Problem Set 1
GR6493 [Dean]
Miguel Acosta, Duarte Gonçalves, & Sara Shahanaghi
November 2017

Question 1 **SARA YOUR WRITE-UP GOES HERE**

Question 2 **DUARTE THERE IS SOME COMMENTED LATEX CODE THAT YOU MAY OR MAY NOT WANT TO USE HERE**

1.

2.

3.

Question 3 **1.** For concreteness, the utilities are

	I	H
d	$-c_I$	$-c_H$
n	$-s$	0

$\leq$

	I	H
d	$-\bar{c}_I$	$-\bar{c}_H$
n	$-s$	0

with $\bar{c}_I < c_I < s$ and $c_H > \max\{0, \bar{c}_H\}$.

NIAS: Algebra In this example, the following conditions constitute NIAS:

$$\sum_{\omega \in \{I,H\}} \mu_\omega P(d \mid \omega)[u(d(\omega)) - u(n(\omega))] \geq 0$$
$$\sum_{\omega \in \{I,H\}} \mu_\omega P(n \mid \omega)[u(n(\omega)) - u(d(\omega))] \geq 0$$

i.e.,

$$\mu_I P(d \mid I)[-c_I + s] + (1 - \mu_I)P(d \mid H)[-c_H] \geq 0 \tag{1a}$$
$$\mu_I P(n \mid I)[-s + c_I] + (1 - \mu_I)P(n \mid H)[c_H] \geq 0. \tag{1b}$$

Replacing $P(n \mid \omega)$ with $1 - P(d \mid \omega)$ in (1b)

$$\mu_I(1 - P(d \mid I))[-s + c_I] + (1 - \mu_I)(1 - P(d \mid H))[c_H] \geq 0$$

i.e.,

$$\mu_I P(d \mid I)[-c_I + s] + (1 - \mu_I)P(d \mid H)[-c_H] \geq \mu_I(s - c_I) - (1 - \mu_I)c_H$$

Which combines with (1a) to form a single NIAS condition

$$\mu_I P(d \mid I)[-c_I + s] + (1 - \mu_I)P(d \mid H)[-c_H] \geq \max\{\mu_I(s - c_I) - (1 - \mu_I)c_H, 0\} \tag{2}$$

Let's examine the RHS:

$$\mu_I(s - c_I) - (1 - \mu_I)c_H > 0 \iff \frac{\mu_I}{1 - \mu_I} > \frac{c_H}{s - c_I} \tag{3}$$

When this is true, then condition (2) becomes

$$\begin{aligned} &\mu_I P(d \mid I)[-c_I + s] + (1 - \mu_I)P(d \mid H)[-c_H] \geq \mu_I(s - c_I) - (1 - \mu_I)c_H \\ \iff &\frac{\mu_I}{1 - \mu_I} P(d \mid I) - P(d \mid H)\frac{c_H}{s - c_I} \geq \frac{\mu_I}{1 - \mu_I} - \frac{c_H}{s - c_I} > 0 \\ \implies &\frac{P(d \mid I)}{P(d \mid H)} > \frac{c_H}{s - c_I}\left[\frac{\mu_I}{1 - \mu_I}\right]^{-1} \end{aligned} \tag{4}$$

Otherwise, when (3) is reversed, then (2) becomes

$$\begin{aligned}
& \mu_I P(d \mid I)[-c_I + s] + (1-\mu_I)P(d \mid H)[-c_H] \geq 0 \\
\iff & \frac{\mu_I}{1-\mu_I} P(d \mid I) - P(d \mid H)\frac{c_H}{s-c_I} \geq 0 \\
\iff & \frac{P(d \mid I)}{P(d \mid H)} \geq \frac{c_H}{s-c_I}\left[\frac{\mu_I}{1-\mu_I}\right]^{-1} > 1 \\
\implies & P(d \mid I) \geq P(d \mid H)
\end{aligned}$$

Which seems sensible.

Summarizing NIAS Conditions To summarize, we are able to write two conditions that, depending on parameters, are necessary for NIAS.

$$\frac{\mu_I}{1-\mu_I} > \frac{c_H}{s-c_I} \implies \frac{P(d \mid I)}{P(d \mid H)} > \frac{c_H}{s-c_I}\left[\frac{\mu_I}{1-\mu_I}\right]^{-1} \tag{5a}$$

$$\frac{\mu_I}{1-\mu_I} < \frac{c_H}{s-c_I} \implies P(d \mid I) \geq P(d \mid H) \tag{5b}$$

In other words, if our data *do not* satisfy these conditions, then we can reject NIAS.

Interpreting NIAS Conditions What are conditions (5) saying? The quantity $s - c_I$ is the additional pain from staying home when your kid is sick. The quantity c_H is how much you save from staying home if your kid is healthy. So maybe we vastly oversimplify and call $c_H/(s - c_I)$ the relative benefit of staying home.

- In the second case, (5b), the prior likelihood of a sick child is relatively low compared to the benefit of staying home. In this case, NIAS predicts that we should expect to see fewer parents taking their healthy kids to the doctor. This makes sense. Just keep reading it to yourself and nod.
- The first case, (5a), is harder to interpret as it is. Note however, that its RHS

$$\frac{c_H(1-\mu_I)}{(s-c_I)\mu_I}$$

  is the ex-ante relative expected "benefit of staying home." So, (5a) tells us that the higher the ex-ante relative expected "benefit of staying home," the less likely you should be to take your healthy kid to the hospital.[1]
- Interestingly, NIAS is also telling us which implication to look for. That is, when we see that the prior likelihood of a sick child is low compared to the benefit of staying home,(5a), then we should just see more sick kids at the hospital. On the other hand, when people start to think that probably their kid is sick, then they have to start thinking more about costs, so the predictions aren't immediate.

NIAC: Derivation In the notation of Caplin and Dean, the following seems reasonable:

$$\{\gamma \in \Gamma(\overline{\pi}_A)\} = \{\gamma \in \Gamma \text{ s.t. } \exists a \in \text{Supp}(P_A) : \overline{\gamma}_A^a = \gamma\} = \{\overline{\gamma}_A^a : a \in A\}. \tag{6}$$

Further, recall that:

$$\overline{\gamma}_A^a(\omega) \equiv P(\omega \mid a \text{ chosen from } A) = \frac{\mu(\omega)P_A(a \mid \omega)}{\sum_{\omega' \in \Omega} \mu(\omega')P_A(a \mid \omega')} \tag{7}$$

[1] This is the converse of the first sentence I had, which made less sense, but follows the equation: "the more likely you should be to take your sick kid to the hospital."

which is just Bayes' rule and the law of total probability. Finally, letting $\overset{i}{=}$ mean "intuitive equality," we have:

$$\begin{aligned}
\operatorname*{argmax}_{a' \in A} \sum_{\omega \in \Omega} \overline{\gamma}^a u(a(\omega)) &= \operatorname*{argmax}_{a' \in A} \sum_{\omega \in \Omega} P(\omega \mid a \text{ chosen}) u(a'(\omega)) \\
&\overset{i}{=} \operatorname*{argmax}_{a' \in A} \mathbb{E}[u(a'(\omega)) \mid a \text{ chosen}] \\
&\overset{i}{=} a \qquad (8)
\end{aligned}$$

Therefore, we can write

$$\begin{aligned}
G(A, \overline{\pi}) &= \sum_{\gamma \in \Gamma(\overline{\pi})} \left[\sum_{\omega \in \Omega} \mu(\omega) \overline{\pi}(\gamma \mid \omega) \right] \left[\max_{a' \in A} \sum_{\omega \in \Omega} \gamma(\omega) u(a'(\omega)) \right] \\
&= \sum_{a \in A} \left[\sum_{\omega \in \Omega} \mu(\omega) P_A(a \mid \omega) \right] \left[\max_{a' \in A} \sum_{\omega \in \Omega} \overline{\gamma}^a(\omega) u(a'(\omega)) \right] && \text{by (6)} \\
&= \sum_{a \in A} \left[\sum_{\omega \in \Omega} \mu(\omega) P_A(a \mid \omega) \right] \left[\sum_{\omega \in \Omega} \overline{\gamma}^a(\omega) u(a(\omega)) \right] && \text{by (8)} \\
&= \sum_{a \in A} \left[\sum_{\omega \in \Omega} \mu(\omega) P_A(a \mid \omega) \right] \left[\sum_{\omega \in \Omega} \frac{\mu(\omega) P_A(a \mid \omega)}{\sum_{\omega' \in \Omega} \mu(\omega') P_A(a \mid \omega')} u(a(\omega)) \right] && \text{by (7)} \\
&= \sum_{a \in A} \left[\sum_{\omega \in \Omega} \mu(\omega) P_A(a \mid \omega) u(a(\omega)) \right]
\end{aligned}$$

Plugging in the parameters of this problem (and letting N, S be the actions under no subsidy and subsidy, respectively) yields

$$\begin{aligned}
G(A_S, \overline{\pi}_i) &= \mu_I P_i(d \mid I)(-c_I) + (1 - \mu_I) P_i(d \mid H)(-c_H) + \mu_I P_i(n \mid I)(-s) \\
G(A_N, \overline{\pi}_i) &= \mu_I P_i(d \mid I)(-\overline{c}_I) + (1 - \mu_I) P_i(d \mid H)(-\overline{c}_H) + \mu_I P_i(n \mid I)(-s)
\end{aligned}$$

for $i \in \{N, S\}$. Therefore NIAC holds iff:

$$\begin{aligned}
& G(A_S, \overline{\pi}_S) - G(A_S, \overline{\pi}_N) + G(A_N, \overline{\pi}_N) - G(A_N, \overline{\pi}_S) \geq 0 \\
\iff & \mu_I [P_N(d \mid I) - P_S(d \mid I)] \{(-c_I + s) - (-\overline{c}_I + s)\} \\
& + (1 - \mu_I)[P_N(n \mid H) - P_S(n \mid H)] \{c_H - \overline{c}_H\} \geq 0.
\end{aligned}$$

Minor rearranging yields the slightly more-interpretable

$$\mu_I \underbrace{[P_S(d \mid I) - P_N(d \mid I)]}_{\text{Change in correct visits}} \underbrace{(c_I - \overline{c}_I)}_{\text{Ill Sub.}} \geq (1 - \mu_I) \underbrace{[P_N(d \mid H) - P_S(d \mid H)]}_{\text{-Change in false positives}} \underbrace{(c_H - \overline{c}_H)}_{\text{Healthy Sub.}}$$

So, if our data can validate or invalidate NIAC if it satisfies this condition. But, what does this condition mean? Good intuition here requires thinking through some cases. The problem is that we don't have a prediction for how the subsidy affects the conditional probability of going to the hospital, regardless of the state. That is, we don't know whether the two terms labeled "Change" are positive or negative. Let's two consider two representative cases. In what follows, let $(+, -)$ mean that the first term on the LHS is positive, and the first term on the RHS is negative. We think through cases in which $\mu_I \geq \frac{1}{2}$ because it gives cleaner predictions (they can just drop out of the equation if this is the case and we have the same inequality—note, however, that unless $\mu = 0.5$, then this condition can only *in*validate NIAC).

$(+, -)$ In this case, the subsidy increases the conditional probability of going to the hospital, regardless of the state. NIAC is trivially satisfied, since the LHS $> 0 >$ RHS.

$(+,+)$ Now, the subsidy induces more correct visits and a decrease in false positives. If the subsidies were in the same amount, (i.e., $\delta \equiv \frac{c_H - \bar{c}_H}{c_I - \bar{c}_I} = 1$) then we should see increase in correct visits be larger than the decrease in false positives. If the healthy subsidy is larger ($\delta > 1$), then we should see a larger increase in correct visits than a decrease in false negatives. That is, false negatives will not go down by as much as correct visits will go up. This, indeed, makes sense. If, finally, $\delta < 1$, then we actually cannot make a prediction about the relative size.

Combining NIAS and NIAC Recall that in the case that 0 is the max in the NIAS condition (2), we have:

$$P_S(d \mid I) \geq P_S(d \mid H) \qquad\qquad P_N(d \mid I) \geq P_N(d \mid H) \tag{9}$$

Note, further, that the bound in (2) is weakly higher under the subsidy. So, if we fall into the case in which 0 is the max, then we can combine (9) with the NIAC condition to get:

$$\begin{aligned}
[P_S(d \mid I) - P_N(d \mid H)](c_I - \bar{c}_I) &\geq [P_S(d \mid I) - P_N(d \mid I)](c_I - \bar{c}_I) \\
&\geq [P_N(d \mid H) - P_S(d \mid H)](c_H - \bar{c}_H) \\
&\geq [P_N(d \mid H) - P_S(d \mid I)](c_H - \bar{c}_H) \\
\implies P_S(d \mid I) \geq P_N(d \mid H).
\end{aligned}$$

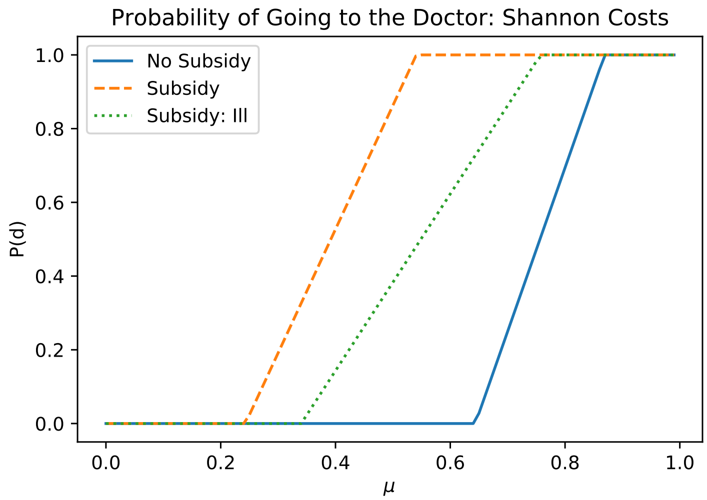
That is, with the subsidy in place, there should be more true positives than there were false positives before the subsidy. This is not necessarily an insightful prediction, but one that can be tested. Similar manipulations would yield $P_N(n \mid H) \geq P_S(n \mid I)$.

2. We know from class that the dataset is consistent with the Shannon costs model iff

$$\begin{aligned}
P(n) = 0, P(d) = 1 &\iff 1 \geq \mu_I \exp\left(\frac{c_I - s}{\lambda}\right) + (1 - \mu_I)\exp\left(\frac{c_H}{\lambda}\right) \\
P(n) = 1, P(d) = 0 &\iff 1 \geq \mu_I \exp\left(\frac{s - c_I}{\lambda}\right) + (1 - \mu_I)\exp\left(\frac{-c_H}{\lambda}\right) \\
P(n) > 0, P(d) > 0 &\iff 0 = \mu_I \cdot \frac{\exp(-{}^{s}/_{\lambda}) - \exp(-{}^{c_I}/_{\lambda})}{P(n)\exp(-{}^{s}/_{\lambda}) + P(d)\exp(-{}^{c_I}/_{\lambda})} \\
&\qquad + (1 - \mu_I) \cdot \frac{1 - \exp(-{}^{c_H}/_{\lambda})}{P(n) + P(d)\exp(-{}^{c_H}/_{\lambda})},
\end{aligned}$$

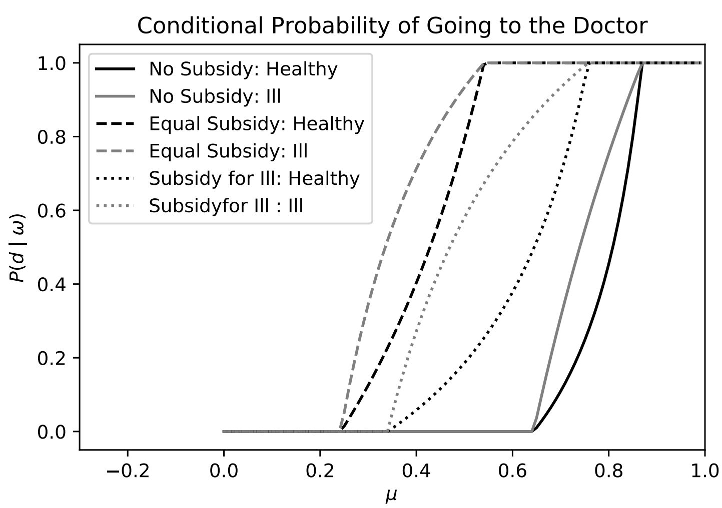
and the MM conditional probabilities hold. The first thing to notice here is that the first two conditions put, respectively, a lower and upper bound on μ_I. In the first case, since $\exp({}^{c_H}/_{\lambda}) > 1$ and $\exp({}^{c_I - s}/_{\lambda} < 1$, then μ has to be high so that the first term dominates. The second case is analogous. What is this result saying? It is saying that if we see d played with probability 1, then you better have a pretty high μ_I—that is, you better be pretty sure that your kid is sick. Furthermore, the lower bound can easily be seen to be decreasing in costs, and the upper bound is also decreasing in costs. So, under a subsidy, there are more priors consistent with parents taking their kid to the hospital.

The third condition implicitly defines $P(d)$ as a function of μ. Mathematica makes the function explicit, and Python plots it. The blue line presents the function with no subsidies; the orange dashed line presents a case in which the subsidies for all types of healthcare are the same; and the green dotted line presents a case in which the subsidy for sick kids is larger than the subsidy for healthy kids.



So, the probability that a relatively uncertain parent (precisely, a parent with an intermediate-range prior) increases with their prior. Another nice prediction. Finally, as discussed in the section on bounds, subsidies increase the range of priors over which we should expect parents to take their kids to the doctor.

There are also predictions about the conditional probabilities. In particular, as we can see from the graph below, when the child is actually healthy, the conditional probability of taking the kid to the hospital should initially increase more slowly than if the kid were sick. This relationship appears to be more pronounced the higher the wedge is between costs for sick vs. healthy patients—this can be seen by noticing that the area between the dotted lines is bigger than the area between the solid lines and the area for an equal subsidy.



3. If the new information just confirms the prior (that is, pushes it more to an extreme), then we do not expect any different implications than what we saw above.**HELP**

In [1]:
```
####### Aggregate Data Treatment


## Assumption: each individual faces each and every one of the different decision problems

%reset
import pandas as pd
import numpy as np
import itertools
from scipy import stats


#######################################################


def ttest2s(x):
    if np.std(x) == 0:
        if np.mean(x) == 0:
            print('Elements in the vector are all zero')
            return (np.nan,1)
        if np.mean(x) != 0:
            print('Elements in the vector are all the same and different from zero')
            return (np.nan,0)
    if (np.std(x) != 0):
        n = len(x)
        tt = np.mean(x)/(np.std(x)/np.sqrt(n))
        pval = stats.t.sf(np.abs(tt), n-1)*2
        return (tt,pval)
def ttest1s(x):
    if np.std(x) == 0:
        if np.mean(x) == 0:
            print('Elements in the vector are all zero')
            return (np.nan,1)
        if np.mean(x) != 0:
            print('Elements in the vector are all the same and different from zero')
            return (np.nan,0)
    if (np.std(x) != 0):
        n = len(x)
        tt = np.mean(x)/(np.std(x)/np.sqrt(n))
        pval = stats.t.sf(tt, n-1)
        return (tt,pval)

#######################################################




df = pd.read_csv('Data_for_HW_1.csv')
df['User ID'] = 1
df.loc[df['State'] == 1,'State'] = 0
df.loc[df['State'] == 2,'State'] = 1
df.loc[df['State'] == 5,'State'] = 2
df.loc[df['State'] == 6,'State'] = 3
df['Chosen Act'] = 0
df.loc[
    (df['Chosen Action'] == 11) |
    (df['Chosen Action'] == 13) |
    (df['Chosen Action'] == 15) |
    (df['Chosen Action'] == 17), 'Chosen Act'] = 1

dict_u = {8: [[1,0,10,0],[0,1,0,10]],
          9: [[10,0,1,0],[0,10,0,1]],
          10: [[1,0,1,0],[0,1,0,1]],
          11: [[10,0,10,0],[0,10,0,10]]}


#######################################################


Q8  = [10,11]
Q9  = [12,13]
Q10 = [14,15]
Q11 = [16,17]
DP = sorted(list(df['Question ID'].unique()))
```

```python
Indiv = sorted(list(df['User ID'].unique()))
Omega = sorted(list(df['State'].unique()))
mu = [1/4, 1/4, 1/4, 1/4]
df2 = pd.DataFrame(data = {'mu(s)': mu, 'State': Omega})

# Define df for revealed information structures
temp2 = []
for A in DP:
    temp = list(itertools.product(Indiv,[A]))
    temp2 = temp2 + temp
df_rinfo = pd.DataFrame(data = temp2)
df_rinfo.columns = ['User ID', 'Question ID']


#######################################################


data = [[i,A,a,s] for i in Indiv for A in DP for a in eval('Q'+str(A)) for s in Omega]
dftemp = pd.DataFrame(data=np.array(data), columns=['User ID','Question ID','Chosen Action','State'])


df_ind = df.groupby(['User ID','Question ID','State','Chosen Action','Chosen Act'])['Chosen Action'].count()
.to_frame()
df_ind.rename(columns={'Chosen Action': 'Frequency'}, inplace=True)
df_ind.reset_index(inplace=True)
df_ind = pd.merge(
            dftemp, df_ind,
            how='outer', on=['User ID','Question ID','State','Chosen Action']
            )
df_ind = df_ind.fillna(0)
df_ind = pd.merge(
            df_ind, df2,
            how='outer', on=['State']
            )
df_ind['PA(a|s)'] = df_ind['Frequency']/df_ind.groupby(['User ID','Question ID','State'])['Frequency'].trans
form('sum')

df_ind['PA(a|s)*mu(s)'] = df_ind['PA(a|s)']*df_ind['mu(s)']
df_ind['PA(s|a)'] = df_ind['PA(a|s)*mu(s)']/df_ind.groupby(['User ID','Question ID','Chosen Action'])['PA(a|
s)*mu(s)'].transform('sum')
df_ind.sort_values(by=['User ID','Question ID','Chosen Action','State'], inplace=True)
df_ind['PA(s|a)'] = df_ind['PA(s|a)'].fillna(0)

df_ind.loc[
    (df_ind['Chosen Action'] == 11) |
    (df_ind['Chosen Action'] == 13) |
    (df_ind['Chosen Action'] == 15) |
    (df_ind['Chosen Action'] == 17), 'Chosen Act'] = 1


######################################################

# Define df for revealed posteriors
df_rpost = df_rinfo.copy(deep=True)
df_ind.sort_values(by=['User ID','Question ID','Chosen Action','State'], inplace=True)

listofpost = [
    [
        [df_ind[
                (df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] == a) & (
df_ind['State'] == s)
            ]['PA(s|a)'].tolist()[0]
            for s in Omega]
        for a in eval('Q'+str(A))] for A in DP for i in Indiv]

listofinfo = [
    [
        [df_ind[
                (df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] == a) & (
df_ind['State'] == s)
            ]['PA(a|s)'].tolist()[0]
            for s in Omega]
        for a in eval('Q'+str(A))] for A in DP for i in Indiv]

df_rpost['gammaA(s)'] = listofpost
df_rinfo['piA(gammaA|s)'] = listofinfo
```

```
df_temp = df_rpost.set_index(['User ID','Question ID'])
dict_gammaA = df_temp['gammaA(s)'].T.to_dict()
df_temp = df_rinfo.set_index(['User ID','Question ID'])
dict_piA = df_temp['piA(gammaA|s)'].T.to_dict()

############################################

def Gvalue(infopi,priormu,postgamma,u):
    G=priormu*infopi.T*((postgamma*u.T).max(0)).T
    return G

############################################

NIAC_Gsum = []
NIAC_Gsumtilde = []
for i in Indiv:
    listDP = sorted(df_rinfo[df_rinfo['User ID']==i]['Question ID'].unique().tolist())
    DPtuples = list(list(itertools.permutations(listDP, x)) for x in range(2,len(listDP)+1))
    DPtuples = [item for sublist in DPtuples for item in sublist]
    NIAC_Gsumtemp = [0 for x in range(0,len(DPtuples))]
    NIAC_Gsumtildetemp = [0 for x in range(0,len(DPtuples))]
    index = 0
    for subtuple in DPtuples:
        Gsum = 0
        Gsumtilde = 0
        for node in range(0,len(subtuple)):
            Gsum = Gsum + Gvalue(np.matrix(dict_piA[(i,subtuple[node])]),mu,np.matrix(dict_gammaA[(i,subtupl
e[node])]),np.matrix(dict_u[subtuple[node]]))
            if node != len(subtuple)-1:
                Gsumtilde = Gsumtilde + Gvalue(np.matrix(dict_piA[(i,subtuple[node+1])]),mu,np.matrix(dict_g
ammaA[(i,subtuple[node+1])]),np.matrix(dict_u[subtuple[node]]))
            else:
                Gsumtilde = Gsumtilde + Gvalue(np.matrix(dict_piA[(i,subtuple[0])]),mu,np.matrix(dict_gammaA
[(i,subtuple[0])]),np.matrix(dict_u[subtuple[node]]))
        NIAC_Gsumtemp[index] = np.asscalar(Gsum)
        NIAC_Gsumtildetemp[index] = np.asscalar(Gsumtilde)
        index = index + 1
    NIAC_Gsum.append(NIAC_Gsumtemp)
    NIAC_Gsumtilde.append(NIAC_Gsumtildetemp)

############################################

print('NIAC: smallest difference')
print((np.array(NIAC_Gsum)-np.array(NIAC_Gsumtilde)).min(1))

temp = np.array(NIAC_Gsum)-np.array(NIAC_Gsumtilde)
np.shape(temp)
NIAC_ttest_indiv = np.apply_along_axis(ttest1s, 1, temp)
print('NIAC: t-test for each individual (t-stat,p-value)')
print(NIAC_ttest_indiv)
NIAC_ttest_poolindiv = ttest1s(np.concatenate(temp))
print('NIAC: t-test pooling all individuals (t-stat,p-value)')
print(NIAC_ttest_poolindiv)


############################################


# Define df for NIAS
temp = [[i,A,a] for i in Indiv for A in DP for a in eval('Q'+str(A))]
temp2 = []
tempu = []
counter = 0
for [i,A,a] in temp:
    condprob = []
    for s in Omega:
        if len(df_ind[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] ==
a) & (df_ind['State'] == s)]) != 0:
            condprob = condprob + [np.asscalar(
                df_ind[
                    (df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] == a)
 & (df_ind['State'] == s)
                ]['PA(a|s)'])]
        else:
            condprob = condprob + [0]
```

```
    if a in (10,12,14,16):
        tempu = tempu + [dict_u[A][0]]
    else:
        tempu = tempu + [dict_u[A][1]]
    temp2 = temp2 + [condprob]

temp3 = [temp[x]+[mu]+[temp2[x]]+[tempu[x]] for x in range(0,len(temp))]
df_nias = pd.DataFrame(data = temp3)
df_nias.columns = ['User ID', 'Question ID', 'Chosen Action', 'mu','PA(a|.)', 'uA']


def NIAS_comp(row):
    if row['Chosen Action'] in (10, 12, 14, 16):
        temp = 'bla'
        mu = np.matrix(row['mu'])
        temp = np.multiply(np.matrix(row['mu']),np.matrix(row['PA(a|.)']))*(np.matrix(dict_u[row['Question I
D']][0])-np.matrix(dict_u[row['Question ID']][1])).T
    else:
        temp = np.multiply(np.matrix(row['mu']),np.matrix(row['PA(a|.)']))*(np.matrix(dict_u[row['Question I
D']][1])-np.matrix(dict_u[row['Question ID']][0])).T
    return np.asscalar(temp)

df_nias['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]'] = df_nias.apply(NIAS_comp, axis =1)
df_nias[df_nias['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']<0]


print('NIAS')
print(df_nias[['User ID', 'Question ID', 'Chosen Action', 'Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']])
print('Individuals failing NIAS deterministically')
print(df_nias[df_nias['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']<0][['User ID', 'Question ID', 'Chosen Action',
'Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']])

temp = [df_nias[df_nias['User ID'] == userid]['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]'].values.tolist() for user
id in Indiv]
np.shape(temp)
NIAS_ttest_indiv = np.apply_along_axis(ttest1s, 1, temp)
print('NIAS: t-test for each individual (t-stat,p-value)')
print(NIAS_ttest_indiv)
NIAS_ttest_poolindiv = ttest1s(np.concatenate(temp))
print('NIAS: t-test pooling all individuals (t-stat,p-value)')
print(NIAS_ttest_poolindiv)





############################################


# Define df for Shannon
df_ind['PA(a)'] = df_ind.groupby(['User ID','Question ID','Chosen Action'])['Frequency'].transform('sum')/df
