MATH4406 HW4

Adam James Murray, 42625364

September 26th, 2014

1. Consider a two front war. You have a finite force and need to distribute it between the two fronts. The opposing forces have almost impenetrable strongholds in their capitals, however their leadership is not imperialist in nature, and will not in general pursue you unless you invade their homeland. Your goal is to maximise territory and minimise casualties. We can conflate these two variables, since a loss in men will more than likely correspond to a loss in territory, likewise a gain in territory will add the forces of the captured lands to your army. Your actions are to divert numbers of forces between the fronts. The strongholds at the heart of enemy nations will see your army suffer high casualties and be chased back across the empire. Our discounting factor here is attrition - as the war rages on, the size of your and your opponent's fighting forces dwindles, as does their morale. For small armies ($\lambda \approx 0$), the cost of pushing to the enemy capital is low and the territory gain is high, hence the optimal policy is to push forward at all costs and storm the capital. For large armies ($\lambda \approx 1$), the casualties are high and the territory gain low, and the optimal policy is to perhaps hedge your bets and only practice your imperialism on the local peasants.

2. The first observation to make is that given the symmetry of the problem, we would expect an optimal policy to be symmetric about the origin (state 0) since the cost of the states on the left is exactly mirrored by the costs of the states on the right. For example, if the optimal policy dictated that we choose action -1 in state -4, then we would expect the optimal action to be 1 in state 4.

Furthermore, when the discounting factor is close to zero, the high costs at the boundary points are less of a factor in our decision, and hitting the adjacent states with probability 0.5 will most likely be the best we can do. Hence in this case we would expect an optimal policy to drive the system to the boundary.

For a λ close to 1, the penalty for hitting the boundary is more severe and in this case we would expect the opposite, i.e. that the system will try to avoid the boundary, and make do with the small rewards (or costs) associated with staying near the origin.

3-5. See appendix for code used.

7. Running each algorithm for $\lambda \in (0, 1)$ in increments of 0.1, we found that each method agreed on the optimal policies. We present the optimal policies in the following table.

λ	Optimal Policy
0.01 - 0.63	3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3
0.64 - 0.85	3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3
0.86 - 0.92	3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3
0.93 - 0.95	3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3
0.96 - 0.99	3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3

When running the policy evaluation we used the parameters $\varepsilon = 0.001$ with a break condition after 1000 iterations. We have also denoted our decision at the origin as 0 to represent that the choice made will not effect the overall behaviour of the policy.

Inspecting the optimal policies above, we see that for low λ, i.e. $\lambda \in (0.01, 0.63)$, the optimal policy is simply to drive the system to the end points. This is expected since the discounting factor nullifies the effect of the high cost at the end points.

For very high λ, i.e. $\lambda \in (0.96, 0.99)$, we find that the optimal policy is to try and keep the system as close as possible to the origin, as we would expect, given that in these cases the extremely high cost of the boundary is still present.

We then observe that in the interim cases between the two extreme, there is a systematic development of optimal policies, with the optimal deviation from the origin inversely proportional to the discounting factor λ.

Further note that all the optimal policies are symmetric about the origin. This is to be expected given the symmetries of the model.

