The mcemGLM package

Felipe Acosta Archila

January 23, 2023

1 A Generalized Linear Mixed Model

Suppose that we observe a vector of data $Y = (Y_1, \ldots, Y_n) \subset \mathcal{Y} \subseteq \mathbb{R}^n$ corresponding to a probability model that depends on a $(p + l)$-dimensional parameter vector θ, a known $n \times p$ fixed effects design matrix X, a known $n \times k$ known random effects design matrix Z, and a k-dimensional vector of unobservable random effects U. Also let $U = (U_1^T, \ldots, U_l^T)^T$, and $Z = (Z_1 \cdots Z_l)$ be decompositions for the vector U and the matrix Z, respectively. We set $\sum_i^l k_i = k$ so that U_i is a k_i-dimensional vector with and that Z_i is a $n \times k_i$ matrix.

Let θ consist of p fixed effects coefficients $\beta = (\beta_1, \ldots, \beta_p)^T$ and l variance parameters, $\sigma^2 = (\sigma_1^2, \ldots, \sigma_l^2)^T$, associated to the random effects $U_1, \ldots, U_l$, i.e. we assume that U_i has a known distribution with variance that depends on the parameter σ_i^2. Our first goal is to find estimators for the $(p + l)$-dimensional parameter θ in a space $\Theta \subset \mathbb{R}^{p+l}$.

We assume that the expected value of Y_i can be written as a linear combination of the observable and unobservable variables through a bijective "link" function g. Let $X^{(i)}$ and $Z^{(i)}$ be the ith rows of the matrices X and Z, and let $\mathrm{E}(Y_i|U = u) = \mu_i$. Then

$$g(\mu_i) = X_i\,\beta + Z^{(i)}\,u, \text{ for } i = 1, \ldots, n.$$

In general let $\mu = (\mu_1, \ldots, \mu_n)$ and let $g(\mu)$ denote the element-wise evaluation of g on the vector

μ, then we can write the mean as

$$g(\mu) = X\,\beta + \sum_{j=1}^{l} Z_j^{(i)}\,u_j. \tag{1}$$

Let $h_U(u)$ be the probability density function of U. We assume that conditional on U, the data is generated from a probability model with probability mass function $f(y|\theta, X, Z, U)$ and that we can write its likelihood function in terms of $\mu = g^{-1}(X\,\beta + \sum_{i=1}^{l} Z_i\,u_i)$, and σ^2. With the model defined this way we can characterize it with the following likelihood functions:

1. A complete data likelihood function:

$$L(\theta|y, u, X, Z) = f(y, u|\theta, X, Z) = f_{Y|U}(y|\theta, X, Z, u)\,h_U(u|\theta). \tag{2}$$

2. And a marginal data likelihood function:

$$L(\theta|y, X, Z) = \int_{\mathbb{R}^k} f(y|\theta, X, Z, U)\,h_U(u|\theta)du. \tag{3}$$

Since the vector U is not observable we need to obtain the parameter estimates from 3. For the rest of the discussion we will drop X and Z from $L(\cdot\,|\,\cdot)$ and $f(\cdot\,|\,\cdot)$ for clearer notation.

The `mcemGLM` package fits models with the following types of data:

1. Bernoulli data. We say that $Y_i \sim \text{Bernoulli}(p_i)$, for $i = 1, \ldots, n$, with $0 < p_i < 1$, if Y_i has probability mass function

$$f(y_i) = p_i^{y_i}(1 - p_i)^{1-y_i}, \text{ for } y_i = 0, 1.$$

   With $\text{E}(Y_i) = p_i$, $\text{Var}(Y_i) = p_i(1 - p_i)$, and $g(p_i) = \log(p_i/(1 - p_i))$.

2. Poisson data. We say that $Y_i \sim \text{Poisson}(\mu_i)$ for $i = 1, \ldots, n$, with $\mu_i > 0$, if Y_i has probability mass function

$$f(y_i) = e^{-\mu_i}\,\frac{\mu_i^{y_i}}{y_i!}, \text{ for } y_i = 0, 1, 2, \ldots$$

   With $\text{E}(Y_i) = \mu_i$, $\text{Var}(Y_i) = \mu_i$, and $g(\mu_i) = \log(\mu_i)$.

3. Negative binomial data. We say that $Y_i \sim$ neg-binom(μ_i, α), for $i = 1, \ldots, n$, with $\mu_i > 0$, and $\alpha > 0$, if Y_i has probability mass function

$$f(y_i) = \frac{\Gamma(y_i + \alpha)}{\Gamma(\alpha)\, y_i!} \left(\frac{\alpha}{\mu_i + \alpha}\right)^{\alpha} \left(\frac{\mu_i}{\mu_i + \alpha}\right)^{y_i}, \text{ for } y_i = 0, 1, 2, \ldots$$

With $\mathrm{E}(Y_i) = \mu_i$, $\mathrm{Var}(Y_i) = \mu_i + \mu_i^2/\alpha$, and $g(\mu_i) = \log(\mu_i)$.

