

# How to Use the Script

## For Items of Three Categories

Model, How to Use the Script, and Listings

Yasuharu Okamoto, 2018

### Model

The coefficient of reliability calculated in this study is based on a psychological model, which is called Thurstonian model, or is known as Samejima's model in item response theory (IRT). According to Thurstonian modeling, categorical response  $Y_{ij}$  on item  $j$  of person  $i$  is generated by categorizing a latent continuous response  $U_{ij}$  on item  $j$  of person  $i$  (Okamoto, 2017). The latent continuous response  $U_{ij}$  consists of a true value  $\mu_j + \lambda_j F_i$  and an error  $E_{ij}$ , that is,

$$(1) \quad U_{ij} = \mu_j + \lambda_j F_i + E_{ij}$$

Strength of the psychological concept to measure of person  $i$  is represented by  $F_i$ . For model (1), the true value is defined to correspond to the psychological concept to measure.

The observed categorical score  $Y_{ij}$  on categorical item  $j$  of person  $i$  is given by categorization of  $U_{ij}$ , that is,

$$(2) \quad Y_{ij} = k, \quad \text{if } C_{k-1} \leq U_{ij} < C_k,$$

where  $C_k$  is category boundary.

A score of a psychological scale of  $M$  categorical items of person  $i$  is given by the sum  $Y_i$ , where

$$(3) \quad Y_i = \sum_{j=1}^M Y_{ij}.$$

For the model given by Equations (1) to (3), relation of a true value  $\mu_j + \lambda_j F_i$ , more specifically  $F_i$ , and the observed scale score  $Y_i$  can be represented by the following regression models:

$$(4) \quad Y_i = a_F + b_F F_i + \varepsilon_F$$

or

$$(5) \quad F_i = a_U + b_U Y_i + \varepsilon_U$$

Reliability of the scale, which represents strength of the relation between the observed score  $Y_i$  and the true value  $F_i$ , can be given by the coefficient of determination  $R^2$ -

index, which is given by squared correlation of  $Y_i$  and  $F_i$ , the common value for regression models (4) and (5). This coefficient of reliability given by  $R^2$ -index is denoted by  $R^2$  (Okamoto (2016)) or  $\rho_{cat}$  (Okamoto (2017)), and given by

$$(6) \quad R^2 = \rho_{cat} = [Cor(Y_i, F_i)]^2$$

Reliability coefficient  $R^2$  or  $\rho_{cat}$  represents strength of the relation between an observed score  $Y_i$  and the true value  $F_i$  of the psychological concept to measure. The popular coefficient of reliability  $\alpha$ , which can be considered to be a coefficient of statistical reliability, tends to overestimate  $R^2$  or  $\rho_{cat}$  (Okamoto, 2016, 2017). On distinction of statistical reliability and psychological reliability, see the discussion of the website from which this document was downloaded.

Python scripts to estimate  $\rho_{cat}$  (that is,  $R^2$ ) are available below. Formulae of the calculation are given in appendix of Okamoto (2017) and rather complex. However, it should be noted that in general, the following simpler equation is wrong:

$$(7) \quad Cov(X, Y) = \int_{-\infty}^{+\infty} Cov(X, Y|\theta)p(\theta)d\theta, \quad (Wrong \ Equation)$$

where  $Cov(X, Y|\theta)$  denotes covariance of  $X$  and  $Y$  conditional on  $\theta$ , and  $p(\theta)$  is a probability density function of  $\theta$ .

The following example shows that Equation (7) is wrong.

Set two variables  $X$  and  $Y$  as follows:

$$\begin{aligned} X &= F, & F &\sim N(0, 1) \text{ a standard normal distribution} \\ Y &= a + bX \end{aligned}$$

Then, denoting expectations of  $X$  and  $Y$  by  $\bar{X}$  and  $\bar{Y}$ , we have

$$Cov(X, Y) = E((X - \bar{X})(Y - \bar{Y})) = E(XY) - \bar{X}\bar{Y}$$

Since

$$\bar{X} = E(F) = 0,$$

we have

$$\bar{X}\bar{Y} = 0.$$

We also have

$$E(XY) = E(aX + bX^2) = a\bar{X} + bE(F^2) = bVar(F) = b$$

Hence, we have

$$(8) \quad Cov(X, Y) = b.$$

On the other hand, denoting expectations of  $X$  and  $Y$  conditional on  $F$  by  $\bar{X}|F$  and  $\bar{Y}|F$ , and variables  $X$  and  $Y$  conditional on  $F$  by  $X|F$  and  $Y|F$ , we have

$$\begin{aligned} (9) \quad Cov(X, Y|F) &= E[(X|F - \bar{X}|F)(Y|F - \bar{Y}|F)]|F \\ &= E[(F - F)((a + bF) - (a + bF))]|F \end{aligned}$$

$$= 0,$$

where  $E[\cdot|F]$  denotes expectation conditional on  $F$ .

Hence, we have

$$(10) \quad \int_{-\infty}^{+\infty} Cov(X, Y|F)\phi(F)dF = 0,$$

where  $\phi(F)$  is a density function of  $N(0, 1)$ .

Equations (8) and (10) shows that Equation (7) does not hold.

Python scripts with Stan script were developed. The models (1) and (2) give probabilities of categorical responses as follows:

$$(11) \quad \begin{aligned} & P(Y_{ij} = k | \mathbf{F}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{\psi}, \mathbf{C}) \\ &= P(C_{k-1} \leq U_{ij} < C_k | \mathbf{F}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{\psi}, \mathbf{C}) \\ &= \Phi\left(\frac{C_k - (\mu_j + \lambda_j F_i)}{\psi_j}\right) - \Phi\left(\frac{C_{k-1} - (\mu_j + \lambda_j F_i)}{\psi_j}\right) \end{aligned}$$

Equation (11) shows that an origin and a unit of the scale are arbitrary. To identify parameter values, some restrictions of parameters are needed (Okamoto, 2017). The Stan script was developed with the following restrictions for model identification.

$$(12) \quad C_1 = -1 \quad \text{and} \quad C_{K-1} = C_2 = 1$$

The origin and unit are set indirectly by restriction (12). The Stan script with condition (12) is shown below (Listing 1).

## How to use the program

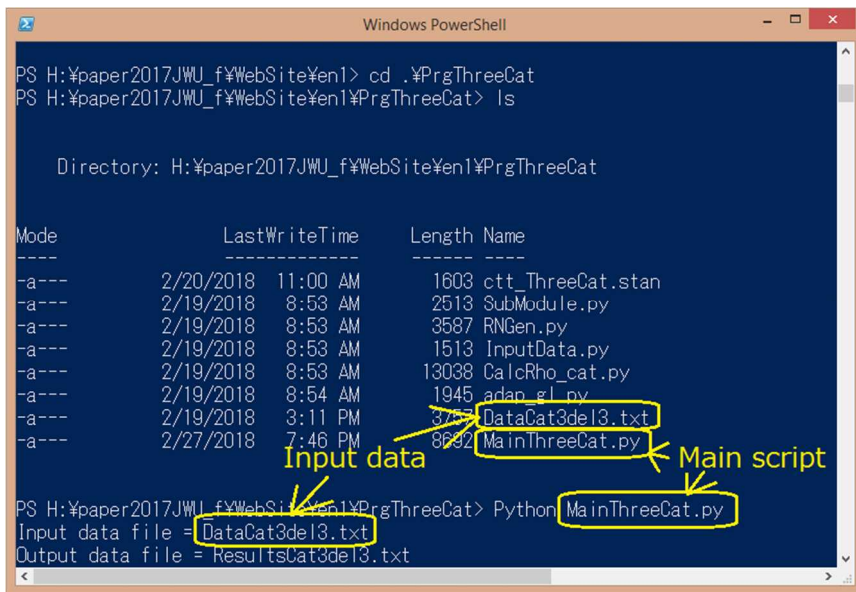


Figure 1

The main script is Main.py (Figure 1). Run this script, then a file name of the input data file is asked. After typing in the input data file name, an output data file name will be asked. Any text file name can be used.

Format of an input data file is as that as shown in Figure 2.

