Values(Ji)')
plt.title('Value Iteration')
plt.show()
```




```
# plot mean of all 100 state values vs iterations
plt.figure(figsize=(16,9))
iter_values = []
for values in all_values:
  iter_values.append(np.mean(np.array(values)))
plt.plot(range(len(iter_values)), iter_values)
plt.xlabel('iterations')
plt.ylabel('Values(Ji)')
plt.title('Value Iteration')
plt.show()
```




```
# Initialization
values = [0]*101
actions = ['a1', 'a2']
policy = {x:0 for x in range(0,101)}
```

```
%%time
# Gauss Seidel

all_values = []
not_done = True
while(not_done):
  delta = 0
  for state in range(0,101):
    j = values[state]

    action_values = []
    for action in actions:
      reward = 0
      for new_state in range(0, 101):
        reward += get_probab(state, action, new_state)*(get_reward(state, action) + gamma*values[new_state])
      action_values.append(reward)
    values[state] = np.max(np.array(action_values))
    policy[state] = actions[np.argmax(np.array(action_values))]

    delta = max(delta, abs(j - values[state]))
  if(delta < 0.1):
    not_done = False
  all_values.append(values.copy())
```

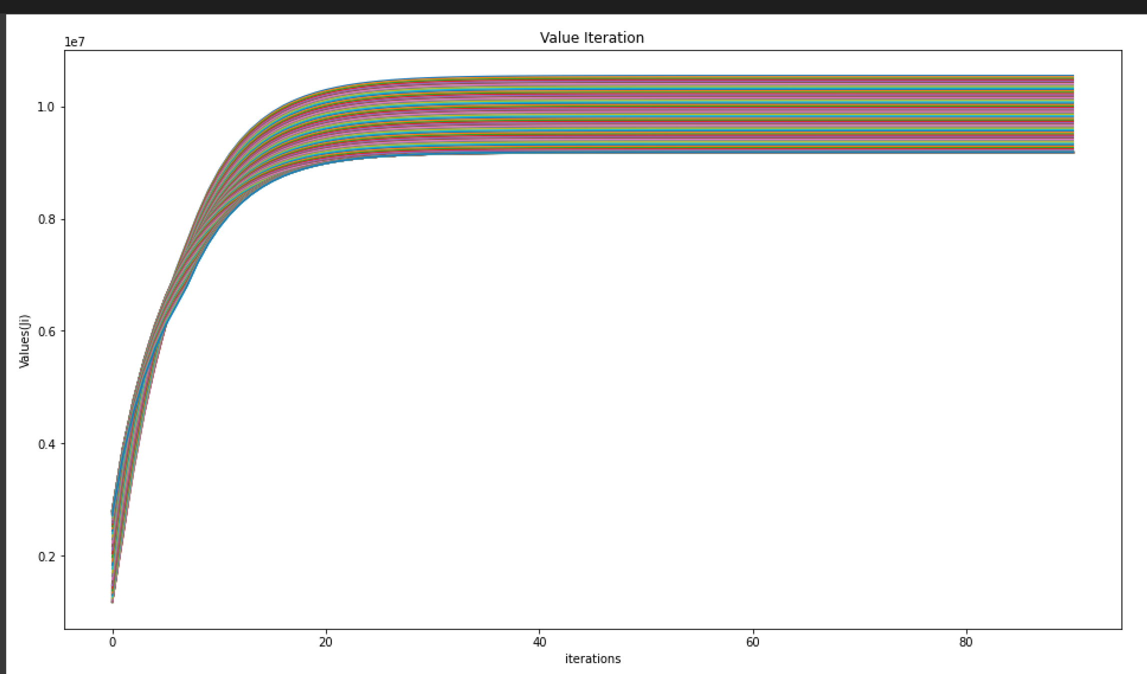
```
    CPU times: user 3min 55s, sys: 460 ms, total: 3min 55s
    Wall time: 3min 56s
```