The expectation and variance of Y_i can be found easily by using iterated expectation with respect to a random variable M distributed gamma with shape parameter α, and rate parameter α/μ and setting $Y_i | M = m \sim$ Poisson(m).

By using this definition of the distribution of Y_i we can treat the parameter α as the amount of over-dispersion with respect to the Poisson distribution. The value $\alpha = \infty$ corresponds to no over-dispersion. Notice that in this model, we need to estimate this extra parameter in addition to β and σ^2.

4. Gamma data. We say that Y_i is distributed Gamma with shape parameter α and rate parameter α/μ_i, for $i = 1, \ldots, n$, with $\alpha > 0$, and $\mu_i > 0$, if Y_i has probability density function

$$f(Y_i) = \frac{\left(\frac{\alpha}{\mu_i}\right)^{\alpha}}{\Gamma(\alpha)} y^{\alpha - 1} e^{-\frac{\alpha}{\mu_i} y_i}, \text{ for } y_i > 0.$$

With $\mathrm{E}(Y_i) = \mu_i$, and $\mathrm{Var}(Y_i) = \mu_i^2/\alpha$.

This kind of response can be used to model variables that feature a variance proportional to its squared mean. Similarly to the negative binomial data, α corresponds to an over-dispersion parameter. The case with no over-dispersion, $\alpha = 1$ corresponds to an exponential distribution with rate parameter $1/\mu_i$.

In addition to a distribution for the observed data, we will specify a distribution on the random effects $U_1, \ldots, U_l$. Let I_k be a $k \times k$, $\mathrm{N}_k(a, B)$ a k-dimensional multivariate normal distribution with mean vector a and covariance matrix B, and $t_k(\nu, a, B)$, a k-dimensional multivariate t distribution with ν degrees of freedom, location vector a, and scale matrix B. We will assume that the random effects are normally or t distributed as follows:

1. Set $U_i \sim \mathrm{N}_{k_i}(0, \sigma_i^2 I_{k_i})$ for $i = 1, \ldots, l$, with the U_is mutually independent.

2. Set $U_i \sim t_{k_i}(\nu_i, 0, \sigma_i^2 I_{k_i})$ for $i = 1, \ldots, l$, with the U_is mutually independent.

The package uses an MCEM algorithm. This is a generalization of the EM algorithm and share the same basic idea. We start by assuming two sets of data: An "observed" dataset we call Y and a second set of "missing" data U. In our context the observed data Y are the actual observations we have measured, i.e., the success and failures for the logistic regression, the counts for the Poisson (and negative binomial) regression, and measurements for the gamma regression. The missing data are the unobservable random effects U which we have assumed either to be normally or t distributed.

The EM algorithm estimates the MLEs of a GLMM by an iterative algorithm. Let $\theta^{(t)}$ denote the current estimate at the ith iteration. Let

$$Q(\theta, \theta^{(t)}) = \mathrm{E}\left[\log f(y, u|\theta)|y, \theta^{(t)}\right]. \tag{4}$$

The next value, $\theta^{(t+1)}$, is found by maximizing 4 with respect to θ. The expectation in 4 is taken with respect to $f(u|y, \theta)$. Hence if we want to obtain its closed form we need $f(y, u|\theta)$ and $f_Y(y|\theta)$. The function $f_Y(y|\theta)$ is not available in closed form for the models we are considering, therefore we need to resort to a numerical method to calculate this expectation.

In the models considered in this package we are not be able to calculate the Q function analytically. However since what we are calculating an expectation we can approximate it by using Monte Carlo simulation. The MCEM algorithm, introduced by [1], consists of the following steps.

---

1. Select an initial value $\theta^{(0)}$ for the EM sequence.

2. At step t, obtain a sample $u_{t,1}, \ldots, u_{t,m_t}$, from $U|\,\theta^{(t)}, Y$.

3. Obtain $\theta^{(t+1)}$ by maximizing

$$\widehat{Q}_t(\theta) = \frac{1}{m_t} \sum_{j=1}^{m_t} \log f(y, u_{t,j}|\theta) \tag{5}$$

   with respect to θ.

4. Repeat 2 and 3 until a convergence criterion is reached or a maximum number of iterations has been done.

The key difference between EM and MCEM is that in the expectation step we approximate the Q function using Monte Carlo simulation through the $\hat{Q}$ function defined in 5. This modification turns the integration problem into a sampling problem. Now we are faced with the task of obtaining a sample from the conditional distribution $U|\,\theta^{(t)}, Y$.

To find the MLE of a specified model, the main function of `mcemGLM` package runs through the following steps:

1. Choose $\theta^{(1)}$, the starting value for the EM step. The default method is to fit a model without random effects and use the MLEs of the fixed coefficients as starting values for β. For σ we set a predefined value of 4. In the case of a negative binomial model, MLEs from a Poisson model are used with the over-dispersion parameter set to 100. User-specified initial values are also supported.

2. At step t, obtain the sample $u_{t,1}, \ldots, u_{t,m}$. This is done by using a Metropolis–Hastings algorithm that uses a multivariate normal random variable as its proposal. The standard deviation vