HW3

杜金鸿 15338039

Content

1 Problem Description

Read article: *Maximum Likelihood Algorithms for Generalized Linear Mixed Models* (McCulloch 1997)[1], and try to understand the basic concept of generalized linear mixed model (GLMM). Section 3.1 in this paper described a Monte Carlo EM (MCEM) method to derive Maximum Likelihood Estimates (MLE). Please use your own software (Matlab, R, C++, etc.) to perform following simulation study and answer related questions

1.1 Model and Notations

In this project, we consider a clustering problem. Suppose we have observed n observations, each observation is a binary process, i.e. the response $Y_{ij} = 0$ or 1, $i = 1,\dots,n$, $j = 1,\dots,T$. Here n is the number of subjects and T is the length of observation. In general, T might vary across subjects, time points may also be different. In this project, however, we simply assume that all subjects have common time length and time points. We also assume that these subjects belong to two clusters. For each cluster, the conditional expectation of response variable is:

$$P_{ij} \equiv \mathbb{E}(Y_{ij}|U_i = 1, X_{1,ij}, Z_{1,i}) = g^{-1}(\beta_1 X_{1,ij} + Z_{1,i})$$
$$P_{ij} \equiv \mathbb{E}(Y_{ij}|U_i = 2, X_{2,ij}, Z_{2,i}) = g^{-1}(\beta_2 X_{2,ij} + Z_{2,i}) \tag{1}$$

where U is cluster membership, $X_{c,ij}$ and $Z_{c,i}$ $(c = 1,2)$ are fixed and random effects, respectively. The link function $g^{-1}(x) = \dfrac{exp(x)}{1+exp(x)}$ is given. In a typical clustering problem, U is usually unknown, and hence we treat U as another random effect.

For random effects, we assume that $Z_{c,i} \sim N(0,\sigma_c^2)$ and $\mathbb{P}(U = 1) = \pi_1$ (then $\pi_2 = 1-\pi_1$). Then the parameter to be estimated is $\Omega = \{\beta_1,\beta_2,\sigma_1,\sigma_2,\pi_1\}$. Treating random effects as missing data, one can write the complete data likelihood function as

$$L(\Omega|Y_{ij},U_i,Z_{U_i,i}) = \prod_{i=1}^{n}\prod_{c=1}^{2}\left\{\pi_c f_c(Z_{c,i})\left[\prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i})\right]\right\}^{\omega_{ic}} \tag{2}$$

where $f_c(Z_{c,i})$ is the density function of Normal distribution, $f_c(Y_{ij}|Z_{c,i}) = P_{ij}^{Y_{ij}}(1-P_{ij})^{1-Y_{ij}}$. ω_{ic} is the dummy variable of U_i, i.e.

$$\omega_{ic} = \begin{cases} 1 & \text{,if subject } i \text{ belongs to cluster } c \\ 0 & \text{,otherwise} \end{cases}$$

1.2 Simulation Setup and Requirement

Generate 100 simulations. In each simulation, set $n = 100$ and $T = 10$. The true values of parameter are: $\beta_1 = 1$, $\beta_2 = 5$, $\pi_1 = 0.6$, $\sigma_1 = 2$, and $\sigma_2 = 10$.

Before you start, use $N(0,1)$ to generate the fixed effect X, and use them for all 100 simulations. Please follow the paper mentioned earlier and use MCEM to evaluate the loglikelihood function. In the E-step, perform $K = 500$ Gibbs sampling incorporated with a Metropolis-Hastings step, and drop the first 100 as a burn-in procedure.

1.3 Your Report

(1) Please write down the form of Monte Carlo average of log-likelihood (which you are going to evaluate)

(2) Please write down details of the EM algorithm you use for this simulation, especially the Metropolis-Hastings steps.

(3) What are your initial values? What is your convergence rule?

(4) How to accelerate your EM algorithm? Any improvement you can observe?

(5) Try different numbers of simulations: $200, 300, \ldots, 1000$. And plot the corresponding MSE.

(6) Write a report in either Chinese or English. Please attach your code to your report.

2 Notations

Symbol	Definition
n	The number of subjects.
T	The length of observation.
N	The number of stimulations.
Y_{ij}	The ith subjects at the time j.
X_{ij}	The fixed effects of the ith subjects at the time j.
U_i	The random effects. The cluster that the ith subject belongs to.
ω_{ic}	The indicator wherter the ith subject belongs to cluster c or not.
$Z_{c,i}$	The random effects of the ith subjects belongs to cluster c.
$\Omega = \{\beta_1, \beta_2, \sigma_1, \sigma_2, \pi_1\}$	All parameters in the model.
Ω_{MC}	Monte Carlo estimator of Ω
$\Phi = \{\beta_1, \beta_2\}$	Parameters not related to missing data.
Φ_{MC}	Monte Carlo estimator of Φ
$\Psi = \{\sigma_1, \sigma_2, \pi_1\}$	Parameters related to missing data.
Ψ_{MC}	Monte Carlo estimator of Ψ
$Q(\cdot,\cdot)$	Q function.
$Q_{MC}(\cdot,\cdot)$	Monte Carlo estimator of Q function.
$P(\cdot,\cdot)$	The part of Q function only related to missing data.
$P_{MC}(\cdot,\cdot)$	Monte Carlo estimator of P function.
$R(\cdot,\cdot)$	$Q - P$.
$R_{MC}(\cdot,\cdot)$	Monte Carlo estimator of R function.
$h_{Z_{U_i,i}}(\cdot)$	Candidate density.
$A_{U_i,\mathbf{Y}_i}(\cdot,\cdot)$	The acceptance function.
$.^{(t)}$	The superscript indicates in the tth step in the EM algorithm.
$\cdot_{(k)}$	The subscript indicates the kth sample in E-step of the EM algorithm.
K	The total number of samples from Gibbs Sampler.
K_2	The total number of samples from Metropolis-Hastings step.
m	The number of samples from Gibbs Sampler after burn-in procedure.

3 Generalized Linear Mixed Model(GLMM)

3.1 Complete Data Likelihood

Let

$$\mathbf{Y} = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1T} \\ Y_{21} & Y_{22} & \cdots & Y_{2T} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{n1} & Y_{n2} & \cdots & Y_{nT} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} U_1 & U_2 & \cdots & U_n \end{bmatrix}^\top$$

$$\mathbf{Z} = \begin{bmatrix} Z_{U_1,1} & Z_{U_2,2} & \cdots & Z_{U_n,n} \end{bmatrix}^\top \qquad \mathbf{Z}_1 = \begin{bmatrix} Z_{1,1} & Z_{1,2} & \cdots & Z_{1,n} \end{bmatrix}^\top \qquad \mathbf{Z}_2 = \begin{bmatrix} Z_{2,1} & Z_{2,2} & \cdots & Z_{2,n} \end{bmatrix}^\top$$

$$\mathbf{X}_1 = \begin{bmatrix} X_{1,11} & X_{1,12} & \cdots & X_{1,1T} \\ X_{1,21} & X_{1,22} & \cdots & X_{1,2T} \\ \vdots & \vdots & \ddots & \vdots \\ X_{1,n1} & X_{1,n2} & \cdots & X_{1,nT} \end{bmatrix} \qquad \mathbf{X}_2 = \begin{bmatrix} X_{2,11} & X_{2,12} & \cdots & X_{2,1T} \\ X_{2,21} & X_{2,22} & \cdots & X_{2,2T} \\ \vdots & \vdots & \ddots & \vdots \\ X_{2,n1} & X_{2,n2} & \cdots & X_{2,nT} \end{bmatrix}$$

Given the mixture distributions (π_1, π_2) and the necesssary parameters for each component Ω, we can write the likelihood as

$$\begin{aligned} L(\Omega|\mathbf{Y},\mathbf{U},\mathbf{Z}) &= \prod_{i=1}^{n}\prod_{j=1}^{T} \pi_{U_i} p(Y_{ij}, Z_{U_i,i}|\Omega) \\ &= \prod_{i=1}^{n}\prod_{j=1}^{T} \pi_{U_i} f_{U_i}(Z_{U_i,i}) f_c(Y_{ij}|Z_{U_i,i}) \\ &= \prod_{i=1}^{n} \pi_{U_i} f_{U_i}(Z_{U_i,i}) \left[\prod_{j=1}^{T} f_{U_i}(Y_{ij}|Z_{U_i,i})\right] \\ &= \prod_{i=1}^{n}\prod_{c=1}^{2} \left\{ \pi_c f_c(Z_{c,i}) \left[\prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i})\right] \right\}^{\omega_{ic}} \end{aligned}$$

since we know that either U_i equals to 1 or 2.

Random effects $\mathbf{U}$ and $\mathbf{Z}$ are called the missing values (or latent varaibles) and $(\mathbf{Y},\mathbf{U},\mathbf{Z})$ is called complete data[2]. The distribution of $\mathbf{U}$ depends on π_1 and the distribution of $\mathbf{Z}$ depends on $\mathbf{U}$, σ_1 and σ_2. Let $\Psi = \{\pi_1, \sigma_1, \sigma_2\}$, $\Phi = \{\beta_1, \beta_2\}$. Let

$$\begin{aligned} f_{Y_{ij}|(U_i,Z_{U_i,i})}(Y_{ij}|U_i, Z_{U_i,i}, \Phi) &= f_{U_i}(Y_{ij}|Z_{U_i,i}) \\ f_{(U_i,Z_{U_i,i})}(U_i, Z_{U_i,i}|\Psi) &= \pi_{U_i} f_{U_i}(Z_{U_i,i}) \end{aligned}$$

Then we can write the likelihood as

$$\begin{aligned} L(\Omega|\mathbf{Y},\mathbf{U},\mathbf{Z}) &= \prod_{i=1}^{n}\left[f_{(U_i,Z_{U_i,i})}(U_i, Z_{U_i,i}|\Psi) \prod_{j=1}^{T} f_{Y_{ij}|(U_i,Z_{U_i,i})}(Y_{ij}|U_i, Z_{U_i,i}|\Phi)\right] \\ &= \left[\prod_{i=1}^{n} f_{(U_i,Z_{U_i,i})}(U_i, Z_{U_i,i}|\Psi)\right] \cdot \left[\prod_{i=1}^{n}\prod_{j=1}^{T} f_{Y_{ij}|(U_i,Z_{U_i,i})}(Y_{ij}|U_i, Z_{U_i,i}|\Phi)\right] \end{aligned}$$

$$\triangleq f_{(\mathbf{U},\mathbf{Z})}(\mathbf{U},\mathbf{Z}|\Psi)\cdot f_{\mathbf{Y}|(\mathbf{U},\mathbf{Z})}(\mathbf{Y}|\mathbf{U},\mathbf{Z},\Phi) \tag{3}$$

3.2 Complete Log-Likelihood

The complete log-likelihood is given by

$$\begin{aligned}
l(\Omega|\mathbf{Y},\mathbf{U},\mathbf{Z}) &= \ln L(\Omega|\mathbf{Y},\mathbf{U},\mathbf{Z}) \\
&= \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\ln\left\{\pi_c f_c(Z_{c,i})\left[\prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i})\right]\right\} \\
&= \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\left[\ln\pi_c + \ln f_c(Z_{c,i}) + \sum_{j=1}^{T}\ln f_c(Y_{ij}|Z_{c,i})\right] \\
&= \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\left\{\ln\pi_c - \frac{Z_{c,i}^2}{2\sigma_c^2} - \frac{1}{2}\ln(2\pi\sigma_c^2) + \sum_{j=1}^{T}[Y_{ij}\ln P_{ij} + (1-Y_{ij})\ln(1-P_{ij})]\right\} \\
&= n\sum_{c=1}^{2}\omega_{ic}\left[\ln\pi_c - \frac{1}{2}\ln(2\pi\sigma_c^2)\right] - \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_c\frac{Z_{c,i}^2}{2\sigma_c^2} \\
&\quad + \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\sum_{j=1}^{T}[Y_{ij}(\beta_c X_{c,ij} + Z_{c,i}) - Y_{ij}\ln(1+e^{\beta_c X_{c,ij}+Z_{c,i}}) - (1-Y_{ij})\ln(1+e^{\beta_c X_{c,ij}+Z_{c,i}})] \\
&= n\sum_{c=1}^{2}\omega_{ic}\left[\ln\pi_c - \frac{1}{2}\ln(2\pi\sigma_c^2)\right] \\
&\quad + \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_c\left\{-\frac{Z_{c,i}^2}{2\sigma_c^2} + \sum_{j=1}^{T}[Y_{ij}(\beta_c X_{c,ij} + Z_{c,i}) - \ln(1+e^{\beta_c X_{c,ij}+Z_{c,i}})]\right\}
\end{aligned}$$

which is equal to

$$\begin{aligned}
l(\Omega|\mathbf{Y},\mathbf{U},\mathbf{Z}) &= \sum_{i=1}^{n}\ln f_{(U_i,Z_{U_i,i})}(U_i,Z_{U_i,i}|\Psi) + \sum_{i=1}^{n}\sum_{j=1}^{T}\ln f_{Y_{ij}|(U_i,Z_{U_i,i})}(Y_{ij}|U_i,Z_{U_i,i},\Phi) \\
&\triangleq \ln f_{(\mathbf{U},\mathbf{Z})}(\mathbf{U},\mathbf{Z}|\Psi) + \ln f_{\mathbf{Y}|(\mathbf{U},\mathbf{Z})}(\mathbf{Y}|\mathbf{U},\mathbf{Z},\Phi)
\end{aligned} \tag{4}$$

3.3 MLE

The partial derivatives of the parameters are given by

$$\begin{aligned}
\frac{\partial l}{\partial\pi_1} &= \sum_{i=1}^{n}\mathbb{1}_{\{U_i=1\}}\frac{1}{\pi_1} - \sum_{i=1}^{n}\mathbb{1}_{\{U_i=2\}}\frac{1}{1-\pi_1} \\
\frac{\partial l}{\partial\sigma_c^2} &= \sum_{i=1}^{n}\mathbb{1}_{\{U_i=c\}}\left(-\frac{1}{2\sigma_c^2} + \frac{Z_{c,i}^2}{2\sigma_c^4}\right) \\
\frac{\partial l}{\partial\beta_c} &= \sum_{i=1}^{n}\mathbb{1}_{\{U_i=c\}}\sum_{j=1}^{T}\left[Y_{ij}X_{c,ij} - \frac{X_{c,ij}e^{\beta_c X_{c,ij}+Z_{c,i}}}{1+e^{\beta_c X_{c,ij}+Z_{c,i}}}\right]
\end{aligned}$$

$$= \sum_{i=1}^{n} \mathbb{1}_{\{U_i=c\}} \sum_{j=1}^{T} X_{c,ij} \left[Y_{ij} - 1 + \frac{1}{1+e^{\beta_c X_{c,ij} + Z_{c,i}}} \right] \tag{5}$$

By setting the first two partial derivatives to 0, we get the maximum likelihood estimators

$$\hat{\pi}_{MLE,1} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{U_i=1\}}$$

$$\hat{\sigma}_{MLE,c} = \sqrt{\frac{\sum_{i=1}^{n} \mathbb{1}_{\{U_i=c\}} Z_{c,i}^2}{\sum_{i=1}^{n} \mathbb{1}_{\{U_i=c\}}}}$$

However the MLE of β_c is hard to obtain as a close form. We can use Netwon-Raphson algorithm to approximate it.

If we know the real data - **U**, **Z**, $\mathbf{X}_1$, $\mathbf{X}_2$ and **Y** - then we can calculate the MLE of these parameters.

4 MCEM

4.1 EM

By taking expectation of $\mathbf{U}$ and $\mathbf{Z}$ given $\mathbf{Y}$ under the current estimate of the parameters $\Omega^{(t)}$ ($\mathbf{U}$ and $\mathbf{Z}$ depend on $\Omega^{(t)}$), we have

$$\begin{aligned}
Q(\Omega,\Omega^{(t)}) &= \mathbb{E}_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})} \ln L(\Omega|\mathbf{Y},\mathbf{U},\mathbf{Z}) \\
&= \mathbb{E}_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})} \left\{ \sum_{i=1}^{n}\sum_{c=1}^{2} \omega_{ic} \left[\ln\pi_c - \frac{1}{2}\ln(2\pi\sigma_c^2) - \frac{Z_{c,i}^2}{2\sigma_c^2} + \sum_{j=1}^{T}[Y_{ij}(\beta_c X_{c,ij} + Z_{c,i}) - \ln(1+e^{\beta_c X_{c,ij}+Z_{c,i}})] \right] \right\} \\
&= \sum_{i=1}^{n}\sum_{c=1}^{2} \mathbb{E}_{\mathbf{U}|(\mathbf{Y},\Omega^{(t)})}(\omega_{ic}) \left[\ln\pi_c - \frac{1}{2}\ln(2\pi\sigma_c^2) \right] \\
&\quad + \sum_{i=1}^{n}\sum_{c=1}^{2} \mathbb{E}_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})} \left\{ \omega_{ic} \left[-\frac{Z_{c,i}^2}{2\sigma_c^2} + \sum_{j=1}^{T}[Y_{ij}(\beta_c X_{c,ij} + Z_{c,i}) - \ln(1+e^{\beta_c X_{c,ij}+Z_{c,i}})] \right] \right\}
\end{aligned}$$

The EM algorithm is given as following:

Algorithm 1 Expectation-Maximization

```
1: Start with the initial value $\Omega^{(0)}$. Set $t = 0$.
2: E-step: Calculate $Q(\Omega,\Omega^{(t)})$.
3: M-step: $\Omega^{(t+1)} \leftarrow \arg\max_{\Omega} Q(\Omega,\Omega^{(t)})$.
4: $t \leftarrow t+1$.
5: Repeat step 2-4 until convergence and then output the maximum likelihood estimators $\Omega^{(t)}$.
```

Notice that in the (4), after taking expectation of $\mathbf{U}$ and $\mathbf{Z}$ given $\mathbf{Y}$ under the current estimate of the parameters $\Omega^{(t)}$, Ψ is only related to the first term and Φ is only related to the second term.

$$\begin{aligned}
Q(\Omega,\Omega^{(t)}) &= \mathbb{E}_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})} \ln f_{(\mathbf{U},\mathbf{Z})}(\mathbf{U},\mathbf{Z}|\Psi) + \mathbb{E}_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})} \ln f_{\mathbf{Y}|(\mathbf{U},\mathbf{Z})}(\mathbf{Y}|\mathbf{U},\mathbf{Z},\Phi) \\
&\triangleq P(\Omega,\Omega^{(t)}) + R(\Omega,\Omega^{(t)})
\end{aligned}$$

Then we can decompose the M-step into 2 seperate updating steps.

Algorithm 2 Expectation-Maximization with Decomposed M-step

```
1: Start with the initial value $\Omega^{(0)}$. Set $t = 0$.
2: E-step: Calculate $Q(\Omega,\Omega^{(t)})$.
3: M-step: a. $\Phi^{(t+1)} \leftarrow \arg\max_{\Phi} P(\Omega,\Omega^{(t)})$.
4:         b. $\Psi^{(t+1)} \leftarrow \arg\max_{\Psi} R(\Omega,\Omega^{(t)})$.
5:         c. $\Omega^{(t+1)} \leftarrow \Phi^{(t+1)} \cup \Psi^{(t+1)}$.
6: $t \leftarrow t+1$.
7: Repeat step 2-6 until convergence and then output the maximum likelihood estimators $\Omega^{(t)}$.
```

4.2 Monte Carlo Integrating

The above integral is actually hard to compute. Alternatively, we can use Monte Carlo Integrating to approximate it. Suppose that $\{(\mathbf{U}_{(k)},\mathbf{Z}_{(k)}),\ k=1,2,\cdots,K\} \overset{i.i.d.}{\sim} f_{(\mathbf{U},\mathbf{Z})|\mathbf{Y}}(\mathbf{U},\mathbf{Z}|\mathbf{Y},\Omega)$.

Notes: More specifically, the ith component of the sample points comes from the distribution $f_{(U_i,Z_{U_i,i})|\mathbf{Y}_i}(U_i,Z_{U_i,i}|\mathbf{Y}_i,\Omega)$. In other words, we can sample the different component independently since they are independent.

Then from **Mean Value Method**,

$$\begin{aligned}
Q(\Omega,\Omega^{(t)}) &\approx \frac{1}{m}\sum_{k=1}^{m}\sum_{\substack{i=1\\ c=U_{(k),i}}}^{n}\left[\ln\pi_c-\frac{1}{2}\ln(2\pi\sigma_c^2)-\frac{Z_{(k),c,i}^2}{2\sigma_c^2}+\sum_{j=1}^{T}[Y_{ij}(\beta_cX_{c,ij}+Z_{(k),c,i})-\ln(1+e^{\beta_cX_{c,ij}+Z_{(k),c,i}})]\right]\\
&\triangleq Q_{MC}(\Omega,\Omega^{(t)})\\
&=P_{MC}(\Omega,\Omega^{(t)})+R_{MC}(\Omega,\Omega^{(t)})
\end{aligned}$$

where the subscript (k) denotes for the kth sample point. We have for $c=1,2,$

$$\frac{\partial Q_{MC}(\Omega,\Omega^{(t)})}{\partial\pi_1}=\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=1\}}\frac{1}{\pi_1}-\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=2\}}\frac{1}{1-\pi_1} \tag{6.a}$$

$$\frac{\partial Q_{MC}(\Omega,\Omega^{(t)})}{\partial\sigma_c^2}=\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=c\}}\left(-\frac{1}{2\sigma_c^2}+\frac{Z_{(k),c,i}^2}{2\sigma_c^4}\right) \tag{6.b}$$

$$\begin{aligned}
\frac{\partial Q_{MC}(\Omega,\Omega^{(t)})}{\partial\beta_c}&=\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=c\}}\sum_{j=1}^{T}\left[Y_{ij}X_{c,ij}-\frac{X_{c,ij}e^{\beta_cX_{c,ij}+Z_{(k),c,i}}}{1+e^{\beta_cX_{c,ij}+Z_{(k),c,i}}}\right]\\
&=\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=c\}}\sum_{j=1}^{T}X_{c,ij}\left[Y_{ij}-1+\frac{1}{1+e^{\beta_cX_{c,ij}+Z_{(k),c,i}}}\right]
\end{aligned} \tag{6.c}$$

By setting the first two partial derivatives (6.a) and (6.b) to 0, we get the maximum likelihood estimators

$$\hat{\pi}_1=\frac{1}{mn}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=1\}} \tag{7.a}$$

$$\hat{\sigma}_c=\sqrt{\frac{\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=c\}}Z_{(k),c,i}^2}{\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=c\}}}} \tag{7.b}$$

To calculate the maximum likelihood estimator of β_c, we can use direct numerical maximization of likelihood by Newton-Raphson Method. The second order partial derivative is given by

$$\frac{\partial^2 Q_{MC}(\Omega,\Omega^{(t)})}{\partial\beta_c^2}=-\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_{(k),i}=c\}}\sum_{j=1}^{T}\frac{X_{c,ij}^2e^{\beta_cX_{c,ij}+Z_{(k),c,i}}}{(1+e^{\beta_cX_{c,ij}+Z_{(k),c,i}})^2}$$

Algorithm 3 Newton-Raphson Method

```
1: Initialize $\hat{\beta}_c^{(0)}$.
2: $t \leftarrow 0$.
3: $\hat{\beta}_c^{(t+1)} \leftarrow \hat{\beta}_c^{(t)} - \dfrac{\dfrac{\partial Q_{MC}(\Omega,\Omega^{(t)})}{\partial \beta_c}\Big|_{\hat{\beta}_c^{(t)}}}{\dfrac{\partial^2 Q_{MC}(\Omega,\Omega^{(t)})}{\partial \beta_c^2}\Big|_{\hat{\beta}_c^{(t)}}}$
4: Repeat step 2-3 until convergence.
```

4.3 Gibbs Sampling

From (3), we have

$$f_{U_i,Z_{U_i,i},\mathbf{Y}_i}(U_i, Z_{U_i,i}, \mathbf{Y}_i|\Omega) = \pi_{U_i} f_{U_i}(Z_{U_i,i}|\Psi) \prod_{j=1}^{T} f_{U_i}(Y_{ij}|Z_{U_i,i}, \Phi)$$

$$f_{(U_i,Z_{(U_i,i)})|\mathbf{Y}_i}(U_i, Z_{U_i,i}|\mathbf{Y}_i, \Omega) = \frac{f_{U_i,Z_{U_i,i},\mathbf{Y}_i}(U_i, Z_{U_i,i}, \mathbf{Y}_i|\Omega)}{f_{\mathbf{Y}_i}(\mathbf{Y}_i|\Omega)}$$

Since it is hard to calculate the prior distribution of $\mathbf{Y}_i$, it is difficult to sample directly from the multivariate distribution $f_{(U_i,Z_{(U_i,i)})|\mathbf{Y}_i}(U_i, Z_{U_i,i}|\mathbf{Y}_i, \Omega)$. We can use **Gibbs Sampling**, a Markov chain Monte Carlo (MCMC) algorithm to obtain a sequence of observations which are approximated from the multivariate distribution.

First, we need to calculate the conditional distributions

$$\frac{f_{(U_i,Z_{(U_i,i)})|\mathbf{Y}_i}(U_i, Z_{U_i,i}|\mathbf{Y}_i, \Omega)}{f_{Z_{(U_i,i)}|\mathbf{Y}_i}(Z_{U_i,i}|\mathbf{Y}_i, \Omega)} = f_{U_i|(Z_{U_i,i},\mathbf{Y}_i)}(U_i|Z_{U_i,i}, \mathbf{Y}_i)$$

and

$$\frac{f_{(U_i,Z_{(U_i,i)})|\mathbf{Y}_i}(U_i, Z_{U_i,i}|\mathbf{Y}_i, \Omega)}{f_{U_i|\mathbf{Y}_i}(U_i|\mathbf{Y}_i, \Omega)} = f_{Z_{U_i,i}|(U_i,\mathbf{Y}_i)}(Z_{U_i,i}|U_i, \mathbf{Y}_i)$$

Notes : The equations come from

$$\begin{aligned}
\frac{f_{(A,B)|C}(a,b|c)}{f_{B|C}(b,c)} &= \frac{\dfrac{f_{A,B,C}(a,b,c)}{f_C(c)}}{\dfrac{f_{B,C}(b,c)}{f_C(c)}} \\
&= \frac{f_{A,B,C}(a,b,c)}{f_{B,C}(b,c)} \\
&= f_{A|(B,C)}(a|b,c)
\end{aligned}$$

$$f_{Z_{U_i,i},\mathbf{Y}_i}(Z_{U_i,i}, \mathbf{Y}_i|\Omega) = \sum_{c=1}^{2} f_{U_i,Z_{U_i,i},\mathbf{Y}_i}(c, Z_{c,i}, \mathbf{Y}_i|\Omega)$$

$$
\begin{aligned}
&= \sum_{c=1}^{2} \pi_c f_c(Z_{c,i}|\Psi) \prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i},\Phi) \\
f_{U_i,\mathbf{Y}_i}(U_i,\mathbf{Y}_i|\Omega) &= \int_{\mathbb{R}} f_{U_i,Z_{U_i,i},\mathbf{Y}_i}(U_i,z,\mathbf{Y}_i|\Omega)\mathrm{d}z \\
&= \int_{\mathbb{R}} \pi_{U_i} f_{U_i}(z|\Psi) \prod_{j=1}^{T} f_{U_i}(Y_{ij}|z,\Phi)\mathrm{d}z \\
f_{U_i|(Z_{U_i,i},\mathbf{Y}_i)}(U_i|Z_{U_i,i},\mathbf{Y}_i) &= \frac{f_{U_i,Z_{U_i,i},\mathbf{Y}_i}(U_i,Z_{U_i,i},\mathbf{Y}_i|\Omega)}{f_{Z_{U_i,i},\mathbf{Y}_i}(Z_{U_i,i},\mathbf{Y}_i|\Omega)} \\
&= \frac{\pi_{U_i} f_{U_i}(Z_{U_i,i}|\Psi) \prod_{j=1}^{T} f_{U_i}(Y_{ij}|Z_{U_i,i},\Phi)}{\sum_{c=1}^{2} \pi_c f_c(Z_{U_i,i}|\Psi) \prod_{j=1}^{T} f_c(Y_{ij}|Z_{U_i,i},\Phi)} \qquad (8.a) \\
f_{Z_{U_i,i}|(U_i,\mathbf{Y}_i)}(Z_{U_i,i}|U_i,\mathbf{Y}_i) &= \frac{f_{U_i,Z_{U_i,i},\mathbf{Y}_i}(U_i,Z_{U_i,i},\mathbf{Y}_i|\Omega)}{f_{U_i,\mathbf{Y}_i}(U_i,\mathbf{Y}_i|\Omega)} \\
&= \frac{\pi_{U_i} f_{U_i}(Z_{U_i,i}|\Psi) \prod_{j=1}^{T} f_{U_i}(Y_{ij}|Z_{U_i,i},\Phi)}{\int_{\mathbb{R}} \pi_{U_i} f_{U_i}(z|\Psi) \prod_{j=1}^{T} f_{U_i}(Y_{ij}|z,\Phi)\mathrm{d}z} \qquad (8.b)
\end{aligned}
$$

In this case, (8.a) is easy to compute. However, the denominator of (8.b) is hard to obtain. To sample from (8.b), we will use Metropolis-Hastings Algorithm described in the next section.

Now we just assume that both of these two conditional distributions can be attained. Then, suppose that $(U_{(k),i}, Z_{(k),U_{(k),i},i})$ is the ith component of the kth sample, we want to draw the ith component of the $(k+1)$th sample. We draw

$$
\begin{aligned}
U_{(k+1),i} &\sim f_{U_i|Z_{U_i,i},\mathbf{Y}_i}(u|Z_{U_{(k),i},i},\mathbf{Y}_i,\Omega) \\
Z_{(k+1),U_{(k+1),i},i} &\sim f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(z|U_{(k+1),i},\mathbf{Y}_i,\Omega)
\end{aligned}
$$

To sum up, the Gibbs Sampler is given by

Algorithm 4 Gibbs Sampler

```
1: for i=1:n do
2:     Initialize $(U_{(0),i}, Z_{(0),U_{(0),i},i})$.
3:     $k \leftarrow 0$.
4:     for k=1:K do
5:         Draw $U_{(k+1),i} \sim f_{U_i|Z_{U_i,i},\mathbf{Y}_i}(u|Z_{U_{(k),i},i},\mathbf{Y}_i,\Omega)$.
6:         Draw $Z_{(k+1),U_{(k+1),i},i} \sim f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(z|U_{(k+1),i},\mathbf{Y}_i,\Omega)$.
7:     end for
8: end for
9: Burn-in procedure and return the last $m$ samples $\{(\mathbf{U}_{(k)}, \mathbf{Z}_{(k)}),\ k = 1,2,\cdots,m\}$.
```

4.4 Metropolis-Hastings Algorithm

Suppose that we have $z = Z_{(k),U_{(k),i},i}$. To sample $Z_{(k+1),U_{(k+1),i},i}$ from $f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(z|U_{(k+1),i},\mathbf{Y}_i,\Omega)$, let $h_{Z_{U_{(k),i},i}}(z)$ be the candidate distribution.

We draw $z^* \sim h_{Z_{U_i,i}}(z)$ and accept z^* as $Z_{(k+1),U_{(k+1),i},i}$ with probability

$$A_{U_i,\mathbf{Y}_i}(z,z^*) = \min\left\{1, \frac{f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(z^*|U_{(k+1),i},\mathbf{Y}_i,\Omega)h_{Z_{U_i,i}}(z|\Psi)}{f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(z|U_{(k+1),i},\mathbf{Y}_i,\Omega)h_{Z_{U_i,i}}(z^*|\Psi)}\right\}$$

otherwise, we retain $Z_{(k+1),U_{(k+1),i},i} = Z_{(k),U_{(k),i},i}$. The candidate distribution should be similar to $f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(z|U_{(k+1),i},\mathbf{Y}_i,\Omega)$. Since

$$f_{Z_{U_i,i}|U_i,\mathbf{Y}_i}(Z_{U_i,i}|U_i,\mathbf{Y}_i) \propto \pi_{U_i} f_{U_i}(z|\Psi)\prod_{j=1}^{T} f_{U_i}(Y_{ij}|z,\Omega)$$

we can choose $h_{Z_{U_i,i}}(z) = f_{U_i}(z|\Psi)$ and the acceptance function is

$$A_{U_i,\mathbf{Y}_i}(z,z^*) = \min\left\{1, e^{\sum_{j=1}^{T} Y_{ij}(z^*-z)} \prod_{j=1}^{T} \frac{1+e^{\beta_i X_{ij}+z}}{1+e^{\beta_i X_{ij}+z^*}}\right\}$$

The another advantage of the choice is that $h_{Z_{U_i,i}}(z)$ is only related to the class c and irrelevant to U_i. **Therefore we can generate the MH chains for $Z_{1,i}$ and $Z_{2,i}$ respectively before we sample U_i and then choose corresponding $Z_{U_i,i}$ based on U_i. By doing in this way, we can save much time.**

Algorithm 5 Metropolis Within Gibbs Sampler

```
1: for i=1:n do
2:     Initialize $(U_{(0),i}, Z_{(0),1,i}, Z_{(0),2,i})$.
3:     for c=1:2 do
4:         $k \leftarrow 0$.
5:         for k=1:$K_2$ do
6:             Draw $z^* \sim f_c(z|\Psi)$.
7:             Accept $z^*$ as $Z_{(k+1),c,i}$ with probability $A_{U_i,\mathbf{Y}_i}(z,z^*)$; otherwise, retain $Z_{(k+1),c,i} = Z_{(k),c,i}$.
8:         end for
9:         Burn-in procedure and let the last $K+1$ samples be the final samples $\{Z_{(k),c,i},\ k=0,1,\cdots,K\}$.
10:    end for
11:    $k \leftarrow 0$.
12:    for k=1:$K$ do
13:        Draw $U_{(k+1),i} \sim f_{U_i|Z_{U_i,i},\mathbf{Y}_i}(u|Z_{U_{(k),i},i},\mathbf{Y}_i|\Omega)$.
14:    end for
15:    Let the last $m$ samples be the final samples $\{U_{(k),i},\ k=0,1,\cdots,K\}$.
16: end for
17: Burn-in procedure and return the $m$ samples $\{(\mathbf{U}_{(k)}, \mathbf{Z}_{(k),1}, \mathbf{Z}_{(k),2}),\ k=1,2,\cdots,m\}$.
```

The use of Metropolis-Hastings Algorithm is similar to **Importance Sampling**. Here we set $K_2 = 600$.

4.5 MCEM

Actually we don't known $f_{(\mathbf{U},\mathbf{Z})|\mathbf{Y}}(\mathbf{U},\mathbf{Z}|\mathbf{Y},\Omega)$, so we use $f_{(\mathbf{U},\mathbf{Z})|\mathbf{Y}}(\mathbf{U},\mathbf{Z}|\mathbf{Y},\Omega^{(t)})$ in the $(t+1)$th step to estimate the distribution. To generate $\{(\mathbf{U}_{(k)},\mathbf{Z}_{(k)}),\ k=1,2,\cdots,m\} \overset{i.i.d.}{\sim} f_{(\mathbf{U},\mathbf{Z})|\mathbf{Y}}(\mathbf{U},\mathbf{Z}|\mathbf{Y},\Omega^{(t)})$.

The Monte Carlo Expectation-Maximization Algorithm we use in every stimulation is given by

Algorithm 6 MCEM

```
1: Start with the initial value $\Omega^{(0)}$. Set $t = 0$.
2: E-step: a. Generate $m$ samples $\{(\mathbf{U}^{(t)}_{(k)},\mathbf{Z}^{(t)}_{(k),1},\mathbf{Z}^{(t)}_{(k),2}),\ k=1,2,\cdots,m\}$ from $f_{\mathbf{u}|\mathbf{Y}}(\mathbf{u}|\mathbf{Y},\Omega^{(t)})$ by Algorithm 5.
3:         b. Calculate the partial derivatives of $Q_{MC}(\Omega,\Omega^{(t)})$, the Monte Carlo estimator of $Q$, w.r.t every parameter.
4: M-step: a. $\Phi^{(t+1)} \leftarrow \arg\max_{\Phi} P_{MC}(\Omega,\Omega^{(t)})$ by Newton-Raphson Method.
5:         b. $\Psi^{(t+1)} \leftarrow \arg\max_{\Psi} R_{MC}(\Omega,\Omega^{(t)})$.
6:         c. $\Omega^{(t+1)} \leftarrow \Phi^{(t+1)} \cup \Psi^{(t+1)}$.
7: $t \leftarrow t+1$.
8: Repeat step 2-6 until convergence and then output the maximum likelihood estimators $\Omega^{(t)}$.
```

5 Acceleration

5.1 Louis Turbo EM

Since $\hat{\beta}_c$ are hard to be estimated, we can apply some acceleration methods to better estimate them and save time.