_ind.groupby(['User ID','Question ID'])['Frequency'].transform('sum')
df_ind['u(a(s))'] = 0
df_ind['u(b(s))'] = 0
df_ind['PA(s|b)'] = 0
temp = [[A,a,s] for A in DP for a in [0,1] for s in Omega]
counter = 0
for [A,a,s] in temp:
    df_ind.loc[(df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a) & (df_ind['State'] == s),'u(a(s))'
] = dict_u[A][a][s]
    df_ind.loc[(df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a) & (df_ind['State'] == s),'u(b(s))'
] = dict_u[A][a-1][s]
    for i in Indiv:
        if a == 0:
            df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a)
 & (df_ind['State'] == s),'PA(s|b)'] = df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) &
(df_ind['Chosen Act'] == 1) & (df_ind['State'] == s),'PA(s|a)'].values[0]
        if a == 1:
            df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a)
 & (df_ind['State'] == s),'PA(s|b)'] = df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) &
(df_ind['Chosen Act'] == 0) & (df_ind['State'] == s),'PA(s|a)'].values[0]

df_shannon = df_ind.copy(deep=True)

# Obtaining lambda from MM conditions, avg lambda per individual, aggregate avg lambda
df_shannon['lambda'] = (df_shannon['u(a(s))'] - df_shannon['u(b(s))'])/(df_shannon['PA(s|a)'].apply(np.log)-
```

```
df_shannon['PA(s|b)'].apply(np.log))
df_shannon.loc[(df_shannon['PA(s|a)'] == 0) | (df_shannon['PA(s|b)'] == 0),'lambda'] = np.nan
df_shannon['avglambda i'] = df_shannon.groupby(['User ID'])['lambda'].transform('mean')
df_shannon['avglambda'] = df_shannon['lambda'].apply('mean')

# Obtaining difference for necessary and sufficient conditions for Shannon
df_shannon['z(a,s)'] = (df_shannon['u(a(s))']/df_shannon['lambda'])
df_shannon['z(a,s)'] = df_shannon['z(a,s)'].apply(np.exp)
df_shannon['z(b,s)'] = (df_shannon['u(b(s))']/df_shannon['lambda'])
df_shannon['z(b,s)'] = df_shannon['z(b,s)'].apply(np.exp)
df_shannon['Shannon NSCond'] = df_shannon['mu(s)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z
(a,s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)'])
df_shannon['Shannon NSCond'] = df_shannon.groupby(['User ID','Question ID','Chosen Action'])['Shannon NSCon
d'].transform('sum')-1
df_shannon['MMCond'] = df_shannon['PA(a)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z(a,s)']+(1
-df_shannon['PA(a)'])*df_shannon['z(b,s)']) - df_shannon['PA(a|s)']
df_shannon.loc[(df_shannon['PA(a)'] == 1) | (df_shannon['PA(a)'] == 0),'MMCond'] = 0


df_shannon['z(a,s)'] = (df_shannon['u(a(s))']/df_shannon['avglambda i'])
df_shannon['z(a,s)'] = df_shannon['z(a,s)'].apply(np.exp)
df_shannon['z(b,s)'] = (df_shannon['u(b(s))']/df_shannon['avglambda i'])
df_shannon['z(b,s)'] = df_shannon['z(b,s)'].apply(np.exp)
df_shannon['Shannon NSCond avg i'] = df_shannon['mu(s)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shanno
n['z(a,s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)'])
df_shannon['Shannon NSCond avg i'] = df_shannon.groupby(['User ID','Question ID','Chosen Action'])['Shannon
 NSCond avg i'].transform('sum')-1
df_shannon['MMCond avg i'] = df_shannon['PA(a)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z(a,
s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)']) - df_shannon['PA(a|s)']
df_shannon.loc[(df_shannon['PA(a)'] == 1) | (df_shannon['PA(a)'] == 0),'MMCond avg i'] = 0

df_shannon['z(a,s)'] = (df_shannon['u(a(s))']/df_shannon['avglambda'])
df_shannon['z(a,s)'] = df_shannon['z(a,s)'].apply(np.exp)
df_shannon['z(b,s)'] = (df_shannon['u(b(s))']/df_shannon['avglambda'])
df_shannon['z(b,s)'] = df_shannon['z(b,s)'].apply(np.exp)
df_shannon['Shannon NSCond avg'] = df_shannon['mu(s)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon[
'z(a,s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)'])
df_shannon['Shannon NSCond avg'] = df_shannon.groupby(['User ID','Question ID','Chosen Action'])['Shannon NS
Cond avg'].transform('sum')-1
df_shannon['MMCond avg'] = df_shannon['PA(a)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z(a,s)'
]+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)']) - df_shannon['PA(a|s)']
df_shannon.loc[(df_shannon['PA(a)'] == 1) | (df_shannon['PA(a)'] == 0),'MMCond avg'] = 0

df_shannon.drop('z(a,s)', axis=1, inplace=True)
df_shannon.drop('z(b,s)', axis=1, inplace=True)



# Testing necessary and sufficient conditions for Shannon; self-explanatory
temp = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['PA(a)'] > 0) & (df_shannon['State'] == 0
)]['Shannon NSCond avg i'].tolist() for userid in Indiv]
temp2 = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['State'] == 0)]['Shannon NSCond avg i'].
tolist() for userid in Indiv]
temp3 = [df_shannon[(df_shannon['User ID'] == userid)]['MMCond avg i'].tolist() for userid in Indiv]

Shannon_ttest_indiv1 = np.array([ttest2s(temp[indiv]) for indiv in range(0,len(temp)-1)])
Shannon_ttest_indiv2 = np.array([ttest1s(temp2[indiv]) for indiv in range(0,len(temp2)-1)])
Shannon_ttest_indiv3 = np.array([ttest2s(temp3[indiv]) for indiv in range(0,len(temp3)-1)])
print('Shannon: t-test for each individual allowing for heterog. cost functions (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_ttest_indiv1)
print('For all a; H0: cond <= 0')
print(Shannon_ttest_indiv2)
print('MM; H0: cond == 0')
print(Shannon_ttest_indiv3)

Shannon_ttest_poolindiv1 = ttest2s([x for x in np.concatenate(temp) if str(x) != 'nan'])
Shannon_ttest_poolindiv2 = ttest1s([x for x in np.concatenate(temp2) if str(x) != 'nan'])
Shannon_ttest_poolindiv3 = ttest2s([x for x in np.concatenate(temp3) if str(x) != 'nan'])
print('Shannon: t-test pooling all individuals allowing for heterog. cost functions (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_ttest_poolindiv1)
print('For all a; H0: cond <= 0')
print(Shannon_ttest_poolindiv2)
print('MM; H0: cond == 0')
print(Shannon_ttest_poolindiv3)
```

```
temp = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['PA(a)'] > 0) & (df_shannon['State'] == 0
)]['Shannon NSCond avg'].tolist() for userid in Indiv]
temp2 = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['State'] == 0)]['Shannon NSCond avg'].to
list() for userid in Indiv]
temp3 = [df_shannon[(df_shannon['User ID'] == userid)]['MMCond avg'].tolist() for userid in Indiv]

Shannon_avg_ttest_indiv1 = np.array([ttest2s(temp[indiv]) for indiv in range(0,len(temp)-1)])
Shannon_avg_ttest_indiv2 = np.array([ttest1s(temp2[indiv]) for indiv in range(0,len(temp2)-1)])
Shannon_avg_ttest_indiv3 = np.array([ttest2s(temp3[indiv]) for indiv in range(0,len(temp3)-1)])
print('Shannon: t-test for each individual with same cost function (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_avg_ttest_indiv1)
print('For all a; H0: cond <= 0')
print(Shannon_avg_ttest_indiv2)
print('MM; H0: cond == 0')
print(Shannon_avg_ttest_indiv3)

Shannon_avg_ttest_poolindiv1 = ttest2s([x for x in np.concatenate(temp) if str(x) != 'nan'])
Shannon_avg_ttest_poolindiv2 = ttest1s([x for x in np.concatenate(temp2) if str(x) != 'nan'])
Shannon_avg_ttest_poolindiv3 = ttest2s([x for x in np.concatenate(temp3) if str(x) != 'nan'])
print('Shannon: t-test pooling all individuals allwith same cost function (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_avg_ttest_poolindiv1)
print('For all a; H0: cond <= 0')
print(Shannon_avg_ttest_poolindiv2)
print('MM; H0: cond == 0')
print(Shannon_avg_ttest_poolindiv3)
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
NIAC: smallest difference
[ 0.12665918]
NIAC: t-test for each individual (t-stat,p-value)
[[  2.22046525e+01   1.22797645e-30]]
NIAC: t-test pooling all individuals (t-stat,p-value)
(22.204652492136876, 1.2279764450012158e-30)
NIAS
   User ID  Question ID  Chosen Action  Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]
0        1            8             10                            0.981533
1        1            8             11                            0.981533
2        1            9             12                            1.473491
3        1            9             13                            1.473491
4        1           10             14                            0.147222
5        1           10             15                            0.147222
6        1           11             16                            2.235541
7        1           11             17                            2.235541
Individuals failing NIAS deterministically
Empty DataFrame
Columns: [User ID, Question ID, Chosen Action, Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]]
Index: []
NIAS: t-test for each individual (t-stat,p-value)
[[  4.50846280e+00   1.38505701e-03]]
NIAS: t-test pooling all individuals (t-stat,p-value)
(4.5084627959076533, 0.0013850570101961055)
Shannon: t-test for each individual allowing for heterog. cost functions (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
[]
For all a; H0: cond <= 0
[]
MM; H0: cond == 0
[]
Shannon: t-test pooling all individuals allowing for heterog. cost functions (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
(0.23872814552870497, 0.8181547337037739)
For all a; H0: cond <= 0
(0.23872814552870497, 0.40907736685188695)
MM; H0: cond == 0
(2.5447815089931843e-16, 0.99999999999999978)
Shannon: t-test for each individual with same cost function (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
[]
For all a; H0: cond <= 0
[]
MM; H0: cond == 0
[]
Shannon: t-test pooling all individuals allwith same cost function (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
(0.23872814552870497, 0.8181547337037739)
For all a; H0: cond <= 0
(0.23872814552870497, 0.40907736685188695)
MM; H0: cond == 0
(2.5447815089931843e-16, 0.99999999999999978)
```

In [2]:

```
####### Individual Data Treatment


## Assumption: each individual faces each and every one of the different decision problems

%reset
import pandas as pd
import numpy as np
import itertools
from scipy import stats


######################################################


def ttest2s(x):
    if np.std(x) == 0:
        if np.mean(x) == 0:
            print('Elements in the vector are all zero')
            return (np.nan,1)
        if np.mean(x) != 0:
            print('Elements in the vector are all the same and different from zero')
```

```
            return (np.nan,0)
    if (np.std(x) != 0):
        n = len(x)
        tt = np.mean(x)/(np.std(x)/np.sqrt(n))
        pval = stats.t.sf(np.abs(tt), n-1)*2
        return (tt,pval)
def ttest1s(x):
    if np.std(x) == 0:
        if np.mean(x) == 0:
            print('Elements in the vector are all zero')
            return (np.nan,1)
        if np.mean(x) != 0:
            print('Elements in the vector are all the same and different from zero')
            return (np.nan,0)
    if (np.std(x) != 0):
        n = len(x)
        tt = np.mean(x)/(np.std(x)/np.sqrt(n))
        pval = stats.t.sf(tt, n-1)
        return (tt,pval)

#######################################################




df = pd.read_csv('Data_for_HW_1.csv')
df.loc[df['State'] == 1,'State'] = 0
df.loc[df['State'] == 2,'State'] = 1
df.loc[df['State'] == 5,'State'] = 2
df.loc[df['State'] == 6,'State'] = 3
df['Chosen Act'] = 0
df.loc[
    (df['Chosen Action'] == 11) |
    (df['Chosen Action'] == 13) |
    (df['Chosen Action'] == 15) |
    (df['Chosen Action'] == 17), 'Chosen Act'] = 1

dict_u = {8: [[1,0,10,0],[0,1,0,10]],
          9: [[10,0,1,0],[0,10,0,1]],
          10: [[1,0,1,0],[0,1,0,1]],
          11: [[10,0,10,0],[0,10,0,10]]}


#######################################################


Q8  = [10,11]
Q9  = [12,13]
Q10 = [14,15]
Q11 = [16,17]
DP = sorted(list(df['Question ID'].unique()))
Indiv = sorted(list(df['User ID'].unique()))
Omega = sorted(list(df['State'].unique()))
mu = [1/4, 1/4, 1/4, 1/4]
df2 = pd.DataFrame(data = {'mu(s)': mu, 'State': Omega})

# Define df for revealed information structures
temp2 = []
for A in DP:
    temp = list(itertools.product(Indiv,[A]))
    temp2 = temp2 + temp
df_rinfo = pd.DataFrame(data = temp2)
df_rinfo.columns = ['User ID', 'Question ID']


######################################################


data = [[i,A,a,s] for i in Indiv for A in DP for a in eval('Q'+str(A)) for s in Omega]
dftemp = pd.DataFrame(data=np.array(data), columns=['User ID','Question ID','Chosen Action','State'])


df_ind = df.groupby(['User ID','Question ID','State','Chosen Action','Chosen Act'])['Chosen Action'].count()
.to_frame()
df_ind.rename(columns={'Chosen Action': 'Frequency'}, inplace=True)
df_ind.reset_index(inplace=True)
```

```
df_ind = pd.merge(
            dftemp, df_ind,
            how='outer', on=['User ID','Question ID','State','Chosen Action']
            )
df_ind = df_ind.fillna(0)
df_ind = pd.merge(
            df_ind, df2,
            how='outer', on=['State']
            )
df_ind['PA(a|s)'] = df_ind['Frequency']/df_ind.groupby(['User ID','Question ID','State'])['Frequency'].trans
form('sum')

df_ind['PA(a|s)*mu(s)'] = df_ind['PA(a|s)']*df_ind['mu(s)']
df_ind['PA(s|a)'] = df_ind['PA(a|s)*mu(s)']/df_ind.groupby(['User ID','Question ID','Chosen Action'])['PA(a|
s)*mu(s)'].transform('sum')
df_ind.sort_values(by=['User ID','Question ID','Chosen Action','State'], inplace=True)
df_ind['PA(s|a)'] = df_ind['PA(s|a)'].fillna(0)

df_ind.loc[
    (df_ind['Chosen Action'] == 11) |
    (df_ind['Chosen Action'] == 13) |
    (df_ind['Chosen Action'] == 15) |
    (df_ind['Chosen Action'] == 17), 'Chosen Act'] = 1


#####################################################

# Define df for revealed posteriors
df_rpost = df_rinfo.copy(deep=True)
df_ind.sort_values(by=['User ID','Question ID','Chosen Action','State'], inplace=True)

listofpost = [
    [
        [df_ind[
                (df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] == a) & (
df_ind['State'] == s)
            ]['PA(s|a)'].tolist()[0]
            for s in Omega]
        for a in eval('Q'+str(A))] for A in DP for i in Indiv]

listofinfo = [
    [
        [df_ind[
                (df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] == a) & (
df_ind['State'] == s)
            ]['PA(a|s)'].tolist()[0]
            for s in Omega]
        for a in eval('Q'+str(A))] for A in DP for i in Indiv]

df_rpost['gammaA(s)'] = listofpost
df_rinfo['piA(gammaA|s)'] = listofinfo

df_temp = df_rpost.set_index(['User ID','Question ID'])
dict_gammaA = df_temp['gammaA(s)'].T.to_dict()
df_temp = df_rinfo.set_index(['User ID','Question ID'])
dict_piA = df_temp['piA(gammaA|s)'].T.to_dict()

############################################

def Gvalue(infopi,priormu,postgamma,u):
    G=priormu*infopi.T*((postgamma*u.T).max(0)).T
    return G

############################################

NIAC_Gsum = []
NIAC_Gsumtilde = []
for i in Indiv:
    listDP = sorted(df_rinfo[df_rinfo['User ID']==i]['Question ID'].unique().tolist())
    DPtuples = list(list(itertools.permutations(listDP, x)) for x in range(2,len(listDP)+1))
    DPtuples = [item for sublist in DPtuples for item in sublist]
    NIAC_Gsumtemp = [0 for x in range(0,len(DPtuples))]
    NIAC_Gsumtildetemp = [0 for x in range(0,len(DPtuples))]
    index = 0
    for subtuple in DPtuples:
        Gsum = 0
```

```
        Gsumtilde = 0
        for node in range(0,len(subtuple)):
            Gsum = Gsum + Gvalue(np.matrix(dict_piA[(i,subtuple[node])]),mu,np.matrix(dict_gammaA[(i,subtupl
e[node])]),np.matrix(dict_u[subtuple[node]]))
            if node != len(subtuple)-1:
                Gsumtilde = Gsumtilde + Gvalue(np.matrix(dict_piA[(i,subtuple[node+1])]),mu,np.matrix(dict_g
ammaA[(i,subtuple[node+1])]),np.matrix(dict_u[subtuple[node]]))
            else:
                Gsumtilde = Gsumtilde + Gvalue(np.matrix(dict_piA[(i,subtuple[0])]),mu,np.matrix(dict_gammaA
[(i,subtuple[0])]),np.matrix(dict_u[subtuple[node]]))
        NIAC_Gsumtemp[index] = np.asscalar(Gsum)
        NIAC_Gsumtildetemp[index] = np.asscalar(Gsumtilde)
        index = index + 1
    NIAC_Gsum.append(NIAC_Gsumtemp)
    NIAC_Gsumtilde.append(NIAC_Gsumtildetemp)

############################################

print('NIAC: smallest difference')
print((np.array(NIAC_Gsum)-np.array(NIAC_Gsumtilde)).min(1))

temp = np.array(NIAC_Gsum)-np.array(NIAC_Gsumtilde)
np.shape(temp)
NIAC_ttest_indiv = np.apply_along_axis(ttest1s, 1, temp)
print('NIAC: t-test for each individual (t-stat,p-value)')
print(NIAC_ttest_indiv)
NIAC_ttest_poolindiv = ttest1s(np.concatenate(temp))
print('NIAC: t-test pooling all individuals (t-stat,p-value)')
print(NIAC_ttest_poolindiv)


############################################


# Define df for NIAS
temp = [[i,A,a] for i in Indiv for A in DP for a in eval('Q'+str(A))]
temp2 = []
tempu = []
counter = 0
for [i,A,a] in temp:
    condprob = []
    for s in Omega:
        if len(df_ind[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] ==
a) & (df_ind['State'] == s)]) != 0:
            condprob = condprob + [np.asscalar(
                df_ind[
                    (df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Action'] == a)
 & (df_ind['State'] == s)
                ]['PA(a|s)'])]
        else:
            condprob = condprob + [0]
    if a in (10,12,14,16):
        tempu = tempu + [dict_u[A][0]]
    else:
        tempu = tempu + [dict_u[A][1]]
    temp2 = temp2 + [condprob]

temp3 = [temp[x]+[mu]+[temp2[x]]+[tempu[x]] for x in range(0,len(temp))]
df_nias = pd.DataFrame(data = temp3)
df_nias.columns = ['User ID', 'Question ID', 'Chosen Action', 'mu','PA(a|.)', 'uA']


def NIAS_comp(row):
    if row['Chosen Action'] in (10, 12, 14, 16):
        temp = 'bla'
        mu = np.matrix(row['mu'])
        temp = np.multiply(np.matrix(row['mu']),np.matrix(row['PA(a|.)']))*(np.matrix(dict_u[row['Question I
D']][0])-np.matrix(dict_u[row['Question ID']][1])).T
    else:
        temp = np.multiply(np.matrix(row['mu']),np.matrix(row['PA(a|.)']))*(np.matrix(dict_u[row['Question I
D']][1])-np.matrix(dict_u[row['Question ID']][0])).T
    return np.asscalar(temp)

df_nias['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]'] = df_nias.apply(NIAS_comp, axis =1)
df_nias[df_nias['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']<0]
```

```
print('NIAS')
print(df_nias[['User ID', 'Question ID', 'Chosen Action', 'Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']])
print('Individuals failing NIAS deterministically')
print(df_nias[df_nias['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']<0][['User ID', 'Question ID', 'Chosen Action',
'Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]']])

temp = [df_nias[df_nias['User ID'] == userid]['Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]'].values.tolist() for user
id in Indiv]
np.shape(temp)
NIAS_ttest_indiv = np.apply_along_axis(ttest1s, 1, temp)
print('NIAS: t-test for each individual (t-stat,p-value)')
print(NIAS_ttest_indiv)
NIAS_ttest_poolindiv = ttest1s(np.concatenate(temp))
print('NIAS: t-test pooling all individuals (t-stat,p-value)')
print(NIAS_ttest_poolindiv)


#############################################

# Define df for Shannon
df_ind['PA(a)'] = df_ind.groupby(['User ID','Question ID','Chosen Action'])['Frequency'].transform('sum')/df
_ind.groupby(['User ID','Question ID'])['Frequency'].transform('sum')
df_ind['u(a(s))'] = 0
df_ind['u(b(s))'] = 0
df_ind['PA(s|b)'] = 0
temp = [[A,a,s] for A in DP for a in [0,1] for s in Omega]
counter = 0
for [A,a,s] in temp:
    df_ind.loc[(df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a) & (df_ind['State'] == s),'u(a(s))'
] = dict_u[A][a][s]
    df_ind.loc[(df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a) & (df_ind['State'] == s),'u(b(s))'
] = dict_u[A][a-1][s]
    for i in Indiv:
        if a == 0:
            df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a)
 & (df_ind['State'] == s),'PA(s|b)'] = df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) &
(df_ind['Chosen Act'] == 1) & (df_ind['State'] == s),'PA(s|a)'].values[0]
        if a == 1:
            df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) & (df_ind['Chosen Act'] == a)
 & (df_ind['State'] == s),'PA(s|b)'] = df_ind.loc[(df_ind['User ID'] == i) & (df_ind['Question ID'] == A) &
(df_ind['Chosen Act'] == 0) & (df_ind['State'] == s),'PA(s|a)'].values[0]

df_shannon = df_ind.copy(deep=True)


