STAT6011/7611/6111/3317 COMPUTATIONAL STATISTICS (2016 Fall) Assignment 4

Kei Ikegami (u3535947)

November 22, 2016

1

The code is below. I denote the 'Fair' by 0 and 'Loaded' by 1 in this code.

Listing 1: problem 1

```python
import numpy as np

def vitervi(transition):
    tosses = "251326344212463366565535614566523665561326345621443235213263461435421"
    prob_loaded = {"1": 1/12, "2": 1/12, "3": 1/12, "4": 1/12, "5":1/3, "6":1/3}
    prob_fair = {"1":1/6, "2":1/6, "3":1/6, "4":1/6, "5":1/6, "6":1/6}
    score_loaded = np.ones(len(tosses))
    score_fair = np.ones(len(tosses))
    score_loaded[0] = 0.5 * prob_loaded[tosses[0]]
    score_fair[0] = 0.5 * prob_fair[tosses[0]]

    for i in range(1,len(tosses)):
        score_loaded[i] = max(score_loaded[i-1] * transition[1,1], score_fair[i-1] * transition[1,0]) * prob_loaded[tosses[i]]
        score_fair[i] = max(score_loaded[i-1] * transition[0,1], score_fair[i-1] * transition[0,0]) * prob_fair[tosses[i]]

    state_estimation = np.ones(len(tosses))
    for n, j in enumerate(score_loaded - score_fair):
        if j < 0:
            state_estimation[n] = 0

    return(state_estimation)

transition1 = np.array([[0.8, 0.3], [0.2, 0.7]])
transition2 = np.array([[0.9, 0.15], [0.1, 0.85]])
transition3 = np.array([[0.95, 0.05], [0.6, 0.4]])
transition4 = np.array([[0.5, 0.5], [0.5, 0.5]])

vitervi(transition1)
vitervi(transition2)
vitervi(transition3)
vitervi(transition4)
```

And the results are as follows.

(a) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

(b) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

(c) [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

(d) [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

2

Listing 2: problem 2

```
import numpy as np
from scipy.special import eval_hermite
from scipy.special import hermite
from scipy.special import h_roots
from scipy.special import he_roots
from scipy.stats import norm

def GH(m):
    points = h_roots(m)[0]
    weights = h_roots(m)[1]
    return sum([weights[i] * np.exp(points[i]**2) * norm.pdf(points[i]) for i in range(len(points))])

def GH_weights(m):
    points = h_roots(m)[0]
    weights = h_roots(m)[1]
    return [weights[i] * np.exp(points[i]**2) for i in range(len(points))]

a = [5, 10, 20, 30]

# approximation result
for m in a:
    print(GH(m))

# weights
for m in a:
    print(GH_weights(m))

# points
for m in a:
    print(h_roots(m)[0])
```

In the below question, I mean the weights by the values multiplied by weight functions.

2.1

I prove the integral of standard normal probability distribution function is equal to 1.

2.2

I use the above code to compute the approximation of the integral.

When the number of nodes is 5,
the approximation result is 0.99684628222456162,
the nodes are [-2.02018287, -0.95857246, 0. , 0.95857246, 2.02018287],
the weights are [1.1814886255359833, 0.98658099675142807, 0.94530872048294179, 0.98658099675142807, 1.1814886255359833]

When the number of nodes is 10,
the approximation result is 0.99998763906433263
the nodes are [-3.43615912, -2.53273167, -1.75668365, -1.03661083, -0.34290133, 0.34290133, 1.03661083, 1.75668365, 2.53273167, 3.43615912],
and the weights are [1.0254516913657519, 0.82066612640481784, 0.74144193194356545, 0.70329632310490586, 0.6870818539512733, 0.6870818539512733, 0.70329632310490586, 0.74144193194356545, 0.82066612640481784, 1.0254516913657519]

When the number of nodes is 20,
the approximation result is 0.99999999979803234,
the nodes are [-5.38748089, -4.60368245, -3.94476404, -3.34785457, -2.78880606, -2.254974 , -1.73853771, -1.23407622, -0.73747373, -0.24534071, 0.24534071, 0.73747373, 1.23407622, 1.73853771, 2.254974 , 2.78880606, 3.34785457, 3.94476404, 4.60368245, 5.38748089],
and the weights are [0.89859196145317, 0.70433296117692357, 0.62227869619138665, 0.57526244285250083, 0.54485174236452072, 0.52408035094855054, 0.50967902711745705, 0.49992087133628998, 0.4938433852720529,