Recall the Newton-Raphson iteration for $\hat{\beta}_c$ in (5) is given by

$$\hat{\beta}_c^{(t+1)} = \hat{\beta}_c^{(t+1)} - \left[\frac{\partial^2 l}{\partial \beta_c^2}\bigg|_{\hat{\beta}_c^{(t)}} \right]^{-1} \frac{\partial l}{\partial \beta_c}\bigg|_{\hat{\beta}_c^{(t)}}$$

Let

$$H(\Omega, \Omega^{(t)}) = \mathbb{E}_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})}[\ln f_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega^{(t)})}(\mathbf{U},\mathbf{Z}|\mathbf{Y},\Omega^{(t)})]$$

From (5) and [3] we have

$$\begin{aligned}
\frac{\partial^2 H_c}{\partial \beta_c^2} &= -Var_{(\mathbf{U},\mathbf{Z})|(\mathbf{Y},\Omega)}\left(\frac{\partial l(\Omega)}{\partial \beta_c}\right) \\
&\approx -\frac{1}{m}\sum_{k=1}^{m}\left\{\sum_{i=1}^{n}\mathbb{1}_{\{U_i=c\}}\sum_{j=1}^{T}X_{c,ij}\left[Y_{ij}-1+\frac{1}{1+e^{\beta_c X_{c,ij}+Z_{(k),c,i}}}\right]\right\}^2 \\
&\quad + \left\{\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{n}\mathbb{1}_{\{U_i=c\}}\sum_{j=1}^{T}X_{c,ij}\left[Y_{ij}-1+\frac{1}{1+e^{\beta_c X_{c,ij}+Z_{(k),c,i}}}\right]\right\}^2 \\
&\triangleq \frac{\partial^2 H_{MC,c}(\Omega,\Omega^{(t)})}{\partial \beta_c^2} \\
\frac{\partial^2 l(\Omega)}{\partial \beta_c^2} &= \frac{\partial^2 Q(\Omega,\Omega^{(t)})}{\partial \beta_c^2} + \frac{\partial^2 H_c(\Omega,\Omega^{(t)})}{\partial \beta_c^2} \qquad (9)
\end{aligned}$$

The Taylor Expansion gives

$$0 = \frac{\partial Q(\Omega,\Omega^{(t)})}{\partial \beta_c}\bigg|_{\beta c=\hat{\beta}_{MC,c}} \approx \frac{\partial Q(\Omega,\Omega^{(t)})}{\partial \beta_c}\bigg|_{\beta c=\hat{\beta}_c^{(t)}} + \frac{\partial^2 Q(\Omega,\Omega^{(t)})}{\partial \beta_c^2}\bigg|_{\beta c=\hat{\beta}_c^{(t)}}(\hat{\beta}_{MC,c}-\hat{\beta}_c)$$

And therefore

$$\frac{\partial Q(\Omega,\Omega^{(t)})}{\partial \beta_c}\bigg|_{\beta_c=\hat{\beta}_c^{(t)}} \approx -\frac{\partial^2 Q(\Omega,\Omega^{(t)})}{\partial \beta_c^2}\bigg|_{\beta c=\hat{\beta}_c^{(t)}}(\hat{\beta}_{MC,c}-\hat{\beta}_c) \qquad (10)$$

From **Louis Turbo**[4], combining the above result (9) and (10) in one Newton-Raphson step, we update $\hat{\beta}_c^{(t+1)}$ based on the $\hat{\beta}_c^{(t)}$ and $\hat{\beta}_{MC,c}^{(t+1)}$, the estimator of E-step at $(t+1)$th step as following

$$\begin{aligned}
\hat{\beta}_c^{(t+1)} = \hat{\beta}_c^{(t)} &+ \left[\frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\bigg|_{\hat{\beta}_{MC,c}^{(t+1)}} + \frac{\partial^2 H_{MC,c}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\bigg|_{\hat{\beta}_{MC,c}^{(t+1)}}\right]^{-1} \\
&\cdot \left[\frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\bigg|_{\hat{\beta}_{MC,c}^{(t+1)}}\right](\hat{\beta}_{MC,c}^{(t+1)}-\hat{\beta}_c^{(t)}) \qquad (11)
\end{aligned}$$

where $\Omega_{MC}^{(t+1)}$ is the estimator after E-step. This relationship is an approximation useful only local to MLE and so should not be use until some EM iterations have been performed.

Algorithm 7 Louis Turbo EM (LTEM)

```
1: Start with the initial value $\Omega^{(0)}$. Set $t = 0$.
2: E-step: a. Generate $m$ samples $\{(\mathbf{U}_{(k)}^{(t)}, \mathbf{Z}_{(k),1}^{(t)}, \mathbf{Z}_{(k),2}^{(t)}),\ k = 1, 2, \cdots, m\}$ from $f_{\mathbf{u}|\mathbf{Y}}(\mathbf{u}|\mathbf{Y}, \Omega^{(t)})$ by Algorithm 4.
3:            b. Calculate the partial derivatives of $Q_{MC}(\Omega, \Omega^{(t)})$, the Monte Carlo estimator of $Q$, w.r.t every parameter.
4: M-step: a. $\Phi_{MC}^{(t+1)} \leftarrow \arg\max_{\Phi} P_{MC}(\Omega, \Omega^{(t)})$.
5:            b. $\Psi^{(t+1)} \leftarrow \arg\max_{\Psi} R_{MC}(\Omega, \Omega^{(t)})$.
6:            c. $\Omega_{MC}^{(t+1)} \leftarrow \Phi_{MC}^{(t+1)} \cup \Psi^{(t+1)}$.
7: NR-step: a. $\hat{\beta}_c^{(t+1)} \leftarrow \hat{\beta}_c^{(t)} + \left[\left(\frac{\partial^2 Q_{MC}(\Omega, \Omega_{MC}^{(t+1)})}{\partial \beta_c^2} + \frac{\partial^2 H_{MC}(\Omega, \Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\right)^{-1} \frac{\partial^2 Q_{MC}(\Omega, \Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\right]\Bigg|_{\hat{\beta}_{MC,c}^{(t+1)}} (\hat{\beta}_{MC,c}^{(t+1)} - \hat{\beta}_c^{(t)})$, $c \in \{1, 2\}$.
8:            b. $\Phi^{(t+1)} \leftarrow \{\hat{\beta}_1^{(t+1)}, \hat{\beta}_2^{(t+1)}\}$.
9:            c. $\Omega^{(t+1)} \leftarrow \Phi^{(t+1)} \cup \Psi^{(t+1)}$.
10: $t \leftarrow t + 1$.
11: Repeat step 2-10 until convergence and then output the maximum likelihood estimators $\Omega^{(t)}$.
```

Advantage of **Louis Turbo**:

(1) $\hat{\beta}_{MC,c}^{(t+1)}$ may be the best estimator with respect to the specifical samples at tth step and not exact the best to the original data. $\hat{\beta}_c^{(t)}$ helps to calibrate the estimating process.

(2) More pricise with less iterations if parameters are initialized properly.

5.2 Newton-Downhill Algorithm

Since $\hat{\beta}_c^{(t)}$ may not be close to $\hat{\beta}_{MC,c}^{(t+1)}$, the change of first and second order derivatives at $\hat{\beta}_c^{(t)}$ may be drastic and so the algorithm may be diverge. To deal with this case, we can use Newton-Downhill Algorithm to ameliorate Algorithm 7.

To prevent divergence, we need to ensure the updating process to be monotone,

$$\left| \frac{\partial Q_{MC}(\Omega, \Omega^{(t+1)})}{\partial \beta_c} \Bigg|_{\hat{\beta}_c^{(t+1)}} \right| \leq \left| \frac{\partial Q_{MC}(\Omega, \Omega^{(t+1)})}{\partial \beta_c} \Bigg|_{\hat{\beta}_c^{(t)}} \right|$$

since our purpose is to estimate the root of the equation $0 = \frac{\partial l}{\partial \beta_c} \approx \frac{\partial Q_{MC}}{\partial \beta_c}$.

To satisfy the above condition, we multiple $\left.\frac{\partial Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c}\right|_{\hat{\beta}_c^{(t+1)}}$ by coefficient $\lambda_c^{(t+1)} \in [0,1]$ so that the inequality holds. Then the update fomula (11) will become

$$\begin{aligned}\hat{\beta}_c^{(t+1)} &= \hat{\beta}_c^{(t)} - \lambda_c^{(t+1)} \left[\left.\frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\right|_{\hat{\beta}_{MC,c}^{(t+1)}} \right]^{-1} \left[\left.\frac{\partial Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c}\right|_{\hat{\beta}_{MC,c}^{(t+1)}} \right] \\ &\approx \hat{\beta}_c^{(t)} + \lambda_c^{(t+1)} \left[\left.\frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\right|_{\hat{\beta}_{MC,c}^{(t+1)}} + \left.\frac{\partial^2 H_{MC,c}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\right|_{\hat{\beta}_{MC,c}^{(t+1)}} \right]^{-1} \\ &\quad \cdot \left[\left.\frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2}\right|_{\hat{\beta}_{MC,c}^{(t+1)}} \right] (\hat{\beta}_{MC,c}^{(t+1)} - \hat{\beta}_c^{(t)}) \end{aligned} \tag{12}$$

In practice, we will initialize $\lambda_c^{(t+1)} = 1$ and halve it until the inequality holds. Therefore, the Monte Carlo Newton-Downhill Algorithm with Louis Method is given by

Algorithm 8 Newton-Downhill Louis Turbo EM (NDLTEM)

```
1:  Start with the initial value $\Omega^{(0)}$. Set $t = 0$.
2:  E-step: a. Generate $m$ samples $\{(\mathbf{U}_{(k)}^{(t)}, \mathbf{Z}_{(k),1}^{(t)}, \mathbf{Z}_{(k),2}^{(t)}),\ k = 1,2,\cdots,m\}$ from $f_{\mathbf{u}|\mathbf{Y}}(\mathbf{u}|\mathbf{Y},\Omega^{(t)})$ by Algorithm 5.
3:          b. Calculate the partial derivatives of $Q_{MC}(\Omega,\Omega^{(t)})$, the Monte Carlo estimator of $Q$, w.r.t every parameter.
4:  M-step: a. $\Phi_{MC}^{(t+1)} \leftarrow \arg\max_{\Phi} P_{MC}(\Omega,\Omega^{(t)})$.
5:          b. $\Psi^{(t+1)} \leftarrow \arg\max_{\Psi} R_{MC}(\Omega,\Omega^{(t)})$.
6:          c. $\Omega_{MC}^{(t+1)} \leftarrow \Phi_{MC}^{(t+1)} \cup \Psi^{(t+1)}$.
7:  ND-step: a. For $c = 1:2$ do
8:                $\lambda_c^{(t+1)} \leftarrow 1$.
9:                While $\left| \left.\frac{\partial Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c}\right|_{\beta_c^{(t)}} \right| < \lambda_c^{(t+1)} \left| \left.\frac{\partial Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c}\right|_{\beta_c^{(t+1)}} \right|$ do
10:                   $\lambda_c^{(t+1)} \leftarrow \frac{1}{2}\lambda_c^{(t+1)}$.
11:               EndWhile
12:               $\hat{\beta}_c^{(t+1)} \leftarrow \hat{\beta}_c^{(t)} + \lambda_c^{(t+1)} \left.\left[ \left( \frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2} + \frac{\partial^2 H_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2} \right)^{-1} \frac{\partial^2 Q_{MC}(\Omega,\Omega_{MC}^{(t+1)})}{\partial \beta_c^2} \right]\right|_{\hat{\beta}_{MC,c}^{(t+1)}}$
    $\cdot(\hat{\beta}_{MC,c}^{(t+1)} - \hat{\beta}_c^{(t)}),\ c \in \{1,2\}$.
13:            EndFor
14:         d. $\Phi^{(t+1)} \leftarrow \{\hat{\beta}_1^{(t+1)}, \hat{\beta}_2^{(t+1)}\}$.
15:         e. $\Omega^{(t+1)} \leftarrow \Phi^{(t+1)} \cup \Psi^{(t+1)}$.
16: $t \leftarrow t+1$.
17: Repeat step 2-16 until convergence and then output the maximum likelihood estimators $\Omega^{(t)}$.
```

6 Simulation

6.1 Assumptions

6.1.1 About Fixed Effects

In this experiment, I have tried to carry out simulations under two cases: fixed X effects and seperating X_1 and X_2 effects.

In the data exploration step, we can see that for one random initialization, the sample distribution of $\beta_c X_{c,ij} + Z_{c,i}$ with fixed X is hard to be seperated into 2 classes. And the estimated parameters also seem to be extremely inaccurate. Therefore, I decided to seperate X_1 and X_2 effects and the result in this case seems to be reasonable, which means that **we assume that X has different effects on different clusters for every observation**.

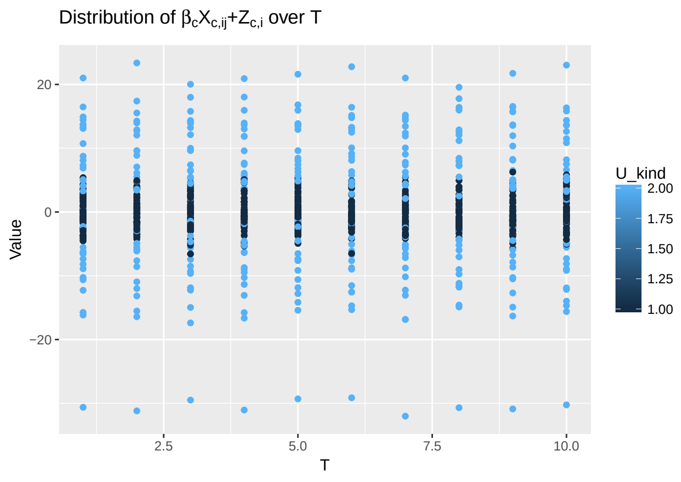

Figure 1: Sample Distribution of $\beta_c X_{c,ij} + Z_{c,i}$

6.1.2 About Initialization in Metropolis with Gibbs Sampler

We assume that $\mathbf{U}_{(0)}^{(t)}$, $\mathbf{Z}_{(0),1}^{(t)}$ and $\mathbf{Z}_{(0),2}^{(t)}$ are sample from the distributions of the real parameters. Also, the candidate distribution $h_{Z_{U_i,i}}(z) = f_{U_i}(z|\Psi)$ makes the acceptance function easy to be calculated.

6.2 Convergence Rules

For each parameter, we calculate the changing rate of it and let $\epsilon_{(t)}$ be as following

$$\epsilon^{(t)} = \max\left\{ \frac{|\hat{\pi}_1^{(t)} - \hat{\pi}_1^{(t-1)}|}{|\hat{\pi}_1^{(t-1)}| + \delta}, \frac{|\hat{\beta}_c^{(t)} - \hat{\beta}_c^{(t-1)}|}{|\hat{\beta}_c^{(t-1)}| + \delta}, \frac{|\hat{\sigma}_c^{(t)} - \hat{\sigma}_c^{(t-1)}|}{|\hat{\sigma}_c^{(t-1)}| + \delta} \middle| c = 1,2 \right\}$$

where $\delta > 0$ is to assure that the denominator is positive. Setting the threshold ϵ_0, if $\epsilon^{(t)} < \epsilon_0$ then we will consider the simulation converges.

$\epsilon^{(t)}$ describes the range of the fluctuation of all parameters. If we set ϵ_0 too small, it will need much time to converge and may even not converge. On the other hand, if we set ϵ_0 too big, the final parameters we get may not be the best.

In this experiment, we choose $\delta = 10^{-12}$, $\epsilon_0 = 2.5 \times 10^{-2}$.

6.3 Initialization

6.3.1 Fixed Case

EM Algorithm is sensitive to the initial values of parameters. We choose two fixed initialization.

The first choice $\{\pi_1^{(0)} = 0.5,\ \beta_1^{(0)} = 1.5,\ \beta_2^{(0)} = 3,\ \sigma_1^{(0)} = 4,\ \sigma_2^{(0)} = 8\}$. It is near the ground truth values.

The second choice is $\{\pi_1^{(0)} = 0.5,\ \beta_1^{(0)} = 3,\ \beta_2^{(0)} = 2,\ \sigma_1^{(0)} = 5,\ \sigma_2^{(0)} = 5\}$. It is far from the ground truth values.

6.3.2 Random Case

In this report we also implement the random initialization version of EM algorithm where the parameters are drawn from the below sets.

$$\pi_1^{(0)} \in \{0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7\}$$
$$\beta_1^{(0)} \in \{0,\ 0.5,\ 1,\ 1.5,\ 2\}$$
$$\beta_2^{(0)} \in \{4,\ 4.5,\ 5,\ 5.5,\ 6\}$$
$$\sigma_c^{(0)} \in \{2,\ 5,\ 8\}$$

6.4 Flow Chart

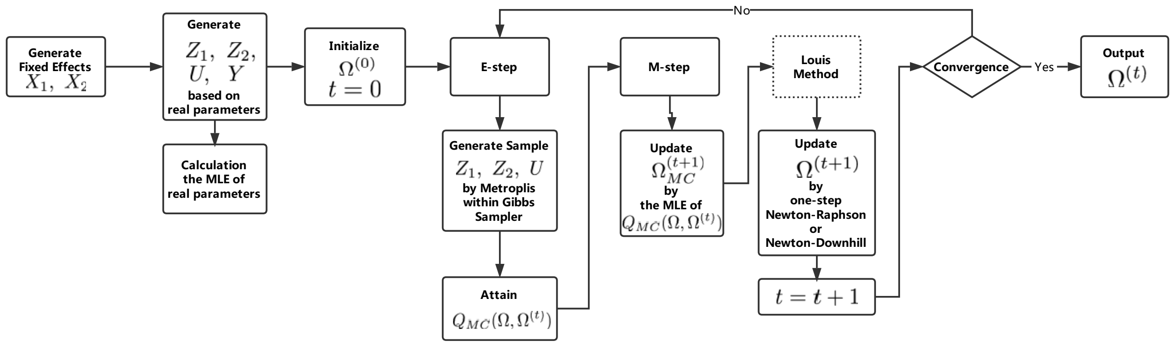
Generate Fixed Effects $X_1,\ X_2$
Generate $Z_1,\ Z_2,\ U,\ Y$ based on real parameters
Calculation the MLE of real parameters
Initialize $\Omega^{(0)}$ $t = 0$
E-step
Generate Sample $Z_1,\ Z_2,\ U$ by Metroplis within Gibbs Sampler
Attain $Q_{MC}(\Omega, \Omega^{(t)})$
M-step
Update $\Omega_{MC}^{(t+1)}$ by the MLE of $Q_{MC}(\Omega, \Omega^{(t)})$
Louis Method
Update $\Omega^{(t+1)}$ by one-step Newton-Raphson or Newton-Downhill
$t = t + 1$
Convergence
No
Yes
Output $\Omega^{(t)}$

Figure 2: Flow Chart For One Simulation

6.5 Result

From the below result of Table 1, Table 2 and Table 3, we can see that

1. All the EM algorithms rely on initial values and are easy to fall in local minimums. NDLTEM attains similar results with less time with a bad initialization. **It may be a good choice to first use NDLTEM to find good initial values and then use normal LTEM to find the better estimators**.
2. Louis Turbo accelerates the EM algorithm as we can see that LTEM attains better result with good initialization and random initialization. But it fails with bad initialization.
3. The MSE of $\hat{\beta}_2$ and $\hat{\sigma}_2$ by NDLTEM are much bigger than other parameters. This may be the result of the difference of the magnitudes.
4. With good initialization, we run NDLTEM for different numbers of stimulations. The result is given in Table 4. The MSE and average time lessen as N increases.

		π_1	β_1	β_2	σ_1	σ_2	Average Time
	MCEM	0.005471377	0.006810743	0.6891008	0.06012247	1.304981	20.85419
MSE	LTEM	0.004353057	0.006741913	0.6741328	0.05360465	1.279655	17.96544
	NDLTEM	0.004295486	0.006850141	0.7283367	0.0512091	1.497237	18.99893

Table 1: Simulations for different methods with $\pi_1^{(0)} = 0.5,\ \beta_1^{(0)} = 1.5,\ \beta_2^{(0)} = 3,\ \sigma_1^{(0)} = 4,\ \sigma_2^{(0)} = 8,\ N = 100$.

		π_1	β_1	β_2	σ_1	σ_2	Average Time
	MCEM	0.004921372	0.006783916	0.7905055	0.065711	1.061491	21.21759
MSE	LTEM	0.009848974	0.01426035	1.573184	0.1068302	1.075164	27.926
	NDLTEM	0.005615705	0.01009929	0.5256143	0.03342289	1.193098	19.28948

Table 2: Simulations for different methods with $\pi_1^{(0)} = 0.5,\ \beta_1^{(0)} = 3,\ \beta_2^{(0)} = 2,\ \sigma_1^{(0)} = 5,\ \sigma_2^{(0)} = 5,\ N = 100$.

		π_1	β_1	β_2	σ_1	σ_2	Average Time
	MCEM	0.004735782	0.009901802	0.7850846	0.03863162	1.256002	19.74115
MSE	LTEM	0.006905555	0.01014608	0.9102121	0.04530097	1.190026	16.5436
	NDLTEM	0.005472489	0.009881596	0.6214851	0.03280244	1.419779	19.51121

Table 3: Simulations for different methods with Random Initialization, $N = 100$.

	N	π_1	β_1	β_2	σ_1	σ_2	Average Time
MSE	100	0.00481093	0.006473371	0.7359094	0.06467358	1.239724	19.05205
	200	0.005005662	0.006982838	0.6983989	0.06072127	1.307294	19.31955
	300	0.004687861	0.007007113	0.6690311	0.05588335	1.251337	18.42706
	400	0.004762217	0.007276011	0.6812163	0.05891973	1.208865	17.7429
	500	0.004728446	0.007439576	0.6682399	0.05735861	1.15533	17.23455
	600	0.004845346	0.007192535	0.6785793	0.05753842	1.17558	17.06286
	700	0.004845532	0.007041375	0.6776848	0.05818015	1.168694	16.91251
	800	0.00481962	0.006914588	0.6746586	0.0584427	1.16743	16.82945
	900	0.004791556	0.006976409	0.6747397	0.05910764	1.169026	16.83027
	1000	0.004879037	0.007192204	0.6729532	0.05859428	1.164292	16.88205

Table 4: Simulations for different N by NDLTEM with $\pi_1^{(0)} = 0.5,\ \beta_1^{(0)} = 1.5,\ \beta_2^{(0)} = 3,\ \sigma_1^{(0)} = 4,\ \sigma_2^{(0)} = 8$.