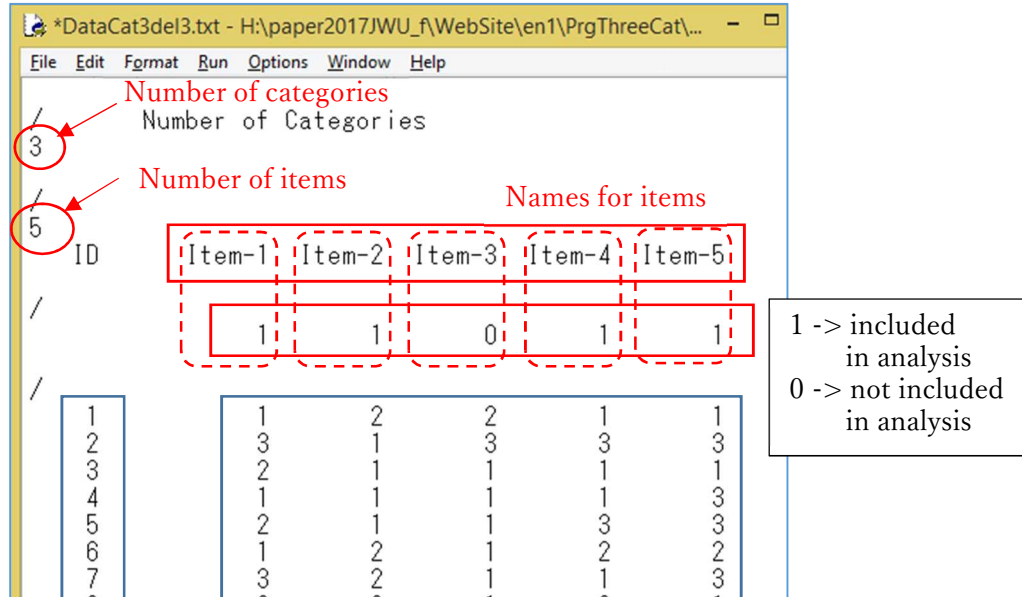


Figure 2

In the next line after the first one with a slash / at the head, the number of categories is written.

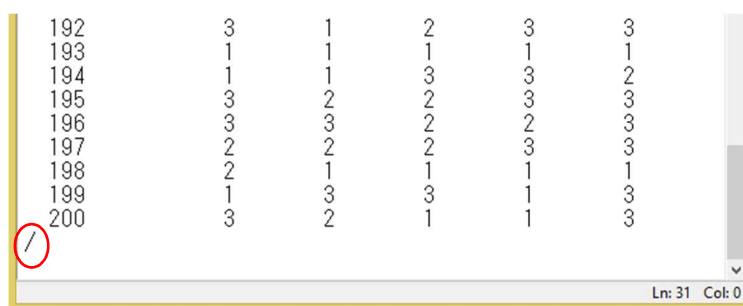
In the next line after the second one with a slash / at the head, the number of items

is written, and in the succeeding line, a name of a person identification variable and names of items are written.

In the next line after the third one with a slash / at the head, integers, 0 or 1, are arranged. Integer 1 indicates that the corresponding item is included in the analysis, and integer 0 indicates that the corresponding item is not included in the analysis.

After the fourth line with a slash / at the head, scores on items are written following the format in which an identification string and one set of scores of a person is written in one line, each line starts with the person identification string. Scores on items of  $K$  categories should be integers from 1 to  $K$  (Okamoto, 2017). An integer out of this range 1 to  $K$  is treated as a missing value.

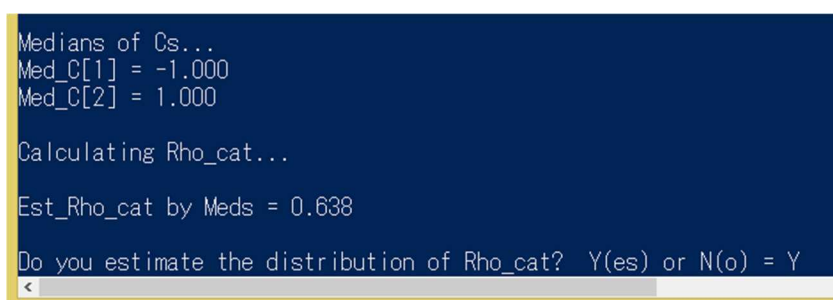
The next line after the last data must have a slash at the head, by which the end of the data is shown (Figure 3).



192	3	1	2	3	3
193	1	1	1	1	1
194	1	1	3	3	2
195	3	2	2	3	3
196	3	3	2	2	3
197	2	2	2	3	3
198	2	1	1	1	1
199	1	3	3	1	3
200	3	2	1	1	3
/					

Figure 3

After typing in the output file name, the Stan script will run. When execution of the Stan script ends, the coefficient of reliability  $\rho_{cat}$  will be calculated using the medians of the posterior distributions of the parameters. In Figure 4, the estimated value  $\hat{\rho}_{cat} \approx 0.638$  is shown as Est\_Rho\_cat by Meds.



```

Medians of Cs...
Med_C[1] = -1.000
Med_C[2] = 1.000

Calculating Rho_cat...

Est_Rho_cat by Meds = 0.638

Do you estimate the distribution of Rho_cat? Y(es) or N(o) = Y

```

Figure 4

After showing the estimated value of the coefficient of reliability using the medians of the parameters, whether you want the distribution of the reliability coefficient or not will be asked. If you type in character 'N', the program ends. But, you type in character 'Y', the

program starts estimating the posterior distribution. Calculation of this distribution will be done using 100 samples of set of parameters, which are randomly selected from the samples generated by the Stan script, and executed by multiprocessing with four processes, each of which will display a window with the current step of calculation shown (Figure 5).

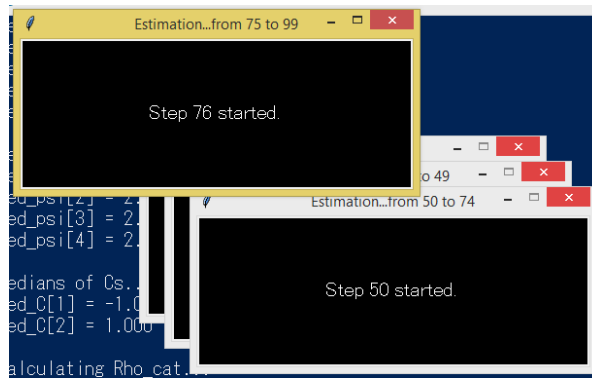


Figure 5

After the calculation ends, a histogram of the distribution of the 100 samples of the coefficient of reliability  $\rho_{cat}$  will be displayed (Figure 6). If you want a histogram of more than 100 samples, you can get the one by modifying the script.

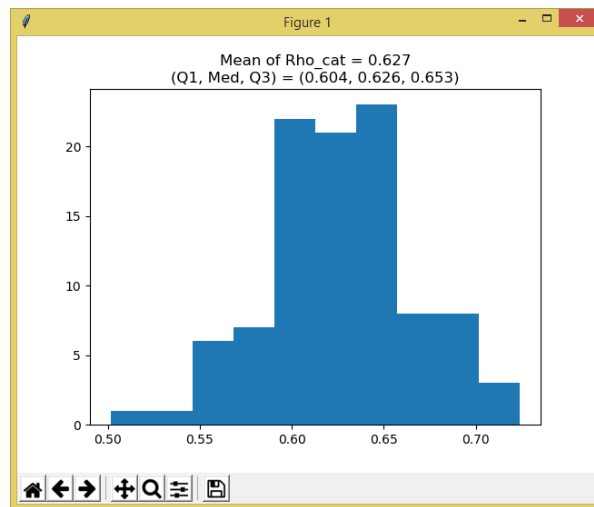


Figure 6

Click the icon X at the upper right corner to close the window, then the program will end. After the program ends, you can open the output file by any editor. An example of the content of an output file is as follows:

Input data file = DataCat3del3.txt

Input data file name

NPerson = 200  
NItems = 4

Number of persons, Number of items,  
Number of categories

NCat = 3

```

1/1: 1 2 1 1
2/2: 3 1 3 3
3/3: 2 1 1 1
4/4: 1 1 1 3
5/5: 2 1 3 3
6/6: 1 2 2 2
7/7: 3 2 1 3
8/8: 2 3 2 1
9/9: 3 1 1 2
10/10: 2 3 3 3
11/11: 1 1 1 1

```



Input Data  
Serial Number/Person ID: Item1, Item2, Item3, Item4

```

195/195: 3 2 3 3
196/196: 3 3 2 3
197/197: 2 2 3 3
198/198: 2 1 1 1
199/199: 1 3 1 3
200/200: 3 2 1 3

```



Reconstructed data for the Stan script...

```

1/800 IDpsn/1 IDitm/1 Res/1
2/800 IDpsn/1 IDitm/2 Res/2
3/800 IDpsn/1 IDitm/3 Res/1
4/800 IDpsn/1 IDitm/4 Res/1
5/800 IDpsn/2 IDitm/1 Res/3
6/800 IDpsn/2 IDitm/2 Res/1
7/800 IDpsn/2 IDitm/3 Res/3
8/800 IDpsn/2 IDitm/4 Res/3
9/800 IDpsn/3 IDitm/1 Res/2
10/800 IDpsn/3 IDitm/2 Res/1
11/800 IDpsn/3 IDitm/3 Res/1
12/800 IDpsn/3 IDitm/4 Res/1

