Fitsome Inc - RL Assignment
Reinforcement Learning

S			
	B		
			G

Implement value-iteration and q-learning for above grid problem.

S - Start State
B - Bad State
G - Goal State
E - End State (Think of it some state outside the grid)

An agent start from the start state, S. It has to reach the goal state using the moves from the set {UP, DOWN, LEFT, RIGHT}. However the agent is high. From a state, if it intend to go up then it goes up with probability *a* and it goes left with probability b and right with probability b.

ACTION	**POSSIBLE STATES**	**PROBABILITY**
UP	UP, RIGHT, LEFT	a, b, b
DOWN	DOWN, RIGHT, LEFT	a, b, b
RIGHT	RIGHT, UP, DOWN	a, b, b
LEFT	LEFT, UP, DOWN	a, b ,b

If the agent reaches goal state, G, then it reaches the end state with probability 1. For each state, the agent gets a reward,

STATE	**REWARD**
G	100
B	-70
Other	-1
E	0

I. **Value Iteration**
Calculate the optimal policy using value iteration algorithm. Use discount factor = 0.99 and a = 0.8, b = 0.1.

II. **Q-learning**
Compute the optimal policy using q-learning assuming you don't know the transition and reward model. Use appropriate learning rate, α. Use ϵ-greedy exploration with ϵ = 0.05

(Optimal Action with probability 0.95 and random action 0.05). Try other values and see if it improves the convergence of q-learning.

Implement the code in the language of your choice(Python preferably). Also show the optimal policy generated.

Read about Markov Decision Process and Reinforcement Learning from Internet source

1. https://www.edx.org/course/artificial-intelligence-uc-berkeleyx-cs188-1x (Week 5 & 6)
2. https://www.cs.cmu.edu/~avrim/ML14/lect0326.pdf
3. http://www.ai.rug.nl/~mwiering/Intro_RLBOOK.pdf

You can use any other resources as well.

Solution:

Approach:

Value Iteration Method:

1,1	1,2	1,3	1,4
2,1(S)	2,2	2,3	2,4
3,1	3,2(B)	3,3	3,4
4,1	4,2	4,3	4,4(G)

V(S), where S can be (1,1) , (1,2) ,…………(4,4)
"a" is the set of actions i.e. UP, DOWN, LEFT and RIGHT
S' is the new state reached after taking an action from the list "a"

$$V(S) = max_a \left[\gamma \sum_{S'} P(S'|S,a).V(S') \right] + R(S)$$

R(S) is the reward attached to each state.
For terminal states, i.e. B and G states have V(S) = R(S). $P(S'|S,a).V(S')$ refers to the multiplication of the probability to land up in state S' from S if action a is taken and the Value V(S') of the landing up (new reached) state S'.

Calculate V(S), i.e. value of iteration for each state for “n” number of iterations so that the V(S) for a particular state converges to a stable value.
γ is the discount factor multiplied to the cost(i.e. value) associated with the movement. It gives more preference to the new values than the previous.

V(S) calculation example:

For a particular state S, four actions are attached to it. Since a is {UP,DOWN,RIGHT,LEFT}

Therefore, for each action calculate V(S) i.e. V(S,UP) , V(S,DOWN) , V(S,RIGHT) and V(S,LEFT) and see for which action a V(S,a) is max and assign that to V(S).

Repeat this process for all the state during every iteration.

While calculating the V(S), optimal policy π(S) can also be obtained by:

$$\pi(S) = argmax_a \left[\sum_{S'} P(S'|S,a).V(S') \right]$$

π(S) calculation example:

For a particular state S, four actions are attached to it. Since a is {UP,DOWN,RIGHT,LEFT}

Therefore, for each action calculate $[\sum_{S'} P(S'|S,a).V(S')]$ i.e. for a being UP) , DOWN , RIGHT and LEFT and see for which action a it is maximum and assign that action to π(S).

Repeat this process for all the state during every iteration.

Q-Learning Method:

Learn Q(S,a) values as you go:

1,1	1,2	1,3	1,4
2,1(S)	2,2	2,3	2,4
3,1	3,2(B)	3,3	3,4
4,1	4,2	4,3	4,4(G)

Each state S,
Say (1,1) has 4 actions associated with it so Q([1,1],UP) , Q([1,1],DOWN), Q([1,1],LEFT) and Q([1,1],RIGHT)

Similarly for any state S, we have to calculate Q(S,a) i.e. Q(S,UP) , Q(S,DOWN), Q(S,LEFT) and Q(S, RIGHT)

For calculating a new Q(S,a):

Consider the old estimate Q(S,a)
Consider the new sample estimate:

$$sample = R(S, a, S') + \gamma\ max_{a'} Q(S', a')$$

Therefore, the new estimate will be:
$Q(S, a) \leftarrow (1 - \alpha)Q(S, a) + (\alpha)$(sample)
i.e. $Q(S, a) \leftarrow (1 - \alpha)Q(S, a) + \alpha[R(S, a, S') + \gamma\ max_{a'} Q(S', a')]$