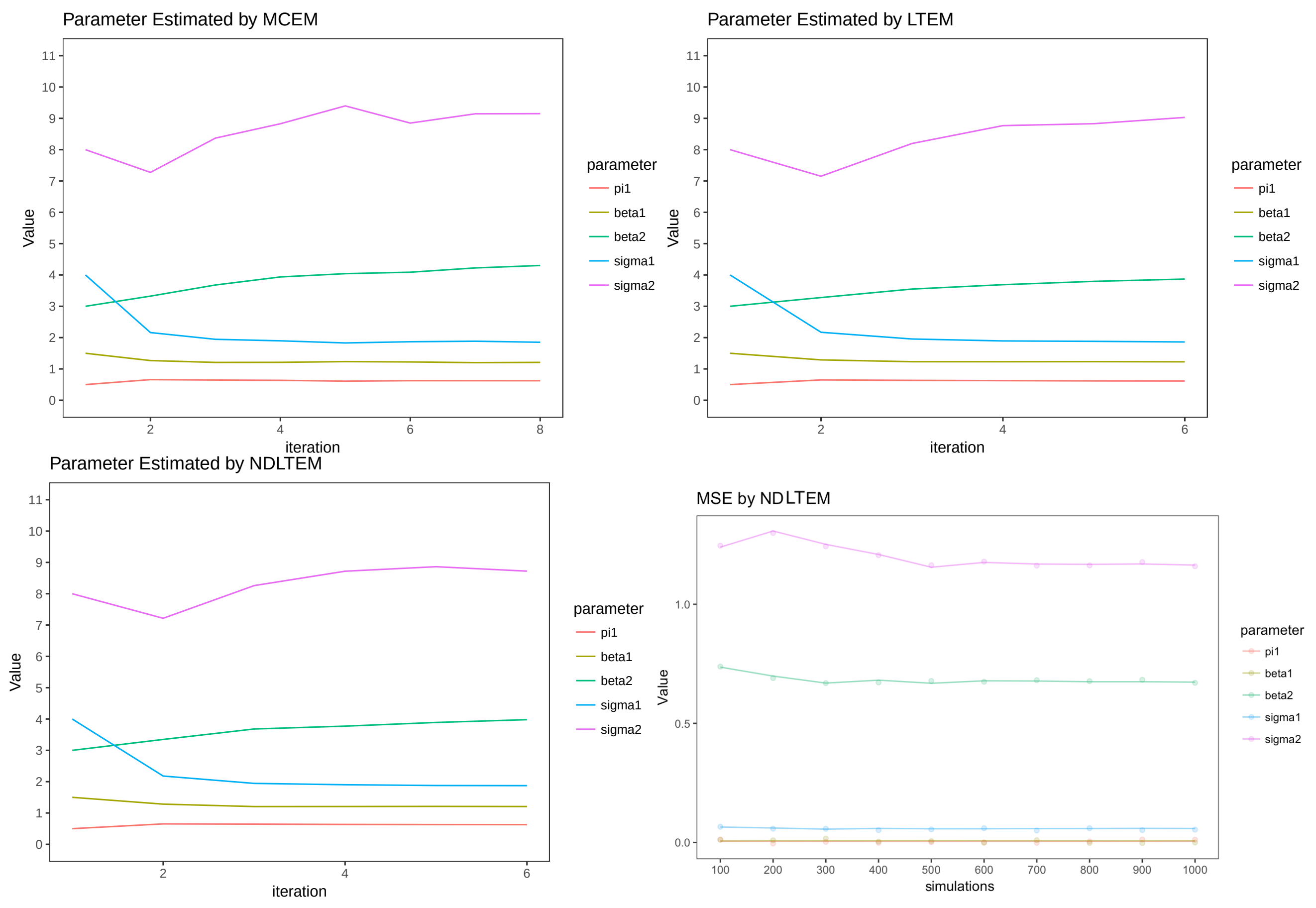

Figure 3: **subfigure $1-3$**: Examples of one simulation of all methods. **subfigure 4**: MSE of parameters over the number of simulations in NDLTEM.

The MSE is calculated as following

$$MSE_\theta = \frac{1}{N}\sum_{n=1}^{N}(\theta^{(n)} - \hat{\theta}^{(n)})^2$$

where $\theta \in \Omega$, $\theta^{(n)}$ is the true MLE of θ in the nth simulation and $\hat{\theta}^{(n)}$ is the EM estimator of $\theta^{(n)}$.

Appendix I Reference

[1] McCulloch, Charles E. “Maximum Likelihood Algorithms for Generalized Linear Mixed Models.”Journal of the American Statistical Association, vol. 92, no. 437, 1997, pp. 162 – 170. JSTOR, JSTOR, www.jstor.org/stable/2291460.

[2] https://en.wikipedia.org/wiki/Expectation-maximization_algorithm#Description

[3] Tanner M A. Tools for statistical inference[M]. New York: Springer, 1991.

[4] McLachlan G, Krishnan T. The EM algorithm and extensions[M]. John Wiley & Sons, 2007.

Appendix II R Code

Data Generation

```r
library(ggplot2)
library(reshape2)
library(latex2exp)
set.seed(1)
n <- 100
T_ <- 10
X1 <- matrix(rnorm(n*T_),n,T_)
X2 <- matrix(rnorm(n*T_),n,T_)
generate_data <- function(X1,
                          X2,
                          n = 100,
                          T_ = 10,
                          beta1 = 1,
                          beta2 = 1,
                          pi1 = 0.6,
                          sigma1 = 2,
                          sigma2 = 10){
    Z1 <- rnorm(n,0,sigma1)
    Z2 <- rnorm(n,0,sigma2)
    U <- 2 - rbinom(n,1,pi1)
    x <- ifelse(matrix(U,n,T_)==1, beta1*X1+matrix(Z1,n,T_), beta2*X2+matrix(Z2,n,T_))
    P <- exp(x)/(1+exp(x))
    Y <- ifelse(matrix(runif(n*T_),n,T_)<P, 1, 0)
    return(list(Y, Z1, Z2, U))
}
alldata <- generate_data(X1,X2)
Y <- alldata[[1]]
Z1 <- alldata[[2]]
Z2 <- alldata[[3]]
U <- alldata[[4]]
```

Data Exploration

```r
eta <- data.frame(t(ifelse(matrix(U,n,T_)==1,1*X1,1*X2) + matrix(ifelse(U==1,Z1,Z2),n,T_)))
eta['t'] <- c(1:10)
U_kind <- as.integer(rep(U,each=10))
ggplot(data = melt(eta,id = 't',variable.name="parameter") ,
       aes(x=t, y=value, colour= U_kind)) +
    geom_point()+
    labs(title=TeX("Distribution of $\\beta_{c}X_{c,ij}+Z_{c,i}$ over $T$"), x = 'T', y = 'Value')
```

Calculation for Real Parameters

```r
real_parameters <- function(X1,X2,Y,Z1,Z2,U,
                            n = 100,
                            T_ = 10){
    real_pi1 <- sum(U==1) / n
    real_sigma1 <- sqrt(mean(Z1[U==1]^2))
    real_sigma2 <- sqrt(mean(Z2[U==2]^2))
    epi <- 1
    real_beta1 <- 2
    while (epi>1e-3) {
        f2 <- sum(apply(X1^2 * exp(real_beta1 * X1 + matrix(Z1, n, T_)) /
                    (1 + exp(real_beta1 * X1 + matrix(Z1, n, T_)))^2, 1, sum)[U==1])
        if (f2==0) {
            break
        }
        new_beta1 <- real_beta1 + sum(apply(X1 * (Y - 1 + 1 /
                    (1 + exp(real_beta1 * X1 + matrix(Z1, n, T_)))), 1, sum)[U==1]) / f2
        epi <- abs(new_beta1 - real_beta1)
        real_beta1 <- new_beta1
    }
    epi <- 1
    real_beta2 <- 2
    while (epi>1e-3) {
        f2 <- sum(apply(X2^2 * exp(real_beta2 * X2 + matrix(Z2, n, T_)) /
                     (1 + exp(real_beta2 * X2 + matrix(Z2, n, T_)))^2, 1, sum)[U==2])
        if (f2==0) {
            break
        }
        new_beta2 <- real_beta2 + sum(apply(X2 * (Y - 1 + 1 /
                  (1 + exp(real_beta2 * X2 + matrix(Z2, n, T_)))), 1, sum)[U==2]) / f2
        epi <- abs(new_beta2 - real_beta2)
        real_beta2 <- new_beta2
    }
    return(list(real_pi1,real_beta1,real_beta2,real_sigma1,real_sigma2))
}
```

Metropolis-Hastings Algorithm

```
MH <- function(beta1,
               beta2,
               X1,
               X2,
               Y,
               K = 500,
               burnin = 100,
               n = 100,
               T_ = 10,
               sigma1 = 2,
               sigma2 = 10){
    Z1 <- rnorm(n, 0, sigma1)
    Z2 <- rnorm(n, 0, sigma2)

    Z1_MH <- matrix(0, n, K+burnin)
    Z2_MH <- matrix(0, n, K+burnin)
    for (i in c(1:(K+burnin))) {
         newZ1 <- rnorm(n, 0, sigma1)
         A1 <- apply(Y, 1, sum) * (newZ1 - Z1) + apply(
             log(1 +  exp(beta1 * X1 + matrix(Z1, n, T_))) -
                 log(1 + exp(beta1 * X1 + matrix(newZ1, n, T_))), 1, sum)
         Z1 <- ifelse(log(runif(n))>A1, Z1, newZ1)
         Z1_MH[,i] <- Z1

         newZ2 <- rnorm(n, 0, sigma2)
         A2 <- apply(Y, 1, sum) * (newZ2 - Z2) + apply(
             log(1 + exp(beta2 * X2 + matrix(Z2, n, T_))) -
                 log(1 + exp(beta2 * X2 + matrix(newZ2, n, T_))), 1, sum)
```

```
        Z2 <- ifelse(log(runif(n))>A2, Z2, newZ2)
        Z2_MH[,i] <- Z2
    }

    Z1 <- Z1_MH[,burnin]
    Z2 <- Z2_MH[,burnin]
    Z1_MH <- Z1_MH[,-c(1:burnin)]
    Z2_MH <- Z2_MH[,-c(1:burnin)]
    return(list(Z1, Z2, Z1_MH, Z2_MH))
}
```

Gibbs Sampling

```
Gibbs <- function(pi1,
                  beta1,
                  beta2,
                  sigma1,
                  sigma2,
                  X1,
                  X2,
                  Y,
                  K = 500,
                  burnin = 100,
                  n = 100,
                  T_ = 10){
    U_list <- matrix(0, nrow = n, ncol = K)


    Z_MH <- MH(beta1, beta2, X1, X2, Y, K = 500, burnin = 500)
    Z1 <- Z_MH[[1]]
    Z2 <- Z_MH[[2]]
    Z1_list <- Z_MH[[3]]
    Z2_list <- Z_MH[[4]]

    U <- 2 - rbinom(n, 1, 0.6)
    for (i in c(1:K)) {
        P1 <- exp(beta1 * X1 + matrix(ifelse(U==1, Z1, Z2), n, T_)) /
            (1 + exp(beta1 * X1 + matrix(ifelse(U==1, Z1, Z2), n, T_)))
        P2 <- exp(beta2 * X2 + matrix(ifelse(U==1, Z1, Z2), n, T_)) /
```

```
                (1 + exp(beta2 * X2 + matrix(ifelse(U==1, Z1, Z2), n, T_)))
            fU1 <- pi1 * dnorm(ifelse(U==1, Z1, Z2), 0, sigma1) *
                apply(ifelse(Y==1, P1, 1-P1), 1, prod)
            fU2 <- (1 - pi1) * dnorm(ifelse(U==1, Z1, Z2), 0, sigma2) *
                apply(ifelse(Y==1, P2, 1-P2), 1, prod)
            U <- ifelse(runif(n) < (fU1 / (fU1 + fU2)), 1, 2)
            U_list[,i] <- U
            Z1 <- Z1_list[,i]
            Z2 <- Z2_list[,i]
    }
    U_list <- U_list[,-c(1:burnin)]
    Z1_list <- Z1_list[,-c(1:burnin)]
    Z2_list <- Z2_list[,-c(1:burnin)]

    return(list(U_list, Z1_list, Z2_list))
}
```