0.49092150066674595, 0.49092150066674595, 0.4938433852720529, 0.49992087133628998, 0.50967902711745705, 0.52408035094855054, 0.54485174236452072, 0.57526244285250083, 0.62227869619138665, 0.70433296117692357, 0.89859196145317]

When the number of nodes is 30,
the approximation result is 0.99999999999999634,
the nodes are [-6.86334529, -6.13827922, -5.53314715, -4.98891897, -4.48305536, -4.0039086 , -3.54444387, -3.09997053, -2.66713212, -2.24339147, -1.82674114, -1.4155278 , -1.00833827, -0.60392106, -0.20112858, 0.20112858, 0.60392106, 1.00833827, 1.4155278 , 1.82674114, 2.24339147, 2.66713212, 3.09997053, 3.54444387, 4.0039086 , 4.48305536, 4.98891897, 5.53314715, 6.13827922, 6.86334529],
and the weights are [0.83424747101269592, 0.64909798155433118, 0.56940269194957616, 0.52252568933130883, 0.49105799583287552, 0.4683748125647248, 0.451321035991189, 0.43817702265268194, 0.42791806293273793, 0.41989500373682354, 0.41367936361113872, 0.40898157500353133, 0.40560512332568432, 0.40341981692480389, 0.40234606670190304, 0.40234606670190304, 0.40341981692480389, 0.40560512332568432, 0.40898157500353133, 0.41367936361113872, 0.41989500373682354, 0.42791806293273793, 0.43817702265268194, 0.451321035991189, 0.4683748125647248, 0.49105799583287552, 0.52252568933130883, 0.56940269194957616, 0.64909798155433118, 0.83424747101269592].

3

Listing 3: problem 3

```
import numpy as np
from scipy.special import p_roots
from scipy.special import u_roots
from scipy.special import t_roots

def f(x):
    return (x**9) / np.sqrt(x**2 + 1)

def GL(m, a, b):
    points = p_roots(m)[0]
    weights = p_roots(m)[1]
    return (b-a)*sum([weights[i]*f((b-a)*points[i]/2 + (b+a)/2) for i in range(len(points))])/2

def Cheby1(m, a, b):
    points = t_roots(m)[0]
    weights = t_roots(m)[1]
    return (b-a)*sum([weights[i]*np.sqrt(1-(points[i])**2) * f((b-a)*points[i]/2 + (b+a)/2) for i in range(len(points))])/2

def Cheby2(m, a, b):
    points = u_roots(m)[0]
    weights = u_roots(m)[1]
    return (b-a)*sum([(weights[i]/np.sqrt(1-(points[i])**2)) * f((b-a)*points[i]/2 + (b+a)/2) for i in range(len(points))])/2

a = [5, 10]

for m in a:
    # approximation results
    print(GL(m))
    print(Cheby1(m))
    print(Cheby2(m))

    # nodes
    print(p_roots(m)[0])
    print(t_roots(m)[0])
    print(u_roots(m)[0])

    # weights
    print(p_roots(m)[1])
    print(t_roots(m)[1])
    print(u_roots(m)[1])
```

In the below questions, I mean the weights by the values not multiplied by the weight functions.

3.1

When I use Legendre polynomial as orthogonal polynomial,
the approximation result is -49.506283813990549,
the nodes are [-0.90617985, -0.53846931, 0. , 0.53846931, 0.90617985],
the weights are [0.23692689, 0.47862867, 0.56888889, 0.47862867, 0.23692689].

When I use Chebyshev type 1 polynomial as orthogonal polynomial,
the approximation result is -57.161045874594777,
the nodes are [-9.51056516e-01, -5.87785252e-01, 6.12323400e-17, 5.87785252e-01, 9.51056516e-01,]
the weights are [0.62831853, 0.62831853, 0.62831853, 0.62831853, 0.62831853].

When I use Chebyshev type 2 polynomial as orthogonal polynomial,
the approximation result is -40.789633611622008,
the nodes are [-8.66025404e-01, -5.00000000e-01, 6.12323400e-17, 5.00000000e-01, 8.66025404e-01]
the weights are [0.13089969, 0.39269908, 0.52359878, 0.39269908, 0.13089969].