# Obtaining lambda from MM conditions, avg lambda per individual, aggregate avg lambda
df_shannon['lambda'] = (df_shannon['u(a(s))'] - df_shannon['u(b(s))'])/(df_shannon['PA(s|a)'].apply(np.log)-
df_shannon['PA(s|b)'].apply(np.log))
df_shannon.loc[(df_shannon['PA(s|a)'] == 0) | (df_shannon['PA(s|b)'] == 0),'lambda'] = np.nan
df_shannon['avglambda i'] = df_shannon.groupby(['User ID'])['lambda'].transform('mean')
df_shannon['avglambda'] = df_shannon['lambda'].apply('mean')

# Obtaining difference for necessary and sufficient conditions for Shannon
df_shannon['z(a,s)'] = (df_shannon['u(a(s))']/df_shannon['lambda'])
df_shannon['z(a,s)'] = df_shannon['z(a,s)'].apply(np.exp)
df_shannon['z(b,s)'] = (df_shannon['u(b(s))']/df_shannon['lambda'])
df_shannon['z(b,s)'] = df_shannon['z(b,s)'].apply(np.exp)
df_shannon['Shannon NSCond'] = df_shannon['mu(s)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z
(a,s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)'])
df_shannon['Shannon NSCond'] = df_shannon.groupby(['User ID','Question ID','Chosen Action'])['Shannon NSCon
d'].transform('sum')-1
df_shannon['MMCond'] = df_shannon['PA(a)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z(a,s)']+(1
-df_shannon['PA(a)'])*df_shannon['z(b,s)']) - df_shannon['PA(a|s)']
df_shannon.loc[(df_shannon['PA(a)'] == 1) | (df_shannon['PA(a)'] == 0),'MMCond'] = 0


df_shannon['z(a,s)'] = (df_shannon['u(a(s))']/df_shannon['avglambda i'])
df_shannon['z(a,s)'] = df_shannon['z(a,s)'].apply(np.exp)
df_shannon['z(b,s)'] = (df_shannon['u(b(s))']/df_shannon['avglambda i'])
df_shannon['z(b,s)'] = df_shannon['z(b,s)'].apply(np.exp)
df_shannon['Shannon NSCond avg i'] = df_shannon['mu(s)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shanno
n['z(a,s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)'])
df_shannon['Shannon NSCond avg i'] = df_shannon.groupby(['User ID','Question ID','Chosen Action'])['Shannon
 NSCond avg i'].transform('sum')-1
df_shannon['MMCond avg i'] = df_shannon['PA(a)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z(a,
```

```
s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)']) - df_shannon['PA(a|s)']
df_shannon.loc[(df_shannon['PA(a)'] == 1) | (df_shannon['PA(a)'] == 0),'MMCond avg i'] = 0

df_shannon['z(a,s)'] = (df_shannon['u(a(s))']/df_shannon['avglambda'])
df_shannon['z(a,s)'] = df_shannon['z(a,s)'].apply(np.exp)
df_shannon['z(b,s)'] = (df_shannon['u(b(s))']/df_shannon['avglambda'])
df_shannon['z(b,s)'] = df_shannon['z(b,s)'].apply(np.exp)
df_shannon['Shannon NSCond avg'] = df_shannon['mu(s)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon[
'z(a,s)']+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)'])
df_shannon['Shannon NSCond avg'] = df_shannon.groupby(['User ID','Question ID','Chosen Action'])['Shannon NS
Cond avg'].transform('sum')-1
df_shannon['MMCond avg'] = df_shannon['PA(a)']*df_shannon['z(a,s)']/(df_shannon['PA(a)']*df_shannon['z(a,s)'
]+(1-df_shannon['PA(a)'])*df_shannon['z(b,s)']) - df_shannon['PA(a|s)']
df_shannon.loc[(df_shannon['PA(a)'] == 1) | (df_shannon['PA(a)'] == 0),'MMCond avg'] = 0

df_shannon.drop('z(a,s)', axis=1, inplace=True)
df_shannon.drop('z(b,s)', axis=1, inplace=True)

# Testing necessary and sufficient conditions for Shannon; self-explanatory
temp = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['PA(a)'] > 0) & (df_shannon['State'] == 0
)]['Shannon NSCond avg i'].tolist() for userid in Indiv]
temp2 = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['State'] == 0)]['Shannon NSCond avg i'].
tolist() for userid in Indiv]
temp3 = [df_shannon[(df_shannon['User ID'] == userid)]['MMCond avg i'].tolist() for userid in Indiv]

Shannon_ttest_indiv1 = np.array([ttest2s(temp[indiv]) for indiv in range(0,len(temp)-1)])
Shannon_ttest_indiv2 = np.array([ttest1s(temp2[indiv]) for indiv in range(0,len(temp2)-1)])
Shannon_ttest_indiv3 = np.array([ttest2s(temp3[indiv]) for indiv in range(0,len(temp3)-1)])
print('Shannon: t-test for each individual allowing for heterog. cost functions (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_ttest_indiv1)
print('For all a; H0: cond <= 0')
print(Shannon_ttest_indiv2)
print('MM; H0: cond == 0')
print(Shannon_ttest_indiv3)

Shannon_ttest_poolindiv1 = ttest2s([x for x in np.concatenate(temp) if str(x) != 'nan'])
Shannon_ttest_poolindiv2 = ttest1s([x for x in np.concatenate(temp2) if str(x) != 'nan'])
Shannon_ttest_poolindiv3 = ttest2s([x for x in np.concatenate(temp3) if str(x) != 'nan'])
print('Shannon: t-test pooling all individuals allowing for heterog. cost functions (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_ttest_poolindiv1)
print('For all a; H0: cond <= 0')
print(Shannon_ttest_poolindiv2)
print('MM; H0: cond == 0')
print(Shannon_ttest_poolindiv3)

temp = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['PA(a)'] > 0) & (df_shannon['State'] == 0
)]['Shannon NSCond avg'].tolist() for userid in Indiv]
temp2 = [df_shannon[(df_shannon['User ID'] == userid) & (df_shannon['State'] == 0)]['Shannon NSCond avg'].to
list() for userid in Indiv]
temp3 = [df_shannon[(df_shannon['User ID'] == userid)]['MMCond avg'].tolist() for userid in Indiv]

Shannon_avg_ttest_indiv1 = np.array([ttest2s(temp[indiv]) for indiv in range(0,len(temp)-1)])
Shannon_avg_ttest_indiv2 = np.array([ttest1s(temp2[indiv]) for indiv in range(0,len(temp2)-1)])
Shannon_avg_ttest_indiv3 = np.array([ttest2s(temp3[indiv]) for indiv in range(0,len(temp3)-1)])
print('Shannon: t-test for each individual with same cost function (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_avg_ttest_indiv1)
print('For all a; H0: cond <= 0')
print(Shannon_avg_ttest_indiv2)
print('MM; H0: cond == 0')
print(Shannon_avg_ttest_indiv3)

Shannon_avg_ttest_poolindiv1 = ttest2s([x for x in np.concatenate(temp) if str(x) != 'nan'])
Shannon_avg_ttest_poolindiv2 = ttest1s([x for x in np.concatenate(temp2) if str(x) != 'nan'])
Shannon_avg_ttest_poolindiv3 = ttest2s([x for x in np.concatenate(temp3) if str(x) != 'nan'])
print('Shannon: t-test pooling all individuals allwith same cost function (t-stat,p-value)')
print('For PA(a) > 0; H0: cond == 0')
print(Shannon_avg_ttest_poolindiv1)
print('For all a; H0: cond <= 0')
print(Shannon_avg_ttest_poolindiv2)
print('MM; H0: cond == 0')
print(Shannon_avg_ttest_poolindiv3)
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
NIAC: smallest difference
[ 0.         -0.35400752 -0.17307692 -0.225      -0.08082707 -0.34377078
 -2.02790552 -1.41344969 -3.82211795 -1.02857143 -0.25       -0.25817308
 -1.11364694 -0.85909539 -1.37965596 -2.16769481  0.125      -0.69012605
 -0.14983994  0.          0.         -0.37364166 -1.88961039 -0.32263514]
Elements in the vector are all zero
Elements in the vector are all zero
NIAC: t-test for each individual (t-stat,p-value)
[[  1.34054549e+01   7.34751394e-20]
 [  8.78885587e+00   1.28635212e-12]
 [  1.30496724e+01   2.42303473e-19]
 [  4.48519922e+00   1.71116192e-05]
 [  1.29561181e+01   3.32506071e-19]
 [  8.78152812e+00   1.32318659e-12]
 [ -7.94315638e+00   1.00000000e+00]
 [  1.16419009e+01   3.19040582e-17]
 [ -1.53482128e+01   1.00000000e+00]
 [  5.18291660e+00   1.39449465e-06]
 [  1.79867646e+01   6.31602677e-26]
 [  2.15042887e+00   1.78149181e-02]
 [ -4.94040198e+00   9.99996617e-01]
 [  4.71852142e+00   7.51016072e-06]
 [ -1.55297296e+01   1.00000000e+00]
 [ -1.15276522e+01   1.00000000e+00]
 [  1.98687567e+01   4.04000335e-28]
 [ -7.78435133e+00   1.00000000e+00]
 [  1.23799008e+01   2.39369508e-18]
 [             nan   1.00000000e+00]
 [             nan   1.00000000e+00]
 [  1.05352201e+01   1.75826592e-15]
 [ -7.73302824e+00   1.00000000e+00]
 [  5.44405810e+00   5.29310327e-07]]
NIAC: t-test pooling all individuals (t-stat,p-value)
(14.255407067995538, 1.5702422430857881e-43)
NIAS
     User ID  Question ID  Chosen Action  Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]
0        601            8             10                            -0.228365
1        601            8             11                            -0.228365
2        601            9             12                             0.000000
3        601            9             13                             0.000000
4        601           10             14                             0.000000
5        601           10             15                             0.000000
6        601           11             16                             0.000000
7        601           11             17                             0.000000
8        602            8             10                             2.598346
9        602            8             11                             2.598346
10       602            9             12                             2.675595
11       602            9             13                             2.675595
12       602           10             14                            -0.028788
13       602           10             15                            -0.028788
14       602           11             16                             3.875000
15       602           11             17                             3.875000
16       603            8             10                             2.493506
17       603            8             11                             2.493506
18       603            9             12                             2.334615
19       603            9             13                             2.334615
20       603           10             14                             0.464286
21       603           10             15                             0.464286
22       603           11             16                             5.000000
23       603           11             17                             5.000000
24       604            8             10                             2.750000
25       604            8             11                             2.750000
26       604            9             12                             2.750000
27       604            9             13                             2.750000
28       604           10             14                             0.452922
29       604           10             15                             0.452922
..       ...          ...            ...                                  ...
162      621            9             12                             0.000000
163      621            9             13                             0.000000
164      621           10             14                             0.000000
165      621           10             15                             0.000000
166      621           11             16                             0.000000
167      621           11             17                             0.000000
168      622            8             10                            -0.141667
169      622            8             11                            -0.141667
```

```
170      622           9           12                             0.625437
171      622           9           13                             0.625437
172      622          10           14                             0.084375
173      622          10           15                             0.084375
174      622          11           16                            -0.425638
175      622          11           17                            -0.425638
176      623           8           10                             2.750000
177      623           8           11                             2.750000
178      623           9           12                             0.954004
179      623           9           13                             0.954004
180      623          10           14                             0.500000
181      623          10           15                             0.500000
182      623          11           16                             4.642857
183      623          11           17                             4.642857
184      624           8           10                             0.013889
185      624           8           11                             0.013889
186      624           9           12                             2.671875
187      624           9           13                             2.671875
188      624          10           14                             0.500000
189      624          10           15                             0.500000
190      624          11           16                             4.791667
191      624          11           17                             4.791667

[192 rows x 4 columns]
Individuals failing NIAS deterministically
     User ID  Question ID  Chosen Action  Sum mu(s) PA(a|s)[u(a(s))-u(b(s))]
0        601           8           10                            -0.228365
1        601           8           11                            -0.228365
12       602          10           14                            -0.028788
13       602          10           15                            -0.028788
44       606          10           14                            -0.010746
45       606          10           15                            -0.010746
60       608          10           14                            -0.005769
61       608          10           15                            -0.005769
70       609          11           16                            -0.357143
71       609          11           17                            -0.357143
96       613           8           10                            -0.155449
97       613           8           11                            -0.155449
114      615           9           12                            -0.112013
115      615           9           13                            -0.112013
132      617          10           14                            -0.136767
133      617          10           15                            -0.136767
148      619          10           14                            -0.012987
149      619          10           15                            -0.012987
168      622           8           10                            -0.141667
169      622           8           11                            -0.141667
174      622          11           16                            -0.425638
175      622          11           17                            -0.425638
Elements in the vector are all zero
Elements in the vector are all zero
NIAS: t-test for each individual (t-stat,p-value)
[[ -1.63299316e+00   9.26755628e-01]
 [  4.52281332e+00   1.36149094e-03]
 [  4.51348647e+00   1.37675647e-03]
 [  4.97554671e+00   8.04528746e-04]
 [  2.91588324e+00   1.12356943e-02]
 [  2.07729113e+00   3.81950195e-02]
 [  1.63299316e+00   7.32443715e-02]
 [  3.46561867e+00   5.23385301e-03]
 [  1.53111946e+00   8.47985328e-02]
 [  3.30125457e+00   6.54904663e-03]
 [  3.72531683e+00   3.70207325e-03]
 [  4.61507793e+00   1.22014306e-03]
 [  2.51861392e+00   1.99456440e-02]
 [  4.10527488e+00   2.27140394e-03]
 [ -1.11285308e+00   8.48738930e-01]
 [  7.51581353e+00   6.77453933e-05]
 [  3.98583786e+00   2.64220087e-03]
 [  4.47698592e+00   1.43833880e-03]
 [  2.37864045e+00   2.44899589e-02]
 [             nan   1.00000000e+00]
 [             nan   1.00000000e+00]
 [  2.61393398e-01   4.00656576e-01]
 [  3.82268750e+00   3.25972025e-03]
 [  2.96896124e+00   1.04192109e-02]]
NIAS: t-test pooling all individuals (t-stat,p-value)
```

```
(11.09683395728077, 1.054914752141725e-22)
Elements in the vector are all zero
Elements in the vector are all zero
Shannon: t-test for each individual allowing for heterog. cost functions (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
[[ 1.02832933  0.36191008]
 [ 0.28516293  0.78377208]
 [ 0.5223888   0.61751917]
 [ 0.25241293  0.80797319]
 [ 0.91670062  0.38978884]
 [ 0.39364535  0.70555804]
 [ 0.1644788   0.87400298]
 [ 0.7511978   0.47702322]
 [ 0.29913999  0.77351935]
 [ 0.19940572  0.84761405]
 [ 0.11344127  0.91286568]
 [ 0.41185653  0.69276369]
 [ 0.39112902  0.70733408]
 [ 0.54569546  0.60222308]
 [ 0.35591255  0.73239075]
 [ 0.44727929  0.66818585]
 [ 0.49453495  0.63606956]
 [ 0.1738381   0.86691315]
 [ 0.50269649  0.63060413]
 [        nan         nan]
 [        nan         nan]
 [ 0.20595851  0.84268525]
 [ 0.48058841  0.64546529]]
For all a; H0: cond <= 0
[[ 1.92731831  0.04764889]
 [ 0.28516293  0.39188604]
 [ 0.5223888   0.30875959]
 [ 0.25241293  0.40398659]
 [ 0.91670062  0.19489442]
 [ 0.39364535  0.35277902]
 [ 0.1644788   0.43700149]
 [ 0.7511978   0.23851161]
 [ 0.29913999  0.38675967]
 [ 0.19940572  0.42380703]
 [ 0.11344127  0.45643284]
 [ 0.41185653  0.34638185]
 [ 0.39112902  0.35366704]
 [ 0.54569546  0.30111154]
 [ 0.35591255  0.36619538]
 [ 0.44727929  0.33409292]
 [ 0.49453495  0.31803478]
 [ 0.1738381   0.43345658]
 [ 0.50269649  0.31530207]
 [        nan         nan]
 [        nan         nan]
 [ 0.20595851  0.42134263]
 [ 0.48058841  0.32273265]]
MM; H0: cond == 0
[[ -9.02861551e-16   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]
 [  3.04019329e-16   1.00000000e+00]
 [  2.23964266e-16   1.00000000e+00]
 [ -3.15981790e-16   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]
 [ -1.76102523e-16   1.00000000e+00]
 [ -1.10262080e-16   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]
 [  5.26303143e-16   1.00000000e+00]
 [  1.43711833e-16   1.00000000e+00]
 [ -7.88492250e-17   1.00000000e+00]
 [  9.55292234e-16   1.00000000e+00]
 [ -8.31382890e-17   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]
 [  5.13169796e-17   1.00000000e+00]
 [  1.90982899e-16   1.00000000e+00]
 [  1.13236422e-16   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]
 [             nan   1.00000000e+00]
 [             nan   1.00000000e+00]
 [ -1.46207035e-16   1.00000000e+00]
 [  1.07427783e-16   1.00000000e+00]]
```

```
Shannon: t-test pooling all individuals allowing for heterog. cost functions (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
(1.6097901198586286, 0.10927746080267502)
For all a; H0: cond <= 0
(1.6943645842810426, 0.045987438661756211)
MM; H0: cond == 0
(3.0852375045474666e-16, 0.99999999999999978)
Elements in the vector are all zero
Elements in the vector are all zero
Elements in the vector are all zero
Elements in the vector are all zero
Shannon: t-test for each individual with same cost function (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
[[ 1.02832933  0.36191008]
 [ 0.28213062  0.78600258]
 [ 0.48645905  0.64150166]
 [ 0.14111055  0.89175763]
 [ 0.91740908  0.38944234]
 [ 0.37595982  0.71808168]
 [ 0.16448679  0.87399692]
 [ 0.69012585  0.51233916]
 [ 0.29839041  0.77406799]
 [ 0.18011967  0.86216206]
 [ 0.11343275  0.91287219]
 [ 0.41206726  0.69261626]
 [ 0.39114708  0.70732133]
 [ 0.54393741  0.60336956]
 [ 0.35277284  0.73464233]
 [ 0.44709489  0.6683127 ]
 [ 0.49380696  0.63655825]
 [ 0.17254231  0.86789396]
 [ 0.49352713  0.63674614]
 [        nan  1.        ]
 [        nan  1.        ]
 [ 0.20585759  0.8427611 ]
 [ 0.48094749  0.6452225 ]]
For all a; H0: cond <= 0
[[ 1.92715689  0.04766021]
 [ 0.28213062  0.39300129]
 [ 0.48645905  0.32075083]
 [ 0.14111055  0.44587882]
 [ 0.91740908  0.19472117]
 [ 0.37595982  0.35904084]
 [ 0.16448679  0.43699846]
 [ 0.69012585  0.25616958]
 [ 0.29839041  0.38703399]
 [ 0.18011967  0.43108103]
 [ 0.11343275  0.4564361 ]
 [ 0.41206726  0.34630813]
 [ 0.39114708  0.35366066]
 [ 0.54393741  0.30168478]
 [ 0.35277284  0.36732116]
 [ 0.44709489  0.33415635]
 [ 0.49380696  0.31827912]
 [ 0.17254231  0.43394698]
 [ 0.49352713  0.31837307]
 [ 2.01967807  0.04158491]
 [ 2.01967807  0.04158491]
 [ 0.20585759  0.42138055]
 [ 0.48094749  0.32261125]]
MM; H0: cond == 0
[[ -1.52080807e-15   1.00000000e+00]
 [  4.18833280e-16   1.00000000e+00]
 [  4.13726096e-16   1.00000000e+00]
 [  3.51233660e-16   1.00000000e+00]
 [  4.41600734e-16   1.00000000e+00]
 [ -2.04348899e-16   1.00000000e+00]
 [  1.77499036e-16   1.00000000e+00]
 [  3.25003316e-16   1.00000000e+00]
 [  3.78080785e-16   1.00000000e+00]
 [  3.00452674e-16   1.00000000e+00]
 [  4.51749222e-16   1.00000000e+00]
 [  7.20827094e-16   1.00000000e+00]
 [  5.85682105e-16   1.00000000e+00]
 [  6.66451978e-16   1.00000000e+00]
 [  3.50339756e-16   1.00000000e+00]
```

```
 [  4.94943187e-16   1.00000000e+00]
 [  8.87825051e-16   1.00000000e+00]
 [  1.02792283e-16   1.00000000e+00]
 [  1.64934708e-16   1.00000000e+00]
 [             nan   1.00000000e+00]
 [             nan   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]
 [  0.00000000e+00   1.00000000e+00]]
Shannon: t-test pooling all individuals allwith same cost function (t-stat,p-value)
For PA(a) > 0; H0: cond == 0
(2.1345073230372655, 0.034154367435795308)
For all a; H0: cond <= 0
(3.3469041281183984, 0.00049204421015822401)
MM; H0: cond == 0
(1.469453749639288e-15, 0.99999999999999878)
```

```
D:\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:879: RuntimeWarning: invalid value encoun
tered in greater
  return (self.a < x) & (x < self.b)
D:\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:879: RuntimeWarning: invalid value encoun
tered in less
  return (self.a < x) & (x < self.b)
D:\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:1818: RuntimeWarning: invalid value encou
ntered in less_equal
  cond2 = cond0 & (x <= self.a)
```

In [ ]:

```
# Procedure:
#     get list of lists with all lambdas per individual;
#     get avg lambdas per individual
#     compute z(a,s)
#     compute mu(s)*z(a,s)
#     compute PA(a)*z(a,s)
#     compute mu(s)*z(a,s)/Sum PA(c)*z(c,s)
#     compute Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s')

#     Test per individual
#     2-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 == 0 for a s.t. PA(a)>0
#     1-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 <= 0 for any a
#     2-sided t-test:  mu(s')*z(a,s')/Sum PA(c)*z(c,s') == PA(a|s)

#     Test pooling the individuals
#     2-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 == 0 for a s.t. PA(a)>0
#     1-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 <= 0 for any a
#     2-sided t-test:  mu(s')*z(a,s')/Sum PA(c)*z(c,s') == PA(a|s)


#     Get avg lambdas pooling all individuals
#     compute z(a,s)
#     compute mu(s)*z(a,s)
#     compute PA(c)*z(a,s)
#     compute mu(s)*z(a,s)/Sum PA(c)*z(c,s)
#     compute Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s')

#     Test per individual
#     2-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 == 0 for a s.t. PA(a)>0
#     1-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 <= 0 for any a
#     2-sided t-test:  PA(s)*z(a,s)/Sum PA(c)*z(c,s') == PA(a|s)

#     Test pooling the individuals
#     2-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 == 0 for a s.t. PA(a)>0
#     1-sided t-test:  Sum s': mu(s')*z(a,s')/Sum PA(c)*z(c,s') -1 <= 0 for any a
#     2-sided t-test:  PA(s)*z(a,s)/Sum PA(c)*z(c,s') == PA(a|s)
```