```

Serial\_Number Person\_ID Item\_ID Response



```

795/800 IDpsn/199 IDitm/3 Res/1
796/800 IDpsn/199 IDitm/4 Res/3
797/800 IDpsn/200 IDitm/1 Res/3
798/800 IDpsn/200 IDitm/2 Res/2
799/800 IDpsn/200 IDitm/3 Res/1
800/800 IDpsn/200 IDitm/4 Res/3

```

The coefficient of reliability  $\alpha$

Coeff\_alpha = 0.659

fit...

Inference for Stan model: anon\_model\_4540cd75995217f7d1792fa82c005109.  
4 chains, each with iter=2000; warmup=1000; thin=1;  
post-warmup draws per chain=1000, total post-warmup draws=4000.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
Lambda[0]	2.23	8.8e-3	0.42	1.5	1.94	2.2	2.48	3.13	2256	1.0
Lambda[1]	1.88	7.2e-3	0.4	1.16	1.61	1.86	2.13	2.73	2992	1.0
Lambda[2]	2.31	9.0e-3	0.47	1.49	1.97	2.27	2.61	3.34	2717	1.0
Lambda[3]	1.96	8.0e-3	0.41	1.24	1.67	1.93	2.2	2.86	2618	1.0
mu[0]	0.08	5.4e-3	0.26	-0.42	-0.09	0.08	0.25	0.59	2302	1.0

mu[1]	-0.22	4.1e-3	0.26	-0.75	-0.39	-0.23	-0.05	0.28	4000	1.0
mu[2]	-0.13	5.7e-3	0.29	-0.73	-0.32	-0.12	0.06	0.45	2656	1.0
mu[3]	0.26	4.2e-3	0.26	-0.26	0.08	0.25	0.43	0.79	4000	1.0
psi[0]	2.28	8.1e-3	0.37	1.59	2.02	2.26	2.51	3.05	2084	1.0
psi[1]	2.52	5.7e-3	0.36	1.89	2.27	2.49	2.74	3.31	4000	1.0
psi[2]	2.68	6.6e-3	0.42	1.98	2.38	2.64	2.94	3.57	4000	1.0
psi[3]	2.53	5.9e-3	0.37	1.89	2.27	2.51	2.78	3.34	4000	1.0
C[0]	-1.0	0.0	0.0	-1.0	-1.0	-1.0	-1.0	-1.0	4000	nan
C[1]	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	4000	nan
lp__	-809.2	0.58	16.11	-841.8	-819.9	-809.5	-798.1	-778.3	759	1.0

Samples were drawn using NUTS at Wed Feb 28 15:17:31 2018.

For each parameter, `n_eff` is a crude measure of effective sample size, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat=1`).

Median of Lambdas...

Med\_Lambda[1] = 2.201

Med\_Lambda[2] = 1.859

Med\_Lambda[3] = 2.273

Med\_Lambda[4] = 1.927

Medians of mus...

Med\_mu[1] = 0.078

Med\_mu[2] = -0.225

Med\_mu[3] = -0.124

Med\_mu[4] = 0.252

Medians of psis...

Med\_psi[1] = 2.263

Med\_psi[2] = 2.491

Med\_psi[3] = 2.642

Med\_psi[4] = 2.511

Medians of Cs...

Med\_C[1] = -1.000

Med\_C[2] = 1.000

Est\_Rho\_cat by Meds = 0.638

$\rho_{cat}$ : calculated with medians

Mean of Rho\_cat = 0.627

Median of Rho\_cat = 0.626

[Q1, Q3] = [0.604, 0.653]

Statistics of the sample distribution of

$\rho_{cat}$



## Listings of scripts

The zip file PrgThree3Cat.zip, which is available at the website, contains the files shown in Figure 7. Listings 1 and 2 below show the Stan script `ctt_ThreeCat.stan`, which do MCMC sampling for the model explained above, and the main Python script `MainThreeCat.py`, which calls the Stan script and calculate the coefficient of reliability  $\rho_{cat}$  using the class `CalcRhoPreRhoYYP` included in `SubModule.py`.

