Renmin University of China, School of Statistics
Bayesian Statistics, Autumn 2024

Final Project: MCEM for GLMM

王子翀
2022201379

January 2025

Abstract

This project conducted Monte Carlo Expectation Maximization(MCEM) algorithm to estimate the latent parameters of a Generalized Linear Mixture Model(GLMM). Theoretical derivation and algorithm implementation are provided. The simulation results show that the algorithm can estimate the parameters on some extent, but the stability and convergence need further improvement. Source code is available at `https://github.com/ZichongWang/MCEM`.

Content

1 Background

This project considers the parameter estimation problem of a Generalized Linear Mixed Model (GLMM). Suppose Y is a binary variable, and the data is divided into two clusters.

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

where U is cluster membership, $X_{c,ij}$ and $Z_{c,i}$ $(c = 1, 2)$ are fixed and random effects, respectively. The link function $g^{-1}(x) = \frac{\exp(x)}{1+\exp(x)}$ is given. We treat U as another random effect.

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

The goal is given the data Y_{ij} and X_{ij}, estimate the parameters Ω.

In the following sections, we will firstly derive the theoratical form of solusion, and then do simulations and estimate the parameters using the Monte Carlo EM algorithm.

2 Theoretical Derivation

2.1 Complete data Likelihood

In this section, we derive the complete data likelihood and its conditional expectation.

For group c, the prior of random effect $Z_{c,i}$ is:

$$f_c(Z_{c,i}) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{Z_{c,i}^2}{2\sigma_c^2}\right)$$

According to Logistic regression, the conditional probability is:

$$P_{ij} = \frac{\exp(\beta_c X_{c,ij} + Z_{c,i})}{1 + \exp(\beta_c X_{c,ij} + Z_{c,i})}$$

Thus the conditional distribution is

$$f_c(Y_{ij}|Z_{c,i}) = \begin{cases} P_{ij}, & \text{if } Y_{ij} = 1, \\ 1 - P_{ij}, & \text{if } Y_{ij} = 0 \end{cases}$$

which can be written as:

$$f_c(Y_{ij}|Z_{c,i}) = P_{ij}^{Y_{ij}}(1-P_{ij})^{1-Y_{ij}}$$

For the group c, prior probability is $P(U_i = c) = \pi_c$.

And for sample i, given $U_i = c$, the complete data likelihood is:

$$f(Y_{ij}, Z_{c,i}, U_i = c) = \pi_c f_c(Z_{c,i}) \prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i})$$

Compute the product, we get the likelihood:

$$\begin{aligned} L(\Omega|Y_{ij}, w_{i1}, w_{i2}, Z_{1,i}, Z_{2,i}) &= \prod_{i=1}^{n}\prod_{c=1}^{2}\left[\pi_c f_c(Z_{c,i}) \prod_{j=1}^{T} f_c(Y_{ij}|Z_{c,i})\right]^{\omega_{ic}} \\ &= \prod_{i=1}^{n}\prod_{c=1}^{2}\left[\pi_c \cdot \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{Z_{c,i}^2}{2\sigma_c^2}\right) \cdot \prod_{j=1}^{T}\left(P_{ij}^{Y_{ij}}(1-P_{ij})^{1-Y_{ij}}\right)\right]^{\omega_{ic}} \end{aligned}$$

And log-likelihood is:

$$\begin{aligned} &\log L(\Omega|Y_{ij}, w_{i1}, w_{i2}, Z_{1,i}, Z_{2,i}) \\ &= \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\left[\log\pi_c + \log f_c(Z_{c,i}) + \sum_{j=1}^{T}\log f_c(Y_{ij}|Z_{c,i})\right] \\ &= \sum_{i=1}^{n}\sum_{c=1}^{2}\omega_{ic}\left[\log\pi_c - \frac{1}{2}\log(2\pi\sigma_c^2) - \frac{Z_{c,i}^2}{2\sigma_c^2} + \sum_{j=1}^{T}\left(Y_{ij}\log P_{ij} + (1-Y_{ij})\log(1-P_{ij})\right)\right] \end{aligned}$$

2.2 E-step

In the E-step, we need to calculate the conditional expectation of the complete data log-likelihood.

$$\begin{aligned} Q(\Omega, \Omega^{(m)}) &= \sum_{i=1}^{n}\mathbb{E}[L_i(\Omega) \mid Y_{ij}, \Omega^{(m)}] \\ &\approx \frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{n} L_i(\Omega \mid Y_{ij}, w_{i1}^{(k)}, w_{i2}^{(k)}, Z_{1,i}^{(k)}, Z_{2,i}^{(k)}) \end{aligned}$$

Here, K is Monte Carlo sample size, and $w_i^{(k)}, Z_i^{(k)}$ are samples from the joint posterior distribution $P(w_i, Z_i \mid Y_{ij}, \Omega^{(m)})$. We use Gibbs sampling to sample $w_i^{(k)}, Z_i^{(k)}$.

2.2.1 Sampling $w_{ic}^{(k+1)}$

Recall that w_{ic} is a binary variable, indicating the cluster membership of sample i; And π_1 is the probability of sample i belonging to cluster 1.

Let $\widetilde{\pi}_{i1} = P(w_{i1} = 1 \mid Z_i^{(k)}, Y_{ij}, \Omega^{(m)})$, then

$$\widetilde{\pi}_{i1} = \frac{\pi_1^{(m)} \cdot A_1(Z_i^{(k)}, Y_{ij}, \Omega^{(m)})}{\pi_1^{(m)} \cdot A_1(Z_i^{(k)}, Y_{ij}, \Omega^{(m)}) + \pi_2^{(m)} \cdot A_2(Z_i^{(k)}, Y_{ij}, \Omega^{(m)})}$$

where

$$\begin{aligned} A_c(Z_i^{(k)}, Y_{ij}, \Omega^{(m)}) &= f_c(Z_i^{(k)} \mid \Omega^{(m)}) \left[\prod_{j=1}^{T} f_c(Y_{ij} \mid Z_i^{(k)}, \Omega^{(m)}) \right] \\ &= f_c(Z_i^{(k)} \mid \sigma_c^{(m)}) \left[\prod_{j=1}^{T} f_c(Y_{ij} \mid Z_i^{(k)}, \Omega^{(m)}) \right] \end{aligned}$$

To directly calculate the likelihood of Y_{ij} will encounter numerical problems, since the product of many small numbers will be very close to 0. We use $\log - \exp$ to solve it.

$$\prod_{j=1}^{T} f_c(Y_{ij} | Z_i^{(k)}, \Omega^{(m)}) = \prod_{j=1}^{T} \left[P_{ij}^{Y_{ij}} \cdot (1 - P_{ij})^{1 - Y_{ij}} \right].$$

Take log form

$$\log \left(\prod_{j=1}^{T} f_c(Y_{ij} | Z_i^{(k)}, \Omega^{(m)}) \right) = \sum_{j=1}^{T} \left[Y_{ij} \log P_{ij} + (1 - Y_{ij}) \log(1 - P_{ij}) \right].$$

And take exponential form

$$\prod_{j=1}^{T} f_c(Y_{ij} | Z_i^{(k)}, \Omega^{(m)}) = \exp \left(\sum_{j=1}^{T} \left[Y_{ij} \log P_{ij} + (1 - Y_{ij}) \log(1 - P_{ij}) \right] \right).$$

We generate

$$w_i^{(k+1)} \sim \text{Bernoulli}(\widetilde{\pi}_{i1})$$

This determines the cluster membership of sample i in next iteration.

2.2.2 Sampling $Z_i^{(k+1)}$

Recall that Z_i is the random effect part of sample i.

