```
#!/usr/bin/python3

import numpy as np # numerical python
# printoptions: output limited to 2 digits after decimal point
np.set_printoptions(precision=2, suppress=False)
import re # re: regular expressions, used in cleanbrackets

T=25 # number of rounds
# player 0 = low cost
# player 1 = high cost
cost = [57, 71] # cost
# first index is always player
demandpotential = [[0]*T,[0]*T] # two lists for the two players
demandpotential[0][0]=200 # initialize first round 0
demandpotential[1][0]=200
prices = [[0]*T,[0]*T]  # prices over T rounds
profit = [[0]*T,[0]*T]  # profit in each of T rounds

def monopolyprice(player, t): # myopic monopoly price
    return (demandpotential[player][t] + cost[player])/2

def updatePricesProfitDemand(pricepair, t):
    # pricepair = list of prices for players 0,1 in current round t
    for player in [0,1]:
        price = pricepair[player]
        prices[player][t] = price
        profit[player][t] = \
            (demandpotential[player][t] - price)*(price - cost[player])
        if t<T-1 :
            demandpotential[player][t+1] = \
                demandpotential[player][t] + (pricepair[1-player] - price)/2
    return

def totalprofit(): # gives pair of total profits over T periods
    return sum(profit[0]), sum(profit[1])

def avgprofit(): # gives pair of average profits per round
    return sum(profit[0])/T, sum(profit[1])/T

def match (stra0, stra1):
    # matches two strategies against each other over T rounds
    # each strategy is a function giving price in round t
    # assume demandpotentials in round 0 are untouched, rest
    # will be overwritten
    for t in range(T):
        pricepair = [ stra0(t), stra1(t) ]
        # no dumping
        pricepair[0] = max (pricepair[0], cost[0])
        pricepair[1] = max (pricepair[1], cost[1])
        updatePricesProfitDemand(pricepair, t)
    return avgprofit()

def tournament(strats0, strats1):
    # strats0,strats1 are lists of strategies for players 0,1
    # all matched against each other
    # returns resulting pair A,B of payoff matrices
    m = len(strats0)
    n = len(strats1)
    # A = np.array([[0.0]*n]*m) # first index=row, second=col
    # B = np.array([[0.0]*n]*m)
    A = np.zeros((m,n))
    B = np.zeros((m,n))
    for i in range (m):
        for j in range (n):
            A[i][j], B[i][j] = match (strats0[i], strats1[j])
    return A,B

```

```
68  def cleanbrackets(astring):
69      # formats matrix string from np.array_str(A) for lrsnash
70      astring = re.sub('[\[\]]', '', astring)
71      astring = re.sub('\n ', '\n', astring)
72      astring = re.sub('\.', ' ', astring)
73      return astring
74
75  def outgame (A,B,divideby):
76      # to stdout: A,B payoff matrices for use with lrsnash
77      # divides entries by divideby (e.g. 10) to get fewer digits
78      # all payoffs output as rounded integers
79      # also gnuplot output in files, REQUIRES ./PLOT to exist
80
81      m = len(A)
82      n = len(A[0])
83      print ("A =")
84      A = A / divideby
85      np.set_printoptions(precision=0)
86      print (cleanbrackets(np.array_str(A)))
87      print ("\nB =")
88      B = B / divideby
89      print (cleanbrackets(np.array_str(B)))
90      # create gnuplot files in ./PLOT/
91      for i in range (m):
92          out = open("PLOT/"+str(i)+stratsinfo0[i],'w')
93          for j in range (n):
94              out.write(str(A[i][j])+" "+str(B[i][j])+"\n")
95      out = open("PLOT/gplot",'w')
96      out.write('set terminal postscript eps color\n')
97      out.write('set output "Pareto.eps"\n')
98      out.write("plot")
99      for i in range (m):
100         out.write(' "'+str(i)+stratsinfo0[i]+'" with lines lw 3,')
101     out.write("\n")
102     return
103
104 # strategies with varying parameters
105 def myopic(player, t):
106     return monopolyprice(player, t)
107
108 def const(player, price, t): # constant price strategy
109     if t == T-1:
110         return monopolyprice(player, t)
111     return price
112
113 def imit(player, firstprice, t): # price imitator strategy
114     if t == 0:
115         return firstprice
116     if t == T-1:
117         return monopolyprice(player, t)
118     return prices[1-player][t-1]
119
120 def fight(player, firstprice, t): # simplified fighting strategy
121     if t == 0:
122         return firstprice
123     if t == T-1:
124         return monopolyprice(player, t)
125     aspire = [ 207, 193 ] # aspiration level for demand potential
126     D = demandpotential[player][t]
127     Asp = aspire [player]
128     if D >= Asp: # keep price; DANGER: price will never rise
129         return prices[player][t-1]
130     # adjust to get to aspiration level using previous
131     # opponent price; own price has to be reduced by twice
132     # the negative amount D - Asp to get demandpotential to Asp
133     P = prices[1-player][t-1] + 2*(D - Asp)
134     # never price to high because even 125 gives good profits
135     P = min(P, 125)
136     return P
137
```

```
138 # sophisticated fighting strategy, compare fight()
139 # estimate *sales* of opponent as their target, kept between
140 # calls in global variable oppsaleguess[]. Assumed behavior
141 # of opponent is similar to this strategy itself.