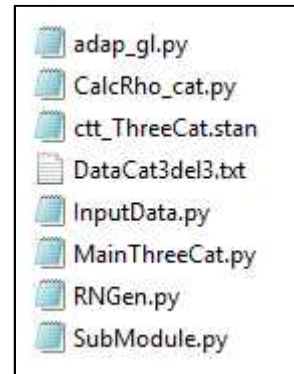


Figure 7

**Listing 1.** The Stan script `ctt_ThreeCat.stan` for estimation of parameter values.

```
// Yasuharu Okamoto, 2017.07

data {
  int <lower = 3, upper = 3> K; // Number of categories = 3
  int Npsn; // Number of persons
  int Nitm; // Number of items
  int Ntot; // Number of data, Ntot <= Nprn * Nitm
  int<lower = 1, upper = Npsn> IDpsn[Ntot]; // Person ID
  int<lower = 1, upper = Nitm> IDitm[Ntot]; // Item ID
  int<lower = 1, upper = K> Res[Ntot]; // Response-> An integer value between 1
  and K
}

parameters {
  real<lower = 0.0> Lambda[Nitm];
  real mu[Nitm];
  real<lower = 0.0> psi[Nitm];
  real F[Npsn];
}

transformed parameters {
  vector[K] p[Ntot];
  real C[K - 1];

  C[1] = -1.0;
  C[2] = 1.0;

  for (i in 1:Ntot) {
    p[i][K] = Phi(((mu[IDitm[i]] + (Lambda[IDitm[i]] * F[IDpsn[i]])) - C[K - 1]) /
    psi[IDitm[i]]);
    p[i][1] = 1.0 - Phi(((mu[IDitm[i]] + (Lambda[IDitm[i]] * F[IDpsn[i]])) - C[1]) /
    psi[IDitm[i]]);

    for (k in 2:(K - 1)) {
      p[i][k] = Phi(((mu[IDitm[i]] + (Lambda[IDitm[i]] * F[IDpsn[i]])) - C[k - 1]) /
      psi[IDitm[i]])
      - Phi(((mu[IDitm[i]] + (Lambda[IDitm[i]] * F[IDpsn[i]])) - C[k]) /
      psi[IDitm[i]]);
    }
  }
}
```

```

    }
  }
}

model {
  for (i in 1:Npsn)
    F[i] ~ normal(0.0, 1.0);
  for (i in 1:Ntot)
    Res[i] ~ categorical(p[i]);
}

```

**Listing 2.** The main Python script MainThreeCat.py.

```

import pystan
from pystan import StanModel
import pickle
from multiprocessing.pool import Pool
import numpy as np
import matplotlib.pyplot as plt
from InputData import *
from SubModule import *
from CalcRho_cat import *
from RNGen import *

if __name__ == '__main__':

    #
    #     Prepare the input data file
    #

    fn_in = input("Input data file = ")
    f_in = open(fn_in, "r")

    #
    #     Prepare the output file
    #

    fn_out = input("Output data file = ")
    f_out = open(fn_out, "w")
    f_out.write("Input data file = " + fn_in + "\n")

    NPerson, NItems, NCat, ID, X = ReadData(f_in, f_out)
    f_out.write('\n\nNPerson = {}'.format(NPerson))
    f_out.write('\nNItems = {}'.format(NItems))
    f_out.write('\nNCat = {}'.format(NCat))

    for i in range(len(ID)):
        f_out.write('\n{0}/{1}: '.format((i + 1), ID[i]))
        for v in X[i]:
            f_out.write(' {}'.format(v))

    if (NCat != 3):
        print('\nNumber of Categories is not three !')
        raise Exception('This programm is for items with three catgories.')