We generate

$$Z_i^{(k)} \sim f(Z_i \mid w_i^{(k+1)}, Y_{ij}, \Omega^{(m)})$$

Howerver, this density function have no cloes form, so we use Metropolis-Hastings algorithm to sample $Z_i^{(k+1)}$. Notice

$$\begin{aligned} f(Z_i \mid w_i^{(k+1)}, Y_{ij}, \Omega^{(m)}) &= \frac{f(Z_i, w_i^{(k+1)}, Y_{ij} \mid \Omega^{(m)})}{f(w_i^{(k+1)}, Y_{ij} \mid \Omega^{(m)})} \\ &= \frac{f(Y_{ij} \mid Z_i, w_i^{(k+1)}, \Omega^{(m)}) f(Z_i, w_i^{(k+1)} \mid \Omega^{(m)})}{f(w_i^{(k+1)}, Y_{ij} \mid \Omega^{(m)})} \\ &= \frac{f(Y_{ij} \mid Z_i, w_i^{(k+1)}, \Omega^{(m)}) f_c(Z_i \mid w_i^{(k+1)}, \Omega^{(m)})}{f(w_i^{(k+1)} \mid Y_{ij}, \Omega^{(m)}) / f(w_i^{(k+1)} \mid \Omega^{(m)})} \\ &\propto f(Y_{ij} \mid Z_i, w_i^{(k+1)}, \Omega^{(m)}) f_c(Z_i \mid w_i^{(k+1)}, \Omega^{(m)}) \end{aligned}$$

It turns to be a standard MH sampling problem: we have prior distribution and likelihood function, and want to sample the posterior distribution. We can directly use the prior distribution as the proposal distribution, and calculate the acceptance rate.

We first generate

$$Z_i^* \sim \begin{cases} \mathcal{N}(0, \sigma_1^{(m)2}), & \text{if } w_{i1} = 1, \\ \mathcal{N}(0, \sigma_2^{(m)2}), & \text{if } w_{i2} = 1 \end{cases}$$

Then calculate

$$\alpha^* = \min \left\{ 1, \quad \frac{\prod_{j=1}^{T} f_c(Y_{ij} \mid Z_i^*, w_{ic}^{(k+1)}, \Omega^{(m)})}{\prod_{j=1}^{T} f_c(Y_{ij} \mid Z_i^{(k)}, w_{ic}^{(k+1)}, \Omega^{(m)})} \right\}$$

If $\alpha^* > U(0,1)$, we accept $Z_i^{(k+1)} = Z_i^*$, otherwise we reject it, $Z_i^{(k+1)} = Z_i^{(k)}$.

2.2.3 Iteration

Recall our goal is using Monte Carlo to estimate the conditional expectation of the complete data log-likelihood, so we need a large number of w_i and Z_i samples. When k is large enough, the samples will converge to follow the real distribution. So we repeat sampling w_i and Z_i for K times, and discard the first K_0 samples as burn-in period. (Here we perform $K = 100$ and drop first 100 as a burn-in precedure.) The rest samples are used to estimate the conditional expectation.

2.3 M-step

From E-step we get a good estimation of

$$Q(\Omega, \Omega^{(m)}) \approx \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} L_i(\Omega \mid Y_{ij}, w_{i1}^{(k)}, w_{i2}^{(k)}, Z_{1,i}^{(k)}, Z_{2,i}^{(k)})$$

Notice Likelihood consists of three parts:

$$\begin{aligned} &\log L(\Omega | Y_{ij}, w_{i1}, w_{i2}, Z_{1,i}, Z_{2,i}) \\ &= \sum_{i=1}^{n} \sum_{c=1}^{2} \omega_{ic} \left[\log \pi_c + \log f_c(Z_{c,i}) + \sum_{j=1}^{T} \log f_c(Y_{ij} | Z_{c,i}) \right] \end{aligned}$$

Thus Q can be decomposed into three parts, depending ono mutually disjoint sets of parameters in Ω:

$$\begin{aligned} Q_1(\pi_c) &= \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \sum_{c=1}^{2} \omega_{ic}^{(k)} \log \pi_c \\ Q_2(\beta_c) &= \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \sum_{c=1}^{2} \omega_{ic}^{(k)} \sum_{j=1}^{T} \log f_c(Y_{ij} \mid Z_{c,i}^{(k)}) \\ Q_3(Z_{c,i}) &= \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \sum_{c=1}^{2} \omega_{ic}^{(k)} \log f_c(Z_{c,i}^{(k)}) \end{aligned}$$

2.3.1 Update π_c

The constrain is $\pi_1 + \pi_2 = 1$. The Lagrange function is:

$$\mathcal{L}(\pi_c, \lambda) = Q_1(\pi_c) + \lambda \left(1 - \sum_{c=1}^{2} \pi_c \right)$$

Take the derivative with respect to π_c and set it to 0:

$$\frac{\partial \mathcal{L}}{\partial \pi_c} = \frac{1}{\pi_c} \cdot \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \omega_{ic}^{(k)} - \lambda = 0$$

Then we get

$$\pi_c = \frac{\sum_{i=1}^{n} \sum_{k=1}^{K} \omega_{ic}^{(k)}}{n \cdot K}, \quad c = 1, 2$$

2.3.2 Update β_c

Here we need to seperately update β_1 and β_2. Here we use Gradient Ascent (since we need to maximize). For c group, the log-likelihood is:

$$\mathcal{L}(\beta_c) = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \sum_{j=1}^{T} \left[Y_{ij} \log P_{ij}^{(c)} + (1 - Y_{ij}) \log(1 - P_{ij}^{(c)}) \right]$$

And the gradient has closed form:

$$\frac{\partial \mathcal{L}(\beta_c)}{\partial \beta_c} = \frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{n}\sum_{j=1}^{T}\frac{\partial \mathcal{L}(\beta_c)}{\partial \eta_{ij}^{(c)}}\cdot\frac{\partial \eta_{ij}^{(c)}}{\partial \beta_c} = \frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{n}\sum_{j=1}^{T}\left(Y_{ij} - P_{ij}^{(c)}\right)X_{ij}$$

2.3.3 Update σ_c

It is a very typical MLE problem. Maximizing Q_3 is equivalent to minimizing the following function:

$$\mathcal{L}(\sigma_c^2) = \frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{n}\omega_{ic}^{(k)}\left[\frac{Z_{c,i}^2}{2\sigma_c^2} + \frac{1}{2}\log(2\pi\sigma_c^2)\right]$$

Take derivative with respect to σ_c^2 and set it to 0:

$$\frac{\partial \mathcal{L}(\sigma_c^2)}{\partial \sigma_c^2} = \frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{n}\omega_{ic}^{(k)}\left(-\frac{Z_{c,i}^2}{2(\sigma_c^2)^2} + \frac{1}{2\sigma_c^2}\right) = 0$$

And we get

$$\sigma_c^2 = \frac{\sum_{k=1}^{K}\sum_{i=1}^{n}\omega_{ic}^{(k)}Z_{c,i}^2}{\sum_{k=1}^{K}\sum_{i=1}^{n}\omega_{ic}^{(k)}} \quad c = 1, 2$$

3 Experiment

3.1 Experiment Setting

Following the guidelines, this project firstly generate a synthetic X matrix with $n = 500$ samples and $T = 20$ time points, and in each simulation generate different Y.

Since MCEM performs poor when initial values are far from the true values, I use the following settings:

- Initial $Z \sim \mathcal{N}(0, 0.5)$
- Initial $\beta_1, \beta_2 \sim \mathcal{N}(1, 1)$
- Initial $\sigma_1, \sigma_2 \sim \mathcal{U}(0, 10)$
- Initial $\pi_1 \sim \mathcal{U}(0, 1)$

And a MCEM is regarded as converged if all five parameters change less than 1% in two consecutive iterations.

In M-step, learning rate of β update is set to 0.01, and the maximum iteration is 100. lr decay to 0.001 after 50 iterations.