Appendix

./MATH4406/HW4.py

```
import string
import numpy as np
import copy
np.set_printoptions(suppress=1,precision=5)

def GenerateDecisionRule(index,stateSpace):
    #make sure our index is sensible
    index = index%(pow(2,len(stateSpace)-2))
    #convert the index to binary string with appropriate padding
    rule = '{:0{pad}b}'.format(index, pad=(len(stateSpace)-2))
    #convert rule string to integer list and convert zeros to -1s
    rule = [(string.atoi(rule[j]) - (1-string.atoi(rule[j]))) for j in xrange(len(rule))]
    #add in the boundary actions
    rule.append(-3)
    rule.insert(0,3)
    return rule

def TransProb(state, newState, action,stateSpace):
    if state == 0 and (newState == 1 or newState == state):
        return 0.5
    elif state == len(stateSpace)-1 and (newState == len(stateSpace)-2 or newState == state):
        return 0.5
    elif newState == state-1 and action == -1:
        return 0.75
    elif newState == state+1 and action == -1:
        return 0.25
    elif newState == state-1 and action == 1:
        return 0.25
    elif newState == state+1 and action == 1:
        return 0.75
    else:
        return 0


def GenerateProbabilityMatrix(decisionRule,stateSpace):
    #create an iterable for the dimensions of our matrix
    dim = xrange(len(stateSpace))
    #initialise the matrix
    probMatrix = [[0 for j in dim]for i in dim]
    #populate matrix withh transition probabilities
    for i in dim:
        for j in dim:
            probMatrix[i][j] = TransProb(i,j,decisionRule[i],stateSpace)
    return np.matrix(probMatrix)

def GenerateRewardVector(decisionRule,stateSpace):
    rewardVector = [decisionRule[i]*stateSpace[i] for i in xrange(len(stateSpace))]
    return np.matrix(rewardVector).T

def CalculateDiscountedReward(lmda,state,action,vOld,stateSpace):
    total = 0
    for i in xrange(len(stateSpace)):
        total = total + lmda*TransProb(state,i,action,stateSpace)*vOld[i]
    return total



#BRUTE FORCE METHOD
def BruteForceSolver(lmda,stateSpace):
    S = stateSpace
    #start our search at negative infinity
    highestValue = np.NINF
    finalRule = 0

    for i in xrange(pow(2,len(S)-2)):
        rule = GenerateDecisionRule(i,S)
        P = GenerateProbabilityMatrix(rule,S)
        r = GenerateRewardVector(rule,S)
        A = np.identity(len(r)) - lmda*P
        Ainv = np.linalg.inv(A)
        value = sum(Ainv*r)
        if value >= highestValue:
            highestValue = value
            #set the middle action as zero for readibility
            rule[(len(S)-1)/2] = 0
            finalRule = rule
            #print rule
    return finalRule



#VALUE ITERATION ALGORITHM
def RunPolicyEvaluationAlg(lmda,epsilon,stateSpace,maxItr):
    S = stateSpace
    vOld = [0 for i in xrange(len(S))]
    vNew = [0 for i in xrange(len(S))]
    policy = [0 for i in xrange(len(S))]
    policy[0] = 3
```

```
    policy[len(S)-1] = -3

    #loop a max of maxItr times
    for i in xrange(maxItr):
        for j in xrange(len(S)):
            #boundary cases
            if j == 0:
                vNew[j] = S[j]*3 + CalculateDiscountedReward(lmda,j,3,vOld,S)
            elif j == len(S)-1:
                vNew[j] = S[j]*(-3) + CalculateDiscountedReward(lmda,j,3,vOld,S)
            else: #all other cases
                a1 = S[j]*(1) + CalculateDiscountedReward(lmda,j,1,vOld,S)
                a2 = S[j]*(-1) + CalculateDiscountedReward(lmda,j,-1,vOld,S)
                if a1 > a2:
                    vNew[j] = a1
                    policy[j] = 1
                else:
                    vNew[j] = a2
                    policy[j] = -1
        if np.linalg.norm(np.matrix(vNew) - np.matrix(vOld)) < (epsilon*(1-lmda))/(2*lmda):
            break
        vOld = copy.copy(vNew)

    policy[(len(S)-1)/2] = 0
    return policy


#POLICY ITERATION ALGORITHM
def RunPolicyIterationAlg(lmda,stateSpace):
    #choose the start policy
    policyOld = GenerateDecisionRule(0,stateSpace)
    policyNew = GenerateDecisionRule(1,stateSpace)
    while policyOld != policyNew:
        policyOld = copy.copy(policyNew)
        P = GenerateProbabilityMatrix(policyOld,stateSpace)
        r = GenerateRewardVector(policyOld,stateSpace)
        A = np.identity(len(r)) - lmda*P
        Ainv = np.linalg.inv(A)
        v = Ainv*r
        vOld = np.NINF
        for i in xrange(pow(2,len(stateSpace)-2)):
            policy = GenerateDecisionRule(i,stateSpace)
            P = GenerateProbabilityMatrix(policy,stateSpace)
            r = GenerateRewardVector(policy,stateSpace)
            vNew = sum(r + lmda*P*v)
            if vNew > vOld:
                policyNew = GenerateDecisionRule(i,stateSpace)
                vOld = vNew
        #print policyOld
        #print policyNew
    policyNew[(len(stateSpace)-1)/2] = 0
    return policyNew


S = range(-5,6)
epsilon = 0.001
maxItr = 1000

print '----\n Solutions for brute force evaluation'
lmda = 0.01
while lmda < 1:
    print "{:.2f}".format(lmda),BruteForceSolver(lmda,S)
    lmda = lmda + 0.01

print '----\n Solutions for policy evaluation'
lmda = 0.01
while lmda < 1:
    print "{:.2f}".format(lmda),RunPolicyEvaluationAlg(lmda,epsilon,S,maxItr)
    lmda = lmda + 0.01

print '----\n Solutions for policy iteration'
lmda = 0.01
while lmda < 1:
    print "{:.2f}".format(lmda),RunPolicyIterationAlg(lmda,S)
    lmda = lmda + 0.01
```

```
output:
----
 Solutions for brute force evaluation
0.01 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.02 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.03 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.04 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.05 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.06 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.07 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.08 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.09 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.10 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.11 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
```

```
0.12 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.13 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.14 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.15 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.16 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.17 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.18 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.19 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.20 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.21 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.22 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.23 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.24 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.25 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.26 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.27 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.28 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.29 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.30 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.31 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.32 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.33 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.34 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.35 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.36 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.37 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.38 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.39 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.40 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.41 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.42 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.43 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.44 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.45 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.46 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.47 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.48 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.49 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.50 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.51 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.52 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.53 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.54 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.55 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.56 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.57 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.58 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.59 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.60 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.61 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.62 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.63 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.64 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.65 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.66 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.67 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.68 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.69 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.70 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.71 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.72 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.73 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.74 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.75 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.76 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.77 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.78 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.79 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.80 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.81 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.82 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.83 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.84 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.85 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.86 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.87 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.88 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.89 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.90 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.91 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.92 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.93 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.94 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.95 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.96 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.97 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.98 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.99 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
----
 Solutions for policy evaluation