```

```

K = NCat
Ntot = 0
IDpsn = []
IDitm = []
Res = []
for i in range(NPerson):
    for j in range(NItems):
        if (X[i][j] >= 1) and (X[i][j] <= K):
            IDpsn.append(i + 1)
            IDitm.append(j + 1)
            Res.append(X[i][j])
            Ntot += 1

f_out.write('\n\nReconstructed data for the Stan script... ')
for t in range(Ntot):
    f_out.write('\n{0}/{1} IDpsn/{2} IDitm/{3} Res/{4}'.format(
        (t + 1), Ntot, IDpsn[t], IDitm[t], Res[t]))

coeff_alpha = CalcAlpha(X, NCat, NPerson, NItems)
if coeff_alpha < 0.0:
    print('\nA missing value was found in the dataset.')
    print('The coefficient alpha was not calculated.')
    f_out.write('\n\nA missing value was found in the dataset.\n')
    f_out.write('The coefficient alpha was not calculated.\n')
else:
    print('\nCoeff_alpha = {0:.3f}\n'.format(coeff_alpha))
    f_out.write('\n\nCoeff_alpha = {0:.3f}\n'.format(coeff_alpha))

Data = {'K': NCat, 'Npsn': NPerson, 'Nitm': NItems, 'Ntot': Ntot,
        'IDpsn': IDpsn, 'IDitm': IDitm, 'Res': Res}

iLmbd = []
imu = []
ipsi = []
for i in range(NItems):
    iLmbd.append(1.0)
    imu.append(0.0)
    ipsi.append(1.0)

def f_init():
    return dict(Lambda = iLmbd, mu = imu, psi = ipsi)

fit = pystan.stan(file = 'ctt_ThreeCat.stan', data = Data,
                  init = f_init, seed = 999,
                  pars = ['Lambda', 'mu', 'psi', 'C'], n_jobs = 1)

"""
sm = StanModel(file = 'ctt_ThreeCat.stan')
with open('model.pkl', 'wb') as f:
    pickle.dump(sm, f)

fit = sm.sampling(data = Data, seed = 999,
                  pars = ['Lambda', 'mu', 'psi', 'C'], n_jobs = 1)
with open('fit.pkl', 'wb') as g:
    pickle.dump(fit, g)

```

```

"""

"""

sm = pickle.load(open('model.pkl', 'rb'))
fit = pickle.load(open('fit.pkl', 'rb'))
"""

print(fit)

f_out.write('\n\nfit...\n')
f_out.write('{}'.format(fit))

Llmbd = fit['Lambda']
Lmu = fit['mu']
Lpsi = fit['psi']
LC = fit['C']
RawNSamples = len(Llmbd)

rn = RNp2to191()

LSamplesID = []
for i in range(RawNSamples):
    LSamplesID.append([])
    LSamplesID[i].append(i)
    LSamplesID[i].append(rn.uni())

LSamplesID.sort(key = lambda v: v[1])

MCMCSmpl_lmbd = []
for i in range(NItems):
    MCMCSmpl_lmbd.append([])
for v in Llmbd:
    for i in range(NItems):
        MCMCSmpl_lmbd[i].append(v[i])

MCMCSmpl_mu = []
MCMCSmpl_psi = []
MCMCSmpl_C = []
for i in range(NItems):
    MCMCSmpl_mu.append([])
    MCMCSmpl_psi.append([])

for i in range(NCat - 1):
    MCMCSmpl_C.append([])

for v in Lmu:
    for i in range(NItems):
        MCMCSmpl_mu[i].append(v[i])
for v in Lpsi:
    for i in range(NItems):
        MCMCSmpl_psi[i].append(v[i])
for v in LC:
    for i in range(NCat - 1):
        MCMCSmpl_C[i].append(v[i])

for i in range(NItems):
    MCMCSmpl_lmbd[i].sort()

```

```

    MCMCSmpl_mu[i].sort()
    MCMCSmpl_psi[i].sort()

for i in range(NCat - 1):
    MCMCSmpl_C[i].sort()

Med_Lmbd = []
for i in range(NItems):
    Med_Lmbd.append(MCMCSmpl_lmbd[i][int(len(MCMCSmpl_lmbd[i]) / 2.0)])

Med_mu = []
for i in range(NItems):
    Med_mu.append(MCMCSmpl_mu[i][int(len(MCMCSmpl_mu[i]) / 2.0)])
Med_psi = []
for i in range(NItems):
    Med_psi.append(MCMCSmpl_psi[i][int(len(MCMCSmpl_psi[i]) / 2.0)])
Med_C = []
for i in range(NCat - 1):
    Med_C.append(MCMCSmpl_C[i][int(len(MCMCSmpl_C[i]) / 2.0)])

print('\nMedians of Lambdas...')
for i in range(NItems):
    print("Med_Lambda[{}] = {:.3f}".format((i + 1), Med_Lmbd[i]))
print('\nMedians of mus...')
for i in range(NItems):
    print('Med_mu[{}] = {:.3f}'.format((i + 1), Med_mu[i]))
print('\nMedians of psis...')
for i in range(NItems):
    print('Med_psi[{}] = {:.3f}'.format((i + 1), Med_psi[i]))
print('\nMedians of Cs...')
for i in range(NCat - 1):
    print('Med_C[{}] = {:.3f}'.format((i + 1), Med_C[i]))

f_out.write('\n\nMedian of Lambdas... \n')
for i in range(NItems):
    f_out.write("Med_Lambda[{}] = {:.3f}\n".format((i + 1), Med_Lmbd[i]))
f_out.write('\nMedians of mus... \n')
for i in range(NItems):
    f_out.write('Med_mu[{}] = {:.3f}\n'.format((i + 1), Med_mu[i]))
f_out.write('\nMedians of psis... \n')
for i in range(NItems):
    f_out.write('Med_psi[{}] = {:.3f}\n'.format((i + 1), Med_psi[i]))
f_out.write('\nMedians of Cs... \n')
for i in range(NCat - 1):
    f_out.write('Med_C[{}] = {:.3f}\n'.format((i + 1), Med_C[i]))

print('\nCalculating Rho_cat...')
calc_rhos = CalcRhoPreRhoYYP( NCat, NItems, Med_mu, Med_Lmbd, Med_psi, Med_C )
print('\nEst_Rho_cat by Meds = {:.3f}'.format(calc_rhos.GetRhoPre()))
f_out.write('\nEst_Rho_cat by Meds = {:.3f}\n'.format(calc_rhos.GetRhoPre()))

print('\nDo you estimate the distribution of Rho_cat? Y(es) or N(o) = ', end = '')
ck_est = input()
if (ck_est == 'Y') or (ck_est == 'y'):
    print('Estimation started.')
    StartPos = 0