```
# Policy That we got from Gauss Seidel
policy
```

```
     42: 'a2',
     43: 'a2',
     44: 'a2',
     45: 'a2',
     46: 'a2',
     47: 'a2',
     48: 'a2',
     49: 'a2',
     50: 'a2',
     51: 'a2',
     52: 'a2',
     53: 'a2',
     54: 'a2',
     55: 'a2',
     56: 'a1',
     57: 'a1',
     58: 'a1',
     59: 'a1',
     60: 'a1',
     61: 'a1',
     62: 'a1',
     63: 'a1',
     64: 'a1',
     65: 'a1',
     66: 'a1',
     67: 'a1',
     68: 'a1',
     69: 'a1',
     70: 'a1',
     71: 'a1',
     72: 'a1',
     73: 'a1',
     74: 'a1',
     75: 'a1',
     76: 'a1',
     77: 'a1',
     78: 'a1',
     79: 'a1',
     80: 'a1',
     81: 'a1',
     82: 'a1',
     83: 'a1',
     84: 'a1',
     85: 'a1',
     86: 'a1',
     87: 'a1',
     88: 'a1',
     89: 'a1',
     90: 'a1',
     91: 'a1',
     92: 'a1',
     93: 'a1',
     94: 'a1',
     95: 'a1',
     96: 'a1',
     97: 'a1',
     98: 'a1',
     99: 'a1',
     100: 'a1'}
```

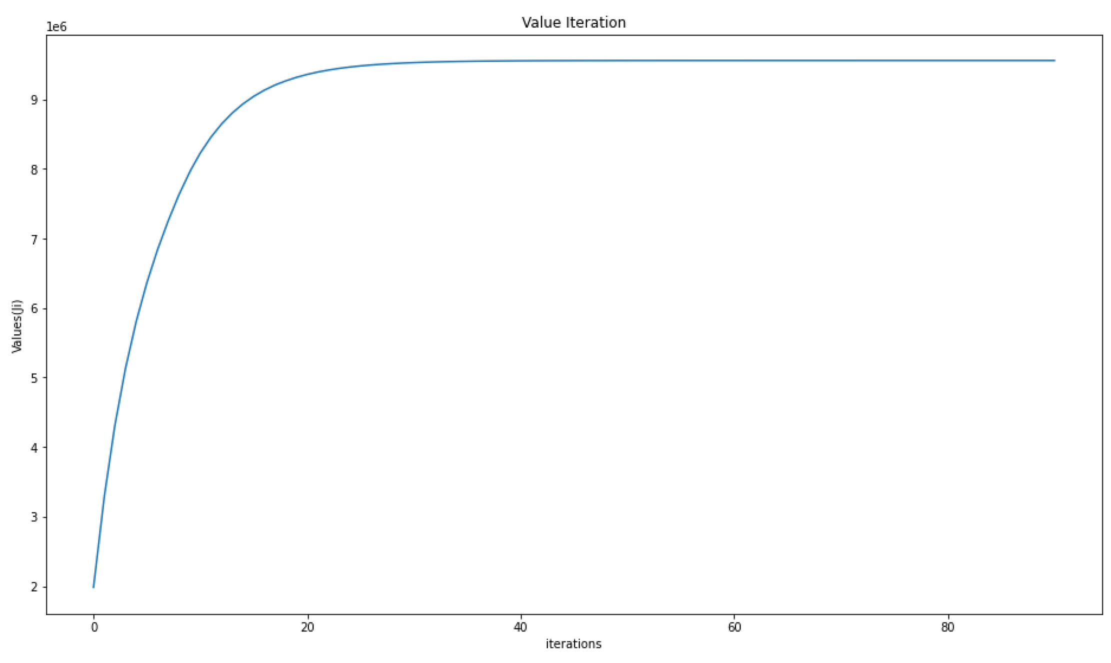
```
# plot values of all 100 states vs iterations
plt.figure(figsize=(16,9))
iter_values = {x:[] for x in range(0,101)}
for values in all_values:
  for state,value in enumerate(values):
    iter_values[state].append(value)
for state in range(0,101):
  plt.plot(range(len(iter_values[state])), iter_values[state], label=state)

plt.xlabel('iterations')
plt.ylabel('Values(Ji)')
plt.title('Value Iteration')
plt.show()
```




```
# plot mean of all 100 state values vs iterations
```

```
# plot mean of all 100 state values vs iterations
plt.figure(figsize=(16,9))
iter_values = []
for values in all_values:
  iter_values.append(np.mean(np.array(values)))
plt.plot(range(len(iter_values)), iter_values)
plt.xlabel('iterations')
plt.ylabel('Values(Ji)')
plt.title('Value Iteration')
plt.show()
```



✓ 0s completed at 23:01