Q-learning converges to the optimal policy

While calculating the Q(S,a), optimal policy π(S) can also be obtained by:

$$\pi(S) = argmax_a[Q(S, a)]$$

Example how to calculate the Q(S,a) and π(S):

At each iteration:

For a particular state S, there are four actions attached to it, therefore, we have to calculate Q(S,UP) , Q(S,DOWN), Q(S,LEFT) and Q(RIGHT)

Lets, say we are calulcating, Q(S,UP) therefore, using ϵ-greedy exploration with ϵ = 0.05, we will see that S' was obtained with 95% probability of UP action or with 5% probability of a random action. Since, while Q-Learning we define a policy(here, current_policy=optimal policy obtained from value iteration) and try to obtain a new policy by calculating Q(S,a).

This way S' (the new state is obtained)
In order to calculate Q(S,a), that value of Q(S',a') is used which has max Q(S',a') i.e. that action Q-value of S' state which has the maximum value.
α is the learning rate used for updating the Q(S,a) and allowing the Q(S,a) to converge with that learning rate.

Q(S,a) for terminal states have Q(S',a') = 0
Once, the Q(S,a) for every now the π(S) is obtained by the above formula, for each state S take that action a for which the Q(S,a) is max. This gives us the Optimal Policy.

Following page consist of the implement of the above approach in R:

(A separate file fitsome_RL_assignment.R has been attached separately)

```
states_matrix=matrix(data="",nrow = 4,ncol = 4)
states_matrix[2,1]="S"
states_matrix[3,2]="B"
states_matrix[4,4]="G"
#creating the 4*4 states matrix showing which cell represent which state


reward_matrix=matrix(data=-1,nrow = 4,ncol = 4)
reward_matrix[3,2]=-70
reward_matrix[4,4]=100
#creating the 4*4 reward matrix showing the rewards associated with each state

value_matrix=matrix(data=0,nrow = 4,ncol = 4)
value_matrix[3,2]=-70
value_matrix[4,4]=100
#creating the 4*4 value matrix to store and show the value of iteration associated with each
state
#value of the terminal states is equivalent to the rewards at the terminal state

optimal_policy=matrix(data="",nrow = 4,ncol = 4)
#creating the 4*4 reward matrix to store the optimal_policy associated with each state

a=0.8   #probability associated with correct movement as per action
b=0.1   #probability associated with other movement as per action
# ACTION      POSSIBLE STATES             PROBABILITY
# UP            UP, RIGHT, LEFT             a, b, b
# DOWN                DOWN, RIGHT, LEFT          a, b, b
# RIGHT               RIGHT, UP, DOWN            a, b, b
# LEFT                LEFT, UP, DOWN             a, b ,b


g=0.99   #discount factor
dir=c("up","down","left","right")    #vector storing the order of the actions associated with each
state
prob=c(a,b,b)    #order of probability associated with each action

goal_state=which(states_matrix=="G",arr.ind = TRUE)
goal_row=goal_state[1]
goal_col=goal_state[2]
#storing the ID of Goal State

bad_state=which(states_matrix=="B",arr.ind = TRUE)
bad_row=bad_state[1]
bad_col=bad_state[2]
```

```
#storing the ID of Bad State

strt_state=which(states_matrix=="S",arr.ind = TRUE)
start_row=strt_state[1]
start_col=strt_state[2]
#storing the ID of Start State

row_vec=c(start_row)
col_vec=c(start_col)


nrows=nrow(states_matrix)
ncols=ncol(states_matrix)

for(i in c(1:(nrows-1))){
  row_id=row_vec[i]-1
  if(row_id<1){
    row_id=nrows+row_id
  }
  row_vec=append(row_vec,row_id)
}
for(i in c(1:(ncols-1))){
  col_id=col_vec[i]+1
  if(col_id>ncols){
    col_id=col_id-ncols
  }
  col_vec=append(col_vec,col_id)
}
#on the basis of the start state setting up the order to iterate along the states to calculate
#the value of iteration, row_vec and col_vec decides the flow of iteration to cover all the states

up_row=function(row_id){   #row_ids of the possible states if UP action is selected
  up=row_id-1
  right=row_id
  left=row_id
  if(up==0){
    up=1
  }
  return(c(up,right,left))
}
down_row=function(row_id){    #row_ids of the possible states if DOWN action is selected
  down=row_id+1
  right=row_id
  left=row_id
```

```
  if(down>nrows){
    down=nrows
  }
  return(c(down,right,left))
}
right_row=function(row_id){      #row_ids of the possible states if RIGHT action is selected
  right=row_id
  up=row_id-1
  down=row_id+1
  if(up==0){
    up=1
  }
  if(down>nrows){
    down=nrows
  }
  return(c(right,up,down))
}
left_row=function(row_id){    #row_ids of the possible states if LEFT action is selected
  left=row_id
  up=row_id-1
  down=row_id+1
  if(up==0){
    up=1
  }
  if(down>nrows){
    down=nrows
  }
  return(c(left,up,down))
}


up_col=function(col_id){       #column_ids of the possible states if UP action is selected
  up=col_id
  right=col_id+1
  left=col_id-1
  if(right>ncols){
    right=ncols
  }
  if(left==0){
    left=1
  }
  return(c(up,right,left))
}
down_col=function(col_id){      #column_ids of the possible states if DOWN action is selected
```

```
  down=col_id
  right=col_id+1
  left=col_id-1
  if(right>ncols){
    right=ncols
  }
  if(left==0){
    left=1
  }
  return(c(down,right,left))
}
right_col=function(col_id){    #column_ids of the possible states if RIGHT action is selected
  right=col_id+1
  up=col_id
  down=col_id
  if(right>ncols){
    right=ncols
  }
  return(c(right,up,down))
}
left_col=function(col_id){     #column_ids of the possible states if LEFT action is selected
  left=col_id-1
  up=col_id
  down=col_id
  if(left==0){
    left=1
  }
  return(c(left,up,down))
}

value_of_iteration=function(row_ids,col_ids,prob){      #Function to calculate a part of Value of
iteration for a particular State-Action pair
  val=0
  for(i in c(1:length(row_ids))){
    val=val+value_matrix[row_ids[i],col_ids[i]]*prob[i]
  }
  return(val)
}


for(x in c(1:1000)){      #doing 1000(editable to check the convergence) iterations to converge
the Value of Iteration at each iteration
  for(i in row_vec){
    for(j in col_vec){
```

```
      if(states_matrix[i,j]=="B" | states_matrix[i,j]=="G"){   #ignore the terminal states
        next
      }
      up_rows=up_row(i)
      up_cols=up_col(j)
      #new row and col ids if UP action is taken

      down_rows=down_row(i)
      down_cols=down_col(j)
      #new row and col ids if DOWN action is taken

      left_rows=left_row(i)
      left_cols=left_col(j)
      #new row and col ids if LEFT action is taken

      right_rows=right_row(i)
      right_cols=right_col(j)
      #new row and col ids if RIGHT action is taken

      v_s_up=value_of_iteration(up_rows,up_cols,prob)   #V(S,UP)
      v_s_down=value_of_iteration(down_rows,down_cols,prob)  #V(S,DOWN)
      v_s_left=value_of_iteration(left_rows,left_cols,prob)  #V(S,LEFT)
      v_s_right=value_of_iteration(right_rows,right_cols,prob) #V(S,RIGHT)

      v_s_list=c(v_s_up,v_s_down,v_s_left,v_s_right)   #list of V(S,UP), V(S,DOWN), V(S,LEFT)
and V(S,RIGHT)
      opt_dir=which.max(v_s_list)    #which V(S,action) out of the four is maximum
      opt_dir=dir[opt_dir]           #which action has the highest value of V(S,action)
      v_s=g*(max(v_s_up,v_s_down,v_s_left,v_s_right))+reward_matrix[i,j]     #V(S) is calculated
      value_matrix[i,j]=v_s          #updating the value V(S) associated with the state
      optimal_policy[i,j]=opt_dir    #updating the optimal policy associated with the state

    }
  }
}


q_matrix=matrix(data = c(1:16),nrow=4,ncol=4,byrow = TRUE)    #4*4 q-matrix to keep a track
of the state by assign an ID to each i.e. from 1 to 16
q_values=matrix(data = 0, nrow=16,ncol=4)    #16*4 q_values matrix where 1 to 16 rows
represent the states and the 4 columns refer to the actions associated with each state and
contains the Q(S,a) in each cell
current_policy=optimal_policy    #using the optimal policy obtained from the Value of Iteration
method as the current policy to start for Q-Learning method
```

```
optimal_policy_final=matrix(data="",nrow = 4,ncol = 4)   #final_optimal policy matrix to story the
final optimal policy obtained after Q-Learning
alpha=0.5    #learning rate (edit to see the changes in the convergence)

up_row=function(row_id){      #row_id of the state if UP action is selected
  up=row_id-1
  if(up==0){
    up=1
  }
  return(up)
}
down_row=function(row_id){    #row_id of the state if DOWN action is selected
  down=row_id+1
  if(down>nrows){
    down=nrows
  }
  return(down)
}
right_row=function(row_id){   #row_id of the state if RIGHT action is selected
  right=row_id
  return(right)
}
left_row=function(row_id){    #row_id of the state if LEFT action is selected
  left=row_id
  return(left)
}


up_col=function(col_id){      #column_id of the state if UP action is selected
  up=col_id
  return(up)
}
down_col=function(col_id){    #column_id of the state if DOWN action is selected
  down=col_id
  return(down)
}
right_col=function(col_id){   #column_id of the state if RIGHT action is selected
  right=col_id+1
  if(right>ncols){
    right=ncols
  }
  return(right)
}
left_col=function(col_id){    #column_id of the state if LEFT action is selected
```

```
  left=col_id-1
  if(left==0){
    left=1
  }
  return(left)
}

q_s_a=function(row_id,col_id){            #obtaining the old Q(S,a) values for all actions a associated with the state
  q_matrix_value=q_matrix[row_id,col_id]
  q=q_values[q_matrix_value,]
  return(q)
}


for(x in c(1:20000)){      #doing 20000(editable to check the convergence) iterations to converge the Q-value at each iteration
  row_id=start_row         #in Q-Learning every iteration starts from the Start State
  col_id=start_col         #here's the row and column ids of the Start State
  row_new=row_id
  col_new=col_id
  i=row_id
  j=col_id
  while(TRUE){
    i=row_new
    j=col_new

    up_rows=up_row(i)          #new row id if UP action is taken
    down_rows=down_row(i)      #new row id if DOWN action is taken
    left_rows=left_row(i)      #new row id if LEFT action is taken
    right_rows=right_row(i)    #new row id if RIGHT action is taken
    row_info=rbind(up_rows,down_rows,left_rows,right_rows)   #row ids associated with the result of the respective actions taken

    up_cols=up_col(j)          #new col id if UP action is taken
    down_cols=down_col(j)      #new col id if DOWN action is taken
    left_cols=left_col(j)      #new col id if LEFT action is taken
    right_cols=right_col(j)    #new col id if RIGHT action is taken
    col_info=rbind(up_cols,down_cols,left_cols,right_cols)   #column ids associated with the result of the respective actions taken


    action_policy=current_policy[i,j]      #as per current policy what should be the action taken
    id_action=which(dir==action_policy)
```

```
    #creating a random number from 1 to 100
    #if the number > 5 then we will go by the current policy
    #otherwise we choose a random action
    #utilising, ϵ-greedy exploration with ϵ = 0.05
    action_values=runif(1,1,100)
    if(action_values>5){
      action=current_policy[i,j]    #action obtained from current policy
    }
    else{
      new_value=runif(1,1,4)
      action=dir[new_value]         #action obtained from random
    }


    id_result=which(dir==action)    #resulting id of the action undertaken
    row_new=row_info[id_result]
    col_new=col_info[id_result]
    q_values_new_state=q_s_a(row_new,col_new)   #Q(S',a') associated with the new state S'
and all the actions a' associated with it
    q_matrix_val=q_matrix[i,j]
    if(states_matrix[i,j]=="G" | states_matrix[i,j]=="B"){    #calculating the Q(S,a) of the terminal
states
      q_values[q_matrix_val,]=(1-alpha)*q_values[q_matrix_val,]+alpha*(reward_matrix[i,j])
#updating the Q(S,a) value of the terminal states
      break
    }
    q_values[q_matrix_val,id_action]=(1-
alpha)*q_values[q_matrix_val,id_action]+alpha*(reward_matrix[i,j]+g*max(q_values_new_state,
na.rm = TRUE))
    #calculating and updating the Q(S,a) value of the other states obtained from the actions taken
  }
}

q_max_policies=c()
q_max_vals=c()
for(i in c(1:nrow(q_values))){    #Obtaining the Q-max values and policy associated with each
state among the actions associated with the state
  q_max_policies=append(q_max_policies,which.max(q_values[i,]))   #Obtaining the policy
associated with Q-max for that state
  q_max_vals=append(q_max_vals,max(q_values[i,]))    #Obtaining the Q-max value of each
state
}
q_max_policies=dir[q_max_policies]   #actions associated with the index values of the policies
```

```
for(i in c(1:nrows)){
  for(j in c(1:ncols)){
    if(states_matrix[i,j]=="G" | states_matrix[i,j]=="B"){
      optimal_policy_final[i,j]=""    #setting up null string as the policy for the terminal states
    }
    else{
      optimal_policy_final[i,j]=q_max_policies[q_matrix[i,j]]   #updating the optimal_policy_final
matrix with the policies obtained for each states after Q-Learning
    }
  }
}


print("Value Iteration Matrix")
value_matrix    #displaying Value Iteration Matrix
print("Optimal Policy Obtained from Value Iteration Algorithm")
optimal_policy  #displaying Optimal Policy Obtained from Value Iteration Algorithm
print("Optimal Policy obtained from Q-learning")
optimal_policy_final  #displaying Optimal Policy obtained from Q-learning
```