```

```

StopPos = 9
param0 = {'NCat': NCat, 'NItems': NItems, 'StartPos': StartPos, 'StopPos': StopPos,
          'LSamplesID': LSamplesID, 'Llmbd': Llmbd, 'Lmu': Lmu, 'Lpsi': Lpsi, 'LC':
LC}

n_proc = 4;
params = []
NEstRhos = None
if (RawNSamples > 100):
    NEstRhos = 100
else:
    NEstRhos = RawNSamples

for iproc in range(n_proc):
    param_t = param0.copy()
    param_t['StartPos'] = int(iproc * NEstRhos / n_proc)
    param_t['StopPos'] = int(((iproc + 1) * NEstRhos / n_proc) - 1)
    params.append(param_t)

pool = Pool()
Results = pool.map(EstDistriRho_cat, params)

SamplesRho_cat = []
for u in Results:
    for v in u:
        SamplesRho_cat.append(v)

sum = 0.0
for v in SamplesRho_cat:
    sum += v
meanRho_cat = sum / len(SamplesRho_cat)
SamplesRho_cat.sort()

medRho_cat = SamplesRho_cat[int(len(SamplesRho_cat) / 2)]
Q1Rho_cat = SamplesRho_cat[int(len(SamplesRho_cat) / 4)]
Q3Rho_cat = SamplesRho_cat[int(len(SamplesRho_cat) * 3 / 4)]

print('\nMean of Rho_cat = {0:.3f}'.format(meanRho_cat))
print('Median of Rho_cat = {0:.3f}'.format(medRho_cat))
print(' [Q1, Q3] = [{0:.3f}, {1:.3f}]'.format(Q1Rho_cat, Q3Rho_cat))

f_out.write('\n\nMean of Rho_cat = {0:.3f}\n'.format(meanRho_cat))
f_out.write('Median of Rho_cat = {0:.3f}\n'.format(medRho_cat))
f_out.write(' [Q1, Q3] = [{0:.3f}, {1:.3f}]\n'.format(Q1Rho_cat, Q3Rho_cat))

title_str = 'Mean of Rho_cat = {0:.3f}\n' + \
            '(Q1, Med, Q3) = ({1:.3f}, {2:.3f}, {3:.3f})'
title_str = title_str.format(meanRho_cat, Q1Rho_cat, medRho_cat, Q3Rho_cat)
plt.title(title_str)
plt.hist(SamplesRho_cat, bins = 10)
plt.show()

f_out.close()

```

## References

Okamoto, Y. (2016). Reliability coefficients for ordinal categorical items: Actual relation of

observed and true scores. *Japan Women's University Journal: Faculty of Integrated Arts and Social Sciences*, 2016, 27, 113-122.

Okamoto, Y. (2017). Bayesian estimation of ordinal categorical items' reliability coefficients: Relationship between true values and observed values. *Japan Women's University Journal: Faculty of Integrated Arts and Social Sciences*, 2017, ??, ??????.