3.2

When I use Legendre polynomial as orthogonal polynomial,
the approximation result is -49.493963006199031,
the nodes are [-0.97390653, -0.86506337, -0.67940957, -0.43339539, -0.14887434, 0.14887434, 0.43339539, 0.67940957, 0.86506337, 0.97390653],
the weights are [0.06667134, 0.14945135, 0.21908636, 0.26926672, 0.29552422, 0.29552422, 0.26926672, 0.21908636, 0.14945135, 0.06667134].

When I use Chebyshev type 1 polynomial as orthogonal polynomial,
the approximation result is -50.987455239343021,
the nodes are [-0.98768834, -0.89100652, -0.70710678, -0.4539905 , -0.15643447, 0.15643447, 0.4539905 , 0.70710678, 0.89100652, 0.98768834]
the weights are [0.31415927, 0.31415927, 0.31415927, 0.31415927, 0.31415927, 0.31415927, 0.31415927, 0.31415927, 0.31415927, 0.31415927].

When I use Chebyshev type 2 polynomial as orthogonal polynomial,
the approximation result is -47.101342989695162,
the nodes are [-0.95949297, -0.84125353, -0.65486073, -0.41541501, -0.14231484, 0.14231484, 0.41541501, 0.65486073, 0.84125353, 0.95949297]
the weights are [0.02266894, 0.08347854, 0.16312218, 0.23631356, 0.27981494, 0.27981494, 0.23631356, 0.16312218, 0.08347854, 0.02266894].

4

Listing 4: problem 4

```
import numpy as np
from scipy.stats import binom

n = 25

iteration = 100
values = np.ones((iteration, 3))
initial_value = [1/3, 1/3, 1/3]
values[0, :] = initial_value

for ite in range(1, iteration):
    q_a = (2*values[ite-1, 2]) / (2*values[ite-1, 2] + values[ite-1, 0])
    q_b = (2*values[ite-1, 2]) / (2*values[ite-1, 2] + values[ite-1, 1])
    under = sum([binom.pmf(i, n, q_a) * binom.pmf(j, n, q_b) * (3*n - i) for i in range(n+1) for j in range(n+1)])
    upper = sum([binom.pmf(i, n, q_a) * binom.pmf(j, n, q_b) * (2*n + i + j) for i in range(n+1) for j in range(n+1)])
    values[ite, 0] = 1/(2+(upper/under))
    values[ite, 1] = 1/(2+(upper/under))
```

```
values[ite, 2] = 1 - 2*values[ite, 0]
```

4.1

Let $\theta = (p_a, p_b, p_o)$ and $\mathbf{Y_{obs}} = (n_A, n_B, n_O, n_{AB})$. The missing values are $\mathbf{Z} = (z_{AO}, z_{BO})$. I calculate Q-function as follows.

$$
\begin{aligned}
Q(\theta|\theta^{(t)}) &= E_{\theta^{(t)}}[\log L(\theta|\mathbf{Y_{obs}}, \mathbf{Z})|\theta^{(t)}, \mathbf{Y_{obs}}] = \int \log L(\theta|\mathbf{Y_{obs}}, \mathbf{Z}) f(\mathbf{Z}|\theta^{(t)}, \mathbf{Y_{obs}}) \mathrm{d}\mathbf{Z} \\
&= \sum_{j=0}^{n_B} \sum_{i=0}^{n_A} \log L(\theta|\mathbf{Y_{obs}}, z_{AO} = i, z_{BO} = j) P(z_{AO} = i, z_{BO} = j|\theta^{(t)}, \mathbf{Y_{obs}})
\end{aligned} \tag{1}
$$

Now I can get the joint probability mass function of (z_{AO}, z_{BO}) as follows.

$$
\begin{aligned}
P_{ij} &= P(z_{AO} = i, z_{BO} = j|\theta^{(t)}, \mathbf{Y_{obs}}) \\
&= \binom{n_A}{i} (q_A)^i (1 - q_A)^{n_A - i} * \binom{n_B}{j} (q_B)^j (1 - q_B)^{n_B - j}
\end{aligned}
$$

where

$$
q_A = \frac{2p_A^{(t)} p_O^{(t)}}{2p_A^{(t)} p_O^{(t)} + (p_A^{(t)})^2} = \frac{2p_O^{(t)}}{2p_O^{(t)} + p_A^{(t)}}
$$

$$
q_B = \frac{2p_B^{(t)} p_O^{(t)}}{2p_B^{(t)} p_O^{(t)} + (p_B^{(t)})^2} = \frac{2p_O^{(t)}}{2p_O^{(t)} + p_B^{(t)}}
$$

And the log likelihood function is transformed into the below.

$$
\begin{aligned}
\log L(\theta|\mathbf{Y_{obs}}, z_{AO} = i, z_{BO} = j) &= \log (2p_A p_O)^i (p_A^2)^{n_A - i} (2p_B p_O)^j (p_B^2)^{n_B - j} (p_O^2)^{n_O} (2p_A p_B)^{n_{AB}} \\
&= (2n_A + n_{AB} - i) \log\, p_A + (2n_B + n_{AB} - j) \log\, p_B \\
&\quad + (2n_O + i + j) \log\, p_O + \text{unrelated terms}
\end{aligned}
$$

Then the maximization step is written as follows.

$$
\begin{aligned}
&\max_{\theta} \sum_{j=0}^{n_B} \sum_{i=0}^{n_A} P_{ij} (2n_A + n_{AB} - i) \log\, p_A + (2n_B + n_{AB} - j) \log\, p_B + (2n_O + i + j) \log\, p_O \\
&\text{s.t. } p_A + p_B + p_O = 1
\end{aligned}
$$

By using the method of Lagrange multiplier, we can get the below conditions.

$$
\frac{\partial Q}{\partial p_A} = \frac{\sum_j \sum_i P_{ij} (2n_A + n_{AB} - i)}{p_A} - \frac{\sum_j \sum_i P_{ij} (2n_O + i + j)}{1 - p_A - p_B} = 0
$$

$$
\frac{\partial Q}{\partial p_B} = \frac{\sum_j \sum_i P_{ij} (2n_B + n_{AB} - i)}{p_B} - \frac{\sum_j \sum_i P_{ij} (2n_O + i + j)}{1 - p_A - p_B} = 0
$$

Since $n_A = n_B$, the above two conditions mean that $p_A^{(t+1)} = p_B^{(t+1)}$. Thus I get the maximizer of Q function under the constraint.

$$
p_B^{(t+1)} = p_A^{(t+1)} = \frac{1}{2 + \frac{\sum_j \sum_i P_{ij} (2n_O + i + j)}{\sum_j \sum_i P_{ij} (2n_A + n_{AB} - i)}}
$$

$$
p_O^{(t+1)} = 1 - 2p_A^{(t+1)}
$$

Note that P_{ij} depends on the previous step estimation results. Repeat the above improvement until converge.

4.2

By using the above code, I get the estimation result after 100 iterations.

The result is $[p_a, p_b, p_o] = [0.28021178, 0.28021178, 0.43957643]$.

5

5.1

First I specify the distribution of the missing values, which is u_i, conditional on the observed values.

$$
\begin{aligned}
f(u_i|y_i,\theta) &\propto f(u_i,y_i) = f(y_i|u_i)f(u_i) = f(y_{i,1},y_{i,2},\ldots,y_{i,J}|u_i)f(u_i) = \{\Pi_j f(y_{i,j}|u_i)\} f(u_i) \\
&\propto \left\{\Pi_j \exp(-\frac{(y_{ij}-\beta_0-\beta_1 x_{ij}-u_i)^2}{2\sigma_\epsilon^2})\right\}\exp(-\frac{u_i^2}{2\sigma_u^2}) \\
&\propto \exp\left(-\frac{1}{2\sigma_\epsilon^2}\left(\sum_j u_i^2 - 2u_i\sum_j (y_{ij}-\beta_0-\beta_1 x_{ij})\right) - \frac{u_i^2}{2\sigma_u^2}\right) \\
&\propto \exp\left(-\frac{J\sigma_u^2+\sigma_\epsilon^2}{2\sigma_u^2\sigma_\epsilon^2}\left(u_i - \frac{\sigma_u^2}{J\sigma_u^2+\sigma_\epsilon^2}\sum_j (y_{ij}-\beta_0-\beta_1 x_{ij})\right)^2\right)
\end{aligned}
$$

This means that $f(u_i|y_i,\theta) = N\left(\frac{\sigma_u^2}{J\sigma_u^2+\sigma_\epsilon^2}\sum_j (y_{ij}-\beta_0-\beta_1 x_{ij}), \frac{\sigma_u^2\sigma_\epsilon^2}{J\sigma_u^2+\sigma_\epsilon^2}\right)$. Let this mean be E_t and the variance be V_t when the parameters are t-step estimation results.