MCEM

```
MCEM <- function(pi1,
                 beta1,
                 beta2,
                 sigma1,
                 sigma2,
                 epsilon0,
                 X1,
                 X2,
                 Y,
                 K = 500,
                 burnin = 100,
                 m = K - burnin,
                 n = 100,
                 T_ = 10,
                 is_plot = F){
    beta1_list <- c(beta1)
    beta2_list <- c(beta2)
    sigma1_list <- c(sigma1)
    sigma2_list <- c(sigma2)
    pi1_list <- c(pi1)
```

```
step <- 1
epsilon <- 1e10
while (epsilon > epsilon0) {
    gibbs <- Gibbs(pi1, beta1, beta2, sigma1, sigma2, X1, X2, Y)
    U_list <- gibbs[[1]]
    Z1_list <- gibbs[[2]]
    Z2_list <- gibbs[[3]]

    # Update parameters
    pi1 <- sum(U_list==1)/m/n
    sigma1 <- sqrt(mean(Z1_list[U_list==1]^2))
    sigma2 <- sqrt(mean(Z2_list[U_list==2]^2))
    epi <- 1
    beta1 <- 1
    while (epi>1e-3) {
        f2 <- sum(apply(Z1_list, 2, function(z){apply(
            X1^2 * exp(beta1 * X1 + matrix(z, n, T_)) /
                (1 + exp(beta1 * X1 + matrix(z, n, T_)))^2, 1, sum)})[U_list==1])
        if (f2==0) {
            break
        }
        new_beta1 <- beta1 + sum(apply(Z1_list, 2, function(z){apply(X1 *
                     (Y - 1 + 1 / (1 + exp(beta1 * X1 +
                     matrix(z, n, T_)))), 1, sum)})[U_list==1]) / f2
        epi <- abs(new_beta1 - beta1)
        beta1 <- new_beta1
    }
    epi <- 1
    beta2 <- 1
    while (epi>1e-3) {
        f2 <- sum(apply(Z2_list, 2, function(z){apply(X2^2 *
                exp(beta2 * X2 + matrix(z, n, T_)) / (1 +
                exp(beta2 * X2 + matrix(z, n, T_)))^2, 1, sum)})[U_list==2])
        if (f2==0) {
            break
        }
        new_beta2 <- beta2 + sum(apply(Z2_list, 2, function(z){apply(X2 *
                     (Y - 1 + 1 / (1 + exp(beta2 * X2 +
                     matrix(z, n, T_)))), 1, sum)})[U_list==2]) / f2
            epi <- abs(new_beta2 - beta2)
            beta2 <- new_beta2
        }

        epsilon <- max( abs(beta1 - beta1_list[step])/(abs(beta1_list[step]) + 1e-12),
                        abs(beta2 - beta2_list[step])/(abs(beta2_list[step]) + 1e-12),
                        abs(sigma1 - sigma1_list[step])/(abs(sigma1_list[step]) + 1e-12),
                        abs(sigma2 - sigma2_list[step])/(abs(sigma2_list[step]) + 1e-12),
                        abs(pi1 - pi1_list[step])/(abs(pi1_list[step]) + 1e-12))

        beta1_list[step+1] <- beta1
        beta2_list[step+1] <- beta2
        sigma1_list[step+1] <- sigma1
        sigma2_list[step+1] <- sigma2
        pi1_list[step+1] <- pi1
        step <- step + 1
    }
    if (is_plot) {
        result <- data.frame(beta1=beta1_list,
                             beta2 = beta2_list,
                             sigma1 = sigma1_list,
                             sigma2 = sigma2_list,
                             pi1 = pi1_list,
                             t=c(1:length(beta1_list)))
        print(ggplot(data = melt(result,id = 't',variable.name="parameter") ,
               aes(x=t, y=value, colour=parameter)) +
            geom_line()+
            coord_cartesian(ylim=c(0, 11)) +
            theme_bw() +
            theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
            scale_y_continuous(breaks=seq(0, 11, 1))+
            labs(title="Parameter Estimated by MCEM", x = 'iteration', y = 'Value'))
    }
    return(list(pi1, beta1, beta2, sigma1, sigma2))
}
```

Parameters in Random Initialization

```
pi1_grid <- c(0.3,0.4,0.5,0.6,0.7)
beta1_grid <- c(0,0.5,1,1.5,2)
beta2_grid <- c(0,0.5,1,1.5,2)
sigma1_grid <- c(2,5,8)
sigma2_grid <- c(2,5,8)
```

Simulations by MCEM

```
N <- 100
time_list <- c(1:N)
pi1_list <- c(1:N)
beta1_list <- c(1:N)
beta2_list <- c(1:N)
sigma1_list <- c(1:N)
sigma2_list <- c(1:N)
real_pi1_list <- c(1:N)
real_beta1_list <- c(1:N)
real_beta2_list <- c(1:N)
real_sigma1_list <- c(1:N)
real_sigma2_list <- c(1:N)
for(i in c(1:N)){
    t_start <- Sys.time()
    alldata <- generate_data(X1,X2)
    Y <- alldata[[1]]
    Z1 <- alldata[[2]]
    Z2 <- alldata[[3]]
    U <- alldata[[4]]
    parameters <- real_parameters(X1,X2,Y,Z1,Z2,U)
    real_pi1_list[i] <- parameters[[1]]
    real_beta1_list[i] <- parameters[[2]]
    real_beta2_list[i] <- parameters[[3]]
    real_sigma1_list[i] <- parameters[[4]]
    real_sigma2_list[i] <- parameters[[5]]
    # Initialization 1
    # parameters <- MCEM(0.5,1.5,0.5,4,11,2.5*1e-2,X1,X2,Y)
    # Initialization 2
```

```r
    # parameters <- MCEM(0.5,3,3,5,5,2.5*1e-2,X1,X2,Y)
    # Initialization 3
    # pi1 <- sample(pi1_grid,1)
    # beta1 <- sample(beta1_grid,1)
    # beta2 <- sample(beta2_grid,1)
    # sigma1 <- sample(sigma1_grid,1)
    # sigma2 <- sample(sigma2_grid,1)
    # parameters <- MCEM(pi1,beta1,beta2,sigma1,sigma2,2.5*1e-2,X1,X2,Y)
    t_end <- Sys.time()

    time_list[i] <- t_end - t_start
    pi1_list[i] <- parameters[[1]]
    beta1_list[i] <- parameters[[2]]
    beta2_list[i] <- parameters[[3]]
    sigma1_list[i] <- parameters[[4]]
    sigma2_list[i] <- parameters[[5]]
}

# MSE
cat(mean((pi1_list - real_pi1_list)^2),'\t&\t')
cat(mean((beta1_list - real_beta1_list)^2),'\t&\t')
cat(mean((beta2_list - real_beta2_list)^2),'\t&\t')
cat(mean((sigma1_list - real_sigma1_list)^2),'\t&\t')
cat(mean((sigma2_list - real_sigma2_list)^2),'\t&\t')
# Mean Time
cat(mean(time_list),'\n')
```

LTEM

```r
LTEM <- function(pi1,
                 beta1,
                 beta2,
                 sigma1,
                 sigma2,
                 epsilon0,
                 X1,
                 X2,
                 Y,
                 K = 500,
                 burnin = 100,
                 m = K - burnin,
                 n = 100,
                 T_ = 10,
                 is_plot = F){
    beta1_list <- c(beta1)
    beta2_list <- c(beta2)
    sigma1_list <- c(sigma1)
    sigma2_list <- c(sigma2)
    pi1_list <- c(pi1)

    step <- 1
    epsilon <- 1e10
    while (epsilon > epsilon0) {
        gibbs <- Gibbs(pi1, beta1, beta2, sigma1, sigma2, X1, X2, Y,K,burnin)
        U_list <- gibbs[[1]]
        Z1_list <- gibbs[[2]]
        Z2_list <- gibbs[[3]]

        # Update parameters
        pi1 <- sum(U_list==1)/m/n
        sigma1 <- sqrt(mean(Z1_list[U_list==1]^2))
        sigma2 <- sqrt(mean(Z2_list[U_list==2]^2))

        epi <- 1
        beta1 <- 1
        while (epi>1e-3) {
            f2 <- sum(apply(Z1_list, 2, function(z){apply(X1^2 *
                    exp(beta1 * X1 + matrix(z, n, T_)) / (1 +
                    exp(beta1 * X1 + matrix(z, n, T_)))^2, 1, sum)})[U_list==1])
            if (f2==0) {
                break
            }
            new_beta1 <- beta1 + sum(apply(Z1_list, 2, function(z){apply(X1 *
                         (Y - 1 + 1 / (1 + exp(beta1 * X1 +
                         matrix(z, n, T_)))), 1, sum)})[U_list==1]) / f2
            epi <- abs(new_beta1 - beta1)
            beta1 <- new_beta1
        }
        epi <- 1
        beta2 <- 5
        while (epi>1e-3) {
            f2 <- sum(apply(Z2_list, 2, function(z){apply(X2^2 *
                    exp(beta2 * X2 + matrix(z, n, T_)) / (1 +
                    exp(beta2 * X2 + matrix(z, n, T_)))^2, 1, sum)})[U_list==2])
            if (f2==0) {
                break
            }
            new_beta2 <- beta2 + sum(apply(Z2_list, 2, function(z){apply(X2 *
                        (Y - 1 + 1 / (1 + exp(beta2 * X2 +
                        matrix(z, n, T_)))), 1, sum)})[U_list==2]) / f2
            epi <- abs(new_beta2 - beta2)
            beta2 <- new_beta2
        }
        if(step>2){

            H2 <- -mean((apply(Z1_list, 2, function(z){apply(X1 *
                    (Y - 1 + 1 / (1 +
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)))), 1, sum)})[U_list==1])^2)
                + mean(apply(Z1_list, 2, function(z){apply(X1 *
                    (Y - 1 + 1 / (1 +
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)))), 1, sum)})[U_list==1])^2
            Q2 <- - mean(apply(Z1_list, 2, function(z){apply(X1^2 *
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)) / (1 +
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)))^2, 1, sum)})[U_list==1])
            if ((H2+Q2)!=0) {
                beta1 <- beta1_list[step] + Q2/(H2+Q2)*(beta1 - beta1_list[step])
            }
            H2 <- - mean((apply(Z2_list, 2, function(z){apply(X2 *
                    (Y - 1 + 1 / (1 +
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)))), 1, sum)})[U_list==2])^2)
                 + mean(apply(Z2_list, 2, function(z){apply(X2 *
                    (Y - 1 + 1 / (1 +
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)))), 1, sum)})[U_list==2])^2
            Q2 <- - mean(apply(Z2_list, 2, function(z){apply(X2^2 *
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)) / (1 +
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)))^2, 1, sum)})[U_list==2])
            if ((H2+Q2)!=0) {
                beta2 <- beta2_list[step] + Q2/(H2+Q2)*
                    (beta2 - beta2_list[step])
            }
        }
        epsilon <- max( abs(beta1 - beta1_list[step])/(abs(beta1_list[step]) + 1e-12),
                        abs(beta2 - beta2_list[step])/(abs(beta2_list[step]) + 1e-12),
                        abs(sigma1 - sigma1_list[step])/(abs(sigma1_list[step]) + 1e-12),
                        abs(sigma2 - sigma2_list[step])/(abs(sigma2_list[step]) + 1e-12),
                        abs(pi1 - pi1_list[step])/(abs(pi1_list[step]) + 1e-12))

        beta1_list[step+1] <- beta1
        beta2_list[step+1] <- beta2
        sigma1_list[step+1] <- sigma1
        sigma2_list[step+1] <- sigma2
        pi1_list[step+1] <- pi1

        step <- step + 1
    }
    if (is_plot) {
        result <- data.frame(pi1=pi1_list,
                             beta1=beta1_list,
                             beta2 = beta2_list,
                             sigma1 = sigma1_list,
                             sigma2 = sigma2_list,
                             t=c(1:length(pi1_list)))
        print(ggplot(data = melt(result,id = 't',variable.name="parameter") ,
               aes(x=t, y=value, colour=parameter)) +
            geom_line()+
            coord_cartesian(ylim=c(0, 11)) +
            scale_y_continuous(breaks=seq(0, 11, 1))+
            theme_bw() +
            theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
            labs(title="Parameter Estimated by LTEM", x = 'iteration', y = 'Value'))
    }
    return(list(pi1, beta1, beta2, sigma1, sigma2))
}
```

Simulation by LTEM

```r
N <- 100
time_list <- c(1:N)
pi1_list <- c(1:N)
beta1_list <- c(1:N)
beta2_list <- c(1:N)
sigma1_list <- c(1:N)
sigma2_list <- c(1:N)
real_pi1_list <- c(1:N)
real_beta1_list <- c(1:N)
real_beta2_list <- c(1:N)
real_sigma1_list <- c(1:N)
real_sigma2_list <- c(1:N)
for(i in c(1:N)){
    alldata <- generate_data(X1,X2)
    Y <- alldata[[1]]
    Z1 <- alldata[[2]]
    Z2 <- alldata[[3]]
    U <- alldata[[4]]
    real_data <- real_parameters(X1,X2,Y,Z1,Z2,U)
    real_pi1_list[i] <- real_data[[1]]
    real_beta1_list[i] <- real_data[[2]]
    real_beta2_list[i] <- real_data[[3]]
    real_sigma1_list[i] <- real_data[[4]]
    real_sigma2_list[i] <- real_data[[5]]
    t_start <- Sys.time()
    # parameters <- tryCatch(LTEM(0.5,1.5,3,4,8,2.5*1e-2,X1,X2,Y),
    #                        error = function(e){0*c(1:5)})
    # parameters <- tryCatch(LTEM(0.5,3,2,5,5,2.5*1e-2,X1,X2,Y),
    #                         error = function(e){0*c(1:5)})
    # pi1 <- sample(pi1_grid,1)
    # beta1 <- sample(beta1_grid,1)
    # beta2 <- sample(beta2_grid,1)
    # sigma1 <- sample(sigma1_grid,1)
    # sigma2 <- sample(sigma2_grid,1)
    # parameters <- tryCatch(LTEM(pi1,beta1,beta2,sigma1,sigma2,2.5*1e-2,X1,X2,Y),
    #                    error = function(e){0*c(1:5)})
    t_end <- Sys.time()
    time_list[i] <- t_end - t_start
    pi1_list[i] <- parameters[[1]]
    beta1_list[i] <- parameters[[2]]
    beta2_list[i] <- parameters[[3]]
    sigma1_list[i] <- parameters[[4]]
    sigma2_list[i] <- parameters[[5]]
}
diver_rate <- sum(pi1_list==0)/N
time_list <- time_list[pi1_list!=0]
beta1_list <- beta1_list[pi1_list!=0]
beta2_list <- beta2_list[pi1_list!=0]
sigma1_list <- sigma1_list[pi1_list!=0]
sigma2_list <- sigma2_list[pi1_list!=0]
real_beta1_list <- real_beta1_list[pi1_list!=0]
real_beta2_list <- real_beta2_list[pi1_list!=0]
real_sigma1_list <- real_sigma1_list[pi1_list!=0]
real_sigma2_list <- real_sigma2_list[pi1_list!=0]
real_pi1_list <- real_pi1_list[pi1_list!=0]
pi1_list <- pi1_list[pi1_list!=0]

# MSE
cat(mean((pi1_list - real_pi1_list)^2),'\t&\t')
cat(mean((beta1_list - real_beta1_list)^2),'\t&\t')
cat(mean((beta2_list - real_beta2_list)^2),'\t&\t')
cat(mean((sigma1_list - real_sigma1_list)^2),'\t&\t')
cat(mean((sigma2_list - real_sigma2_list)^2),'\t&\t')
# Mean Time
cat(mean(time_list),'\n')
```