142 oppsaleguess = [61, 75] # first guess opponent sales as in monopoly
143 def guess(player, firstprice, t): # predictive fighting strategy
144     if t == 0:
145         oppsaleguess[0] = 61 # always same start
146         oppsaleguess[1] = 75 # always same start
147         return firstprice
148     if t == T-1:
149         return monopolyprice(player, t)
150     aspire = [ 207, 193 ] # aspiration level
151     D = demandpotential[player][t]
152     Asp = aspire [player]
153     if D >= Asp: # keep price, but go slightly towards monopoly if good
154         pmono = monopolyprice(player, t)
155         pcurrent = prices[player][t-1]
156         if pcurrent > pmono: # shouldn't happen
157             return pmono
158         if pcurrent > pmono-7: # no change
159             return pcurrent
160         # current low price at 60%, be accommodating towards "collusion"
161         return .6 * pcurrent + .4 * (pmono-7)
162     # guess current *opponent price* from previous sales
163     prevsales = demandpotential[1-player][t-1] - prices[1-player][t-1]
164     # adjust with weight alpha from previous guess
165     alpha = .5
166     newsalesguess = alpha * oppsaleguess[player] + (1-alpha)*prevsales
167     # update
168     oppsaleguess[player] = newsalesguess
169     guessoppPrice = 400 - D - newsalesguess
170     P = guessoppPrice + 2*(D - Asp)
171     if player == 0:
172         P = min(P, 125)
173     if player == 1:
174         P = min(P, 130)
175     return P
176
177 strats0 = [ # use lambda to get function with single argument t
178     lambda t : myopic(0,t)         # 0 = myopic
179     , lambda t : guess(0,125,t)    # 1 = clever guess strategy
180     , lambda t : const(0,125,t)
181     , lambda t : const(0,117,t)
182     , lambda t : const(0,114.2,t)
183     , lambda t : const(0,105,t)
184     , lambda t : const(0,100,t)    # suppressed for easier plot
185     , lambda t : const(0,95,t)
186     , lambda t : imit(0,120,t)
187     , lambda t : imit(0,110,t)
188     , lambda t : fight(0,125,t)
189     ]
190
191 # description of above strategies, please maintain *manually*
192 stratsinfo0 = [
193     "myopic"
194     , "guess125"
195     , "const125"
196     , "const117"
197     , "const114.2"
198     , "const105"
199     , "const100"
200     , "const95"
201     , "imit120"
202     , "imit110"
203     , "fight125"
204     ]
205
```

```
206 strats1 = [
207     lambda t : myopic(1,t)         # 0 = myopic
208     , lambda t : guess(1,130,t)    # 1 = clever guess strategy
209     , lambda t : imit(1,131,t)     # 2 = imit starting nice
210     , lambda t : imit(1,114.2,t)   # 3 = imit starting competitive
211     , lambda t : fight(1,130,t)    # 4 = aggressive fight
212     ]
213 stratsinfo1 = [
214     "myopic"
215     , "guess130"
216     , "imit131"
217     , "imit114.2"
218     , "fight130"
219     ]
220
221 # sample detailed output of two strategies
222 i = 1  # clever guess for player 0
223 # i = 2  # const125 for player 0
224 j = 1  # clever guess for player 1
225 print ("matching", i, stratsinfo0[i],"to", j, stratsinfo1[j])
226 match (strats0[i], strats1[j])
227 print (np.array(demandpotential))
228 print (np.array(prices))
229 print (np.array(profit))
230 avgprof = avgprofit()
231 print (np.array([avgprof[0], avgprof[1]]))
232 print ()
233
234 # tournament
235 # test of single-item list in tournament
236 # A,B = tournament ([ lambda t : myopic(0,t) ], [lambda t : myopic(1,t) ] )
237 A,B = tournament (strats0, strats1)
238 # outgame(A,B,10) # output divided by 10
239 outgame(A,B,1) # output divided by 1 (full 4 digits)
240 # reset to 2 decimal points
241 # np.set_printoptions(precision=2, suppress=False)
242 # print (A) # print with brackets
243 # print (B) # print with brackets
244 print (len(strats0),len(strats1)," should be",
245     len(stratsinfo0),"x",len(stratsinfo1))
246     # check that strats0 and stratsinfo0 match in length
247 print (stratsinfo0) # information about used strategies
248 print (stratsinfo1)
249
250
```