3.2 Accelerating Precedure

3.2.1 Data Generation

Initially I use dual for-loop to generate data, to fill up a $n \times T$ matrix.

```
    for i in range(n):
        for j in range(T):
            if U[i] == 1:
                # Conditional expectation of group 1
                P_ij = np.exp(beta1 * X[i, j] + Z1[i]) / (1 + np.exp(beta1 * X[i, j] + Z1[i]))
            else:
                # Conditional expectation of group 2
                P_ij = np.exp(beta2 * X[i, j] + Z2[i]) / (1 + np.exp(beta2 * X[i, j] + Z2[i]))

            # Generate Y_ij from Bernoulli distribution
            Y[i, j] = np.random.binomial(1, P_ij)
```

Later on, I use the matrix form, and achieved 10X speedup.

```
    P = np.where(U[:, None] == 1,
        expit(beta1 * X + Z1[:, None]),
        expit(beta2 * X + Z2[:, None]))
    Y = np.random.binomial(1, P)
```

3.2.2 Metropolis-Hastings Algorithm

Sampling during E-step is very time-consuming, since we need to sample Z_i and w_i for n samples and K times. Fortunately, the sampling process is independent for each sample, so we can use **parallel computing** to accelerate the process. I assigned each sample to a different process, using *joblib* package. The average time of E-step is reduced from **22** seconds to **1.3** seconds. *Time data is tested on a M4 Macbook Pro Laptop.*

```
from joblib import Parallel, delayed
def sample_chain(i, Y_i, X_i, Z_i, beta1, beta2, sigma1, sigma2, pi1, K=500, k=100):
    w = 0
    Z = Z_i
    sampled_w = []
    sampled_Z = []

    for iteration in range(K):
        w = sampling_w(X_i, Y_i, Z, beta1, beta2, sigma1, sigma2, pi1)
        Z = sampling_Z(Y_i, X_i, Z, w, beta1, beta2, sigma1, sigma2)

        if iteration >= k:
            sampled_w.append(w)
```

```
            sampled_Z.append(Z)
    return (i, sampled_w, sampled_Z)

def E_step(Y, X, Z, beta1, beta2, sigma1, sigma2, pi1, K=500, k=100, n_jobs=-1):
    n = Y.shape[0]  # Number of samples
    results = Parallel(n_jobs=n_jobs)(
        delayed(sample_chain)(
            i, Y[i], X[i], Z[i], beta1, beta2, sigma1, sigma2, pi1, K, k
        ) for i in range(n)
    )

    all_w = np.zeros((K - k, n), dtype=int)
    all_Z = np.zeros((K - k, n))
    for result in results:
            i, sampled_w, sampled_Z = result
            all_w[:, i] = sampled_w  # 将采样结果放在第 i 列
            all_Z[:, i] = sampled_Z

    return all_w, all_Z
```

3.2.3 Vectorization and numba for β update

Another time-consuming part is the β update in M-step. Initially, I used two for-loops to calculate the gradient of β, then I tried to use vectorization to replace the for-loop, which organize the data in a tensor form: having shape (K, n, T). However, it makes it even slower.

Finally, I use **numba** to accelerate the process. Numba can compile the python code to machine code, which gives python code a similar speed to C code. The following is a table of the time cost of different methods. Timecost is tested on a synthetic dataset with same size as the final project, testing for solve β with 100 loops.

Method	Execution Time (s)
For-loop	5.01
Vectorization	10.10
Numba	0.19

Here is the main part of numba, calculating the gradient of β:

```
from numba import njit, prange
@njit(parallel=True)
def compute_gradients_numba(beta1, beta2, X, Y, all_w, all_Z, K, n, T):
    grad1 = 0.0
    grad2 = 0.0
```

```
    for k in prange(K):
        for i in range(n):
            for t in range(T):
                eta = 0.0
                P = 0.0
                if all_w[k, i] == 1:
                    eta = beta1 * X[i, t] + all_Z[k, i]
                    P = 1.0 / (1.0 + np.exp(-eta))
                    grad1 += (Y[i, t] - P) * X[i, t]
                elif all_w[k, i] == 2:
                    eta = beta2 * X[i, t] + all_Z[k, i]
                    P = 1.0 / (1.0 + np.exp(-eta))
                    grad2 += (Y[i, t] - P) * X[i, t]
    grad1 /= K
    grad2 /= K
    return grad1, grad2
```

3.3 Simulation Results

Although algorithm is accelerated, it still takes a long time to run. In average, each iter of E-step takes 1.3s and M-step takes 0.2s. The whole process of MCEM takes about 1.5 minutes to converge or reach the maximum iteration(100).