0.01 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.02 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.03 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.04 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.05 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
```

```
0.06 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.07 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.08 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.09 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.10 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.11 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.12 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.13 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.14 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.15 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.16 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.17 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.18 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.19 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.20 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.21 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.22 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.23 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.24 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.25 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.26 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.27 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.28 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.29 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.30 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.31 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.32 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.33 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.34 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.35 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.36 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.37 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.38 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.39 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.40 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.41 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.42 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.43 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.44 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.45 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.46 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.47 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.48 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.49 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.50 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.51 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.52 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.53 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.54 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.55 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.56 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.57 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.58 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.59 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.60 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.61 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.62 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.63 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.64 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.65 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.66 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.67 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.68 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.69 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.70 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.71 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.72 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.73 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.74 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.75 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.76 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.77 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.78 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.79 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.80 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.81 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.82 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.83 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.84 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.85 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.86 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.87 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.88 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.89 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.90 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.91 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.92 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.93 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.94 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.95 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.96 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.97 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.98 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.99 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
----
```

```
 Solutions for policy iteration
0.01 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.02 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.03 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.04 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.05 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.06 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.07 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.08 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.09 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.10 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.11 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.12 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.13 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.14 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.15 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.16 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.17 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.18 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.19 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.20 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.21 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.22 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.23 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.24 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.25 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.26 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.27 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.28 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.29 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.30 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.31 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.32 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.33 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.34 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.35 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.36 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.37 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.38 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.39 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.40 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.41 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.42 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.43 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.44 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.45 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.46 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.47 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.48 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.49 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.50 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.51 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.52 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.53 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.54 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.55 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.56 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.57 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.58 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.59 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.60 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.61 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.62 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.63 [3, -1, -1, -1, -1, 0, 1, 1, 1, 1, -3]
0.64 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.65 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.66 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.67 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.68 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.69 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.70 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.71 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.72 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.73 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.74 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.75 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.76 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.77 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.78 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.79 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.80 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.81 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.82 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.83 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.84 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.85 [3, 1, -1, -1, -1, 0, 1, 1, 1, -1, -3]
0.86 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.87 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.88 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.89 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.90 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.91 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.92 [3, 1, 1, -1, -1, 0, 1, 1, -1, -1, -3]
0.93 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.94 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
```

```
0.95 [3, 1, 1, 1, -1, 0, 1, -1, -1, -1, -3]
0.96 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.97 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.98 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
0.99 [3, 1, 1, 1, 1, 0, -1, -1, -1, -1, -3]
```