NDLTEM

```
NDLTEM <- function(pi1,
                 beta1,
                 beta2,
                 sigma1,
                 sigma2,
                 epsilon0,
                 X1,
                 X2,
                 Y,
                 K = 500,
                 burnin = 100,
                 m = K - burnin,
                 n = 100,
                 T_ = 10,
                 is_plot = F){
    beta1_list <- c(beta1)
    beta2_list <- c(beta2)
    sigma1_list <- c(sigma1)
    sigma2_list <- c(sigma2)
    pi1_list <- c(pi1)

    step <- 1
    epsilon <- 1e10
    while (epsilon > epsilon0) {
        gibbs <- Gibbs(pi1, beta1, beta2, sigma1, sigma2, X1, X2, Y,K,burnin)
        U_list <- gibbs[[1]]
        Z1_list <- gibbs[[2]]
        Z2_list <- gibbs[[3]]

        # Update parameters
        pi1 <- sum(U_list==1)/m/n
        sigma1 <- sqrt(mean(Z1_list[U_list==1]^2))
        sigma2 <- sqrt(mean(Z2_list[U_list==2]^2))

        epi <- 1
        beta1 <- 1
        while (epi>1e-3) {
            f2 <- sum(apply(Z1_list, 2, function(z){apply(X1^2 *
                    exp(beta1 * X1 + matrix(z, n, T_)) / (1 +
                    exp(beta1 * X1 + matrix(z, n, T_)))^2, 1, sum)})[U_list==1])
            if(is.na(f2)){
                print(mean(U_list))
            }
            if (f2==0) {
                break
            }
            new_beta1 <- beta1 + sum(apply(Z1_list, 2, function(z){apply(X1 *
                        (Y - 1 + 1 / (1 + exp(beta1 * X1 +
                        matrix(z, n, T_)))), 1, sum)})[U_list==1]) / f2
            epi <- abs(new_beta1 - beta1)
            beta1 <- new_beta1
        }
        epi <- 1
        beta2 <- 1
        while (epi>1e-3) {
            f2 <- sum(apply(Z2_list, 2, function(z){apply(X2^2 *
                    exp(beta2 * X2 + matrix(z, n, T_)) / (1 +
                    exp(beta2 * X2 + matrix(z, n, T_)))^2, 1, sum)})[U_list==2])
            if (f2==0) {
                break
            }
            new_beta2 <- beta2 + sum(apply(Z2_list, 2, function(z){apply(X2 *
                        (Y - 1 + 1 / (1 + exp(beta2 * X2 +
                        matrix(z, n, T_)))), 1, sum)})[U_list==2]) / f2
            epi <- abs(new_beta2 - beta2)
            beta2 <- new_beta2
        }
        if(step>2){

            H2 <- -mean((apply(Z1_list, 2, function(z){apply(X1 *
                    (Y - 1 + 1 / (1 +
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)))), 1, sum)})[U_list==1])^2)
                + mean(apply(Z1_list, 2, function(z){apply(X1 *
                    (Y - 1 + 1 / (1 +
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)))^2), 1, sum)})[U_list==1])^2
            Q2 <- - mean(apply(Z1_list, 2, function(z){apply(X1^2 *
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)) / (1 +
                    exp(beta1_list[step] * X1 + matrix(z, n, T_)))^2, 1, sum)})[U_list==1])
            if (H2+Q2!=0) {
                Q1 <- - mean(apply(Z1_list, 2, function(z){apply(X1 *
                            (Y - 1 + 1 / (1 + exp(beta1_list[step] * X1 +
                                    matrix(z, n, T_)))), 1, sum)})[U_list==1])
                Q1_last <- - mean(apply(Z1_list, 2, function(z){apply(X1 *
                            (Y - 1 + 1 / (1 + exp(beta1_list[step-1] * X1 +
                                    matrix(z, n, T_)))), 1, sum)})[U_list==1])
                lambda <- 1
                while (lambda * abs(Q1) > abs(Q1_last)){
                    lambda <- lambda / 2
                }
                beta1 <- beta1_list[step] + lambda * Q2/(H2+Q2)*(beta1 - beta1_list[step])
            }
            H2 <- -mean((apply(Z2_list, 2, function(z){apply(X2 *
                    (Y - 1 + 1 / (1 +
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)))), 1, sum)})[U_list==2])^2)
                + mean(apply(Z2_list, 2, function(z){apply(X2 *
                    (Y - 1 + 1 / (1 +
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)))^2), 1, sum)})[U_list==2])^2
            Q2 <- - mean(apply(Z2_list, 2, function(z){apply(X2^2 *
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)) / (1 +
                    exp(beta2_list[step] * X2 + matrix(z, n, T_)))^2, 1, sum)})[U_list==2])
            if (H2+Q2!=0) {
                Q <- - mean(apply(Z2_list, 2, function(z){apply(X2 *
                            (Y - 1 + 1 / (1 + exp(beta2_list[step] * X2 +
                                    matrix(z, n, T_)))), 1, sum)})[U_list==2])
                Q_last <- - mean(apply(Z2_list, 2, function(z){apply(X2 *
                            (Y - 1 + 1 / (1 + exp(beta2_list[step-1] * X2 +
                                    matrix(z, n, T_)))), 1, sum)})[U_list==2])
                lambda <- 1
                while (lambda * abs(Q) > abs(Q_last)){
                    lambda <- lambda / 2
                }
                beta2 <- beta2_list[step] + lambda * Q2/(H2+Q2)*
                    (beta2 - beta2_list[step])
            }
        }
        epsilon <- max( abs(beta1 - beta1_list[step])/(abs(beta1_list[step]) + 1e-12),
                        abs(beta2 - beta2_list[step])/(abs(beta2_list[step]) + 1e-12),
                        abs(sigma1 - sigma1_list[step])/(abs(sigma1_list[step]) + 1e-12),
                        abs(sigma2 - sigma2_list[step])/(abs(sigma2_list[step]) + 1e-12),
                        abs(pi1 - pi1_list[step])/(abs(pi1_list[step]) + 1e-12))

        beta1_list[step+1] <- beta1
        beta2_list[step+1] <- beta2
        sigma1_list[step+1] <- sigma1
        sigma2_list[step+1] <- sigma2
        pi1_list[step+1] <- pi1

        step <- step + 1
    }
    if (is_plot) {
        result <- data.frame(pi1=pi1_list,
                             beta1=beta1_list,
                             beta2 = beta2_list,
                             sigma1 = sigma1_list,
                             sigma2 = sigma2_list,
                             t=c(1:length(pi1_list)))
        print(ggplot(data = melt(result,id = 't',variable.name="parameter") ,
               aes(x=t, y=value, colour=parameter)) +
            geom_line()+
            coord_cartesian(ylim=c(0, 11)) +
            scale_y_continuous(breaks=seq(0, 11, 1))+
            theme_bw() +
            theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
            labs(title="Parameter Estimated by NDLTEM", x = 'iteration', y = 'Value'))
    }
    return(list(pi1, beta1, beta2, sigma1, sigma2))
}
```

Simulation by NDLTEM

```
N <- 100
time_list <- c(1:N)
pi1_list <- c(1:N)
beta1_list <- c(1:N)
beta2_list <- c(1:N)
sigma1_list <- c(1:N)
sigma2_list <- c(1:N)
real_pi1_list <- c(1:N)
real_beta1_list <- c(1:N)
real_beta2_list <- c(1:N)
real_sigma1_list <- c(1:N)
real_sigma2_list <- c(1:N)
for(i in c(1:N)){
    alldata <- generate_data(X1,X2)
    Y <- alldata[[1]]
    Z1 <- alldata[[2]]
    Z2 <- alldata[[3]]
    U <- alldata[[4]]
    real_data <- real_parameters(X1,X2,Y,Z1,Z2,U)
    real_pi1_list[i] <- real_data[[1]]
    real_beta1_list[i] <- real_data[[2]]
    real_beta2_list[i] <- real_data[[3]]
    real_sigma1_list[i] <- real_data[[4]]
    real_sigma2_list[i] <- real_data[[5]]
    t_start <- Sys.time()
    # Initialization 1
    # parameters <- NDLTEM(0.5,1.5,3,4,8,2.5*1e-2,X1,X2,Y)
    # Initialization 2
    # parameters <- NDLTEM(0.5,3,2,5,5,2.5*1e-2,X1,X2,Y)
    # Initialization 3
    # pi1 <- sample(pi1_grid)
    # beta1 <- sample(beta1_grid,1)
    # beta2 <- sample(beta2_grid,1)
    # sigma1 <- sample(sigma1_grid,1)
    # sigma2 <- sample(sigma2_grid,1)
    # parameters <- NDLTEM(pi1,beta1,beta2,sigma1,sigma2,2.5*1e-2,X1,X2,Y)
    t_end <- Sys.time()

    time_list[i] <- t_end - t_start
    pi1_list[i] <- parameters[[1]]
    beta1_list[i] <- parameters[[2]]
    beta2_list[i] <- parameters[[3]]
    sigma1_list[i] <- parameters[[4]]
    sigma2_list[i] <- parameters[[5]]
}

mse_sigma1<-c(1:10)
mse_sigma2<-c(1:10)
mse_beta1<-c(1:10)
mse_beta2<-c(1:10)
mse_pi1 <- c(1:10)
for(N in c(1:10)){
    mse_pi1[N] <- mean((pi1_list[1:(N*100)] - real_pi1_list[1:(N*100)])^2)
    mse_sigma1[N] <- mean((sigma1_list[1:(N*100)] - real_sigma1_list[1:(N*100)])^2)
    mse_sigma2[N] <- mean((sigma2_list[1:(N*100)] - real_sigma2_list[1:(N*100)])^2)
    mse_beta1[N] <- mean((beta1_list[1:(N*100)] - real_beta1_list[1:(N*100)])^2)
    mse_beta2[N] <- mean((beta2_list[1:(N*100)] - real_beta2_list[1:(N*100)])^2)

    # MSE
    cat(mse_pi1[N],'\t&\t')
    cat(mse_sigma1[N],'\t&\t')
    cat(mse_sigma2[N],'\t&\t')
    cat(mse_beta1[N],'\t&\t')
    cat(mse_beta2[N],'\t&\t')
    # Mean Time
    cat(mean(time_list[1:(N*100)]),'\n')
}
```

Plot MSE

```
result <- data.frame(pi1 = mse_pi1,
                     beta1 = mse_beta1,
                     beta2 = mse_beta2,
                     sigma1 = mse_sigma1,
                     sigma2 = mse_sigma2,
                     t=c(1:10)*100)
print(ggplot(data = melt(result,id = 't',variable.name="parameter") ,
       aes(x=t, y=value, colour=parameter)) +
    geom_line()+
    geom_point() +
    scale_x_continuous(breaks=seq(0, 1000, 100))+
    theme_bw() +
    theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
    labs(title="MSE by NDLTEM", x = 'simulations', y = 'Value'))
```