Stochastic Optimization Assignment 2

Aytek Mutlu

October 15, 2019

1 Introduction

For the given deteriorating system; states are assigned as the time that passed since last replacement. Therefore, the first state represents the state where the maintenance is just happened and state number 91 represents the state where it is 100% probability that a failure will occur. Probability matrix is structured such that transition probabilities between consecutive states decreases from 90% to 0% and transition probabilities from consecutive states to the initial state increases from 10% to 100%.

2 Stationary Distribution & Average Replacement Cost

For the calculation of stationary distribution, it is assumed that the process starts from state 1 where the replacement has just taken place. Then, this state distribution matrix is updated by multiplying it with the transition probability matrix iteratively until the state distribution matrix converges to its stationary point. Here are the first 5 elements of stationary distribution:

	Stationary Distribution
State1	0.1460975
State2	0.1314878
State3	0.1170241
State4	0.1029812
State5	0.08959367

Average replacement cost is then calculated with multiplication of stationary distribution with reward matrix where reward matrix is set to be the probability of failure times replacement cost at each state. Therefore, the reward matrix is a list from 0.1 to 1 with 0.01 increments.

Multiplication of stationary distribution and reward matrix, which is average replacement cost, resulted in: **0.1461**.

3 Poisson Equation

Poisson equation is solved by setting linear equations of values of each state and g against reward. There are 91 states and therefore 91 equations with 92 unknowns (91 state and g). Therefore, V(0)=0 is assumed to obtain a solution. From the solution, g is obtained to be **0.1461**

4 Policy Iteration

With the introduction of the second action of preventive replacement with cost of 0.5 at each stage, the problem has changed to find the optimal policy. Firstly, the optimal policy is obtained with policy iteration. Initial policy is selected as first action at each state. Then, V and g are calculated for this policy and the policy is updated. This process continued until the policy does not change between iterations. From that setup, optimal policy and g is extracted. Cost for optimal policy is found to be **0.144918** and optimal policy is found to be not conducting any preventive replacement before 14th time unit and then choosing preventive replacement afterwards.

5 Value Iteration

For the selection of optimal policy, a value iteration procedure is also applied. Initial value matrix is assigned to be all zeros. Then, value function is updated with cost minimizing actions at each iteration. Once the value function has reached to a convergence (the changes in itself becomes insignificant), the long-run cost minimizing actions are captured to be the optimal policy. Not surprisingly, the optimal policy is again not choosing preventive replacement before 14th time unit. The cost associated for this optimal policy is found to be **0.144918** as well.

6 Code

```
library(matlib)


#probability matrix for first action
p_first = matrix(c(0),91,91)
p_first[,1]=seq(0.1,1,0.01)

for(i in (1:90)){
  p_first[i,1+i]=1-p_first[i,1]
}

#stationary distribution
pi = matrix(c(0),1,91)
pi[1]=1
convergence = 1
while(convergence>0.000001){
  pi_new = pi %*% p_first
  pi_dif = pi_new - pi
  convergence = max(pi_dif)
  pi = pi_new
}

r_first = matrix(c(seq(0.1,1,0.01)),91,1)
avg_replacement = pi %*% r_first

#Poisson function
Poisson <- function(r,p){
  coefs = matrix(c(0),92,92)
  coefs[1:91,92] = 1
  coefs[92,1]=1

  coef_result = matrix(c(0),92,1)

  for (i in (1:91)){
    coefs[i,1:91] =  - p[i,1:91]
    coefs[i,i] = 1 - p[i,i]
    coef_result[i,1] = r[i,1]
  }

  A = solve(coefs,coef_result,fractions=TRUE)

  ret = list("V"=A[1:91],"g"=A[92])
  return(ret)
}


##poisson for first action
```

```
poisson_first = Poisson(r_first,p_first)
g_first_poisson = poisson_first$g
V_first_poisson = poisson_first$V

#preventive Replacement cost and probability matrix
p_second = matrix(c(0),91,91)
p_second[1:91,1] = 1

r_second = matrix(c(0.5),91,1)


###policy iteration
##initial policy is no preventive replacement at all (1st
    action for each state)
R_initial = matrix(c(1),91,1)

R = R_initial

change = 1000
while(change≠0){
  r_policy_iteration = matrix(c(0),91,1)
  p_policy_iteration = matrix(c(0),91,91)
  for(i in (1:91)){
    r_policy_iteration[i,1] = ifelse(R[i,1]==1,r_first[i,1],
        r_second[i,1])
    p_policy_iteration[i,1:91] = if(R[i,1]==1) p_first[i
        ,1:91] else p_second[i,1:91]
  }
  poisson_policy_iteration = Poisson(r_policy_iteration,
      p_policy_iteration)
  first_action = (r_first + p_first %*%
      poisson_policy_iteration$V)
  second_action = (r_second + p_second %*%
      poisson_policy_iteration$V)
  R_new = ifelse(first_action >second_action ,2,1)
  change = sum(abs(R_new - R))
  print(change)
  R  = R_new
}

g_policy_iteration = (pmin(first_action,second_action) -
    poisson_policy_iteration$V)[1]
R_optimal_policy = R


##value iteration
##initial value matrix is all zeros
V_initial = matrix(c(0),91,1)

V = V_initial
```

```
change = 100
while(change≠1){
  first_action = (r_first + p_first %*% V)
  second_action = (r_second + p_second %*% V)
  V_new = pmin(first_action,second_action)
  V_dif = V_new - V
  change = length(unique(round(V_dif,6)))
  V = V_new
}
g_value_iteration = V_dif[1]
R_optimal_value = ifelse(first_action >second_action ,2,1)
```