The final result is shown as follow.

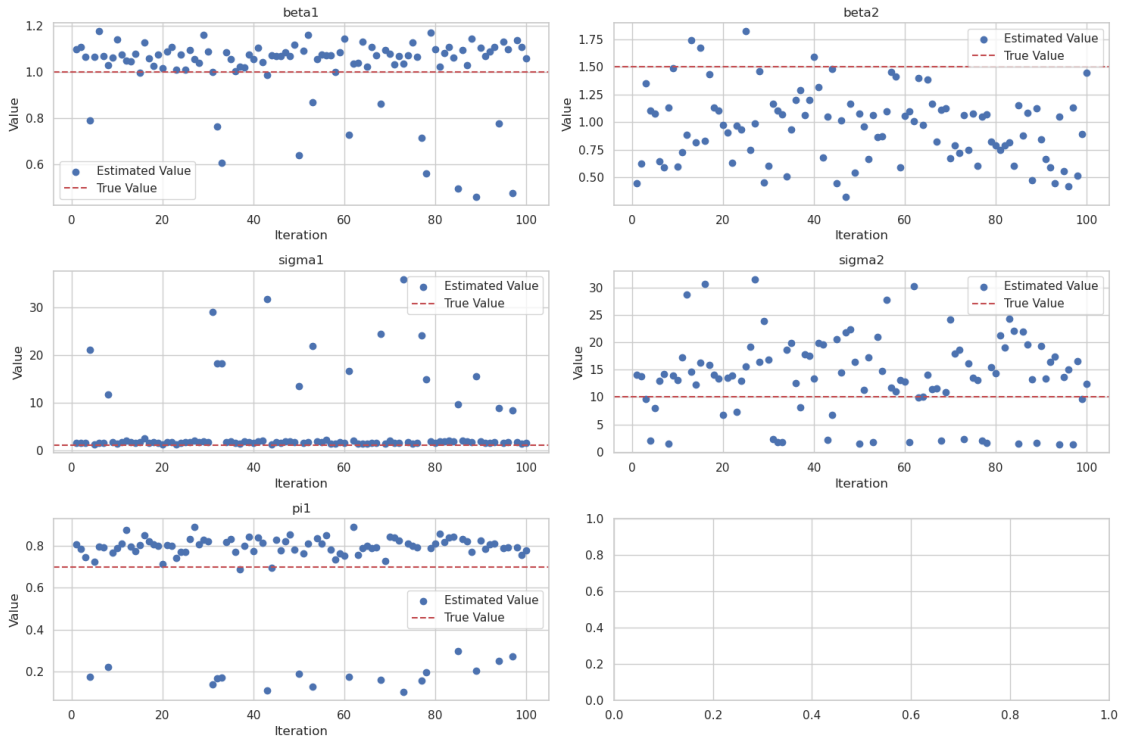

Figure 1: Parameter Estimation of 100 simulations

The results show that the algorithm can estimate the parameters on some extent, but the stability and convergence need further improvement. The estimation of $\sigma 1$ and π_1 is close to the true value and has a small variance, β_1, σ_2 are approximately near the true value, but with a larger variance. The estimation of β_2 is not acceptable.

During the experiment, I also found the convergence of EM algorithm is not stable, although efforts have been made to keep numerical stability (e.g. using log-exp to avoid numerical problems, using clip to avoid extreme values). The estimated parameters may fluctuate in a large range, and σ_1, σ_2 almost monotonic increase.

4 Conclusion

This project conducted Monte Carlo Expectation Maximization(MCEM) algorithm to estimate the latent parameters of a Generalized Linear Mixture Model(GLMM), and achieved some success. It is a beatuiful algorithm, I hope to find the reason of the instability and improve the algorithm in the future.