Then I get Q function as follows.

$$
\begin{aligned}
Q(\theta|\theta^{(t)}) &= \int\cdots\int \log L(\theta|y,u)\Pi_i f(u_i|y_i,\theta^{(t)})(d)u_1\ldots(d)u_I \\
&= \int\cdots\int \left\{\sum_i\sum_j \log f(y_{ij}|u_i) + \log f(u_i)\right\}\Pi_i f(u_i|y_i,\theta^{(t)})(d)u_1\ldots(d)u_I \\
&= \sum_i\left[\int\left\{\sum_j \log f(y_{ij}|u_i) + \log f(u_i)\right\} f(u_i|y_i)(d)u_i\right] \\
&= \sum_i\sum_j\int \log f(y_{ij}|u_i)f(u_i|\theta^{(t)},y_i)(d)u_i + \sum_i\sum_j\int \log f(u_i)f(u_i|\theta^{(t)},y_i)(d)u_i
\end{aligned} \tag{2}
$$

Then I remove the unrelated terms from (2) and separate it into the left part and the right part. Let the left part of (2) be a, and the right one be b. Now I get the below transformations.

$$
\begin{aligned}
a &= -\sum_i\sum_j\int \log\,\sigma_\epsilon^2 f(u_i|\theta^{(t)},y_i)(d)u_i - \sum_i\sum_j\int \frac{(y_{ij}-\beta_0-\beta_1 x_{ij}-u_i)^2}{2\sigma_\epsilon^2} f(u_i|\theta^{(t+1)},y_i)(d)u_i \\
&= -IJ\log\,\sigma_\epsilon - \frac{\sum_i\sum_j (y_{ij}-\beta-\beta_1 x_{ij})^2}{2\sigma_\epsilon^2} + \frac{\sum_i\sum_j (y_{ij}-\beta-\beta_1 x_{ij})E_t}{\sigma_\epsilon^2} - \frac{\sum_i\sum_j V_t + E_t^2}{2\sigma_\epsilon^2}
\end{aligned} \tag{3}
$$

$$
\begin{aligned}
b &= -\sum_i\sum_j\int \log\,\sigma_u^2 f(u_i|\theta^{(t)},y_i)(d)u_i - \sum_i\sum_j\int \frac{u_i^2}{2\sigma_u^2} f(u_i|\theta^{(t)},y_i)(d)u_i \\
&= -IJ\log\,\sigma_u^2 - \frac{\sum_i\sum_j V_t + E_t^2}{2\sigma_u^2}
\end{aligned} \tag{4}
$$

By the above, Q function is $a+b$. Then the maximization step is done by differentiating the Q function by each parameter. By using (3) and (4), I get the belows.

$$\frac{\partial Q}{\partial \beta_0} = -\frac{IJ}{\sigma_\epsilon^2}\beta_0 + \frac{\sum_i \sum_j (y_{ij} - \beta_1 x_{ij})}{\sigma_\epsilon^2} - \frac{J\sum_i E_t}{\sigma_\epsilon^2} = 0 \tag{5}$$

$$\frac{\partial Q}{\partial \beta_1} = -\frac{\sum_i \sum_j x_{ij}^2}{\sigma_\epsilon^2}\beta_1 + \frac{\sum_i \sum_j x_{ij}(y_{ij} - \beta_0)}{\sigma_\epsilon^2} - \frac{\sum_i \sum_j x_{ij} E_t}{\sigma_\epsilon^2} = 0 \tag{6}$$

$$\frac{\partial Q}{\partial \sigma_\epsilon^2} = -\frac{IJ}{2\sigma_\epsilon^2} + \frac{\sum_i \sum_j (y_{ij} - \beta_0 - \beta_1 x_{ij})^2}{2(\sigma_\epsilon^2)^2} - \frac{\sum_i \sum_j (y_{ij} - \beta_0 - \beta_1 x_{ij})E_t}{(\sigma_\epsilon^2)^2} + \frac{\sum_i \sum_j V_t + E_t^2}{2(\sigma_\epsilon^2)^2} = 0 \tag{7}$$

$$\frac{\partial Q}{\partial \sigma_u^2} = -\frac{IJ}{2\sigma_u^2} + \frac{\sum_i \sum_j V_t + E_t^2}{2(\sigma_u^2)^2} = 0 \tag{8}$$

Note that E_t and V_t depend on the previous estimation results.

By (5) and (6) I can get the next estimations of β_0 and β_1 as follows. Let

$$\sum_i \sum_j x_{ij} y_{ij} = XY$$
$$\sum_i \sum_j x_{ij} = X$$
$$\sum_i \sum_j y_{ij} = Y$$
$$\sum_i \sum_j x_{ij}^2 = X_2$$

According to the above notations,

$$\beta_1^{(t+1)} = \frac{IJ * XY - X * Y - J\left\{I\sum_i \sum_j x_{ij}E_t - (\sum_i E_t) * X\right\}}{IJ * X_2 - X * X}$$

$$\beta_0^{(t+1)} = \frac{Y - \beta_1^{(t)} * X - J\sum_i E_t}{IJ}$$

By (7) and (8),

$$(\sigma_\epsilon^2)^{(t+1)} = V_t + \frac{\sum_i \sum_j (y_{ij} - \beta_0^{(t)} - \beta_1^{(t)} x_{ij})^2 + J\sum_i E_t^2 - 2\sum_i \sum_j (y_{ij} - \beta_0^{(t)} - \beta_1^{(t)} x_{ij})E_t}{IJ}$$

$$(\sigma_u^2)^{(t+1)} = V_t + \frac{J\sum_i E_t^2}{IJ}$$

Repeat the above improvements until convergence.

5.2

Listing 5: problem 5

```
import numpy as np

I = 100
J = 2
beta_0 = -1
beta_1 = 1
sigma_u = 0.5
sigma_eps = 1
iteration = 30
initial = [0, 0, 5, 5]
result = []
```

```

def xE(E_t):
    return sum(sum(value_x * np.array([E_t, E_t]).T))

def eps_right(E_t, beta0, beta1):
    return (sum(sum((value_y - beta0-beta1*value_x)**2)) + J * sum(E_t**2) - 2*sum(sum((value_y - beta0 -
        beta1*value_x) * np.array([E_t, E_t]).T)))/(I*J)

for i in range(500):

    # data generation
    np.random.seed(i)
    value_x = np.random.normal(0, 1, (I, J))
    u = np.random.normal(0, sigma_u, I)
    value_u = np.ones((I, J))
    value_u[:, 0] = u
    value_u[:, 1] = u
    value_eps = np.random.normal(0, sigma_eps, (I, J))
    value_y = beta_0 + beta_1 * value_x + value_u + value_eps

    XY = sum(sum(value_y * value_x))
    X = sum(sum(value_x))
    Y = sum(sum(value_y))
    X_2 = sum(sum(value_x * value_x))

    # estimation
    E_t = np.ones(I)
    estimation = np.ones((iteration, 4))
    estimation[0, :] = initial

    for ite in range(1,iteration):
        u = estimation[ite-1, 3]
        eps = estimation[ite-1, 2]
        beta0 = estimation[ite-1, 0]
        beta1 = estimation[ite-1, 1]
        for i in range(I):
            E_t[i] = u*sum([value_y[i, j] - beta0 - beta1*value_x[i, j] for j in range(J)]) / (J*u + eps)
        V_t = eps*u/(J*u + eps)
        estimation[ite, 1] = (I*J *XY - Y*X - J * (I*xE(E_t) - sum(E_t)*X))/(I*J*X_2 - X**2)
        estimation[ite, 0] = (Y-beta1*X - J* sum(E_t))/(I*J)
        estimation[ite, 2] = V_t + eps_right(E_t, estimation[ite-1, 0], estimation[ite-1, 1])
        estimation[ite, 3] = V_t + sum(E_t**2)/I

    a = estimation[iteration-1, :]
    result.append(a)

result = np.array(result)

# change variances into standard deviations
result[:, 2] = np.sqrt(result[:, 2])
result[:, 3] = np.sqrt(result[:, 3])

bias = np.mean(result, axis = 0) - [-1, 1, 1, 0.5]
std = np.std(result, axis = 0)
print(bias)
print(std)
```

The biases of parameters are [-0.00220228, -0.00313101, -0.01689459, 0.00997698].
The standard deviations are [0.08100258, 0.07864963, 0.06433125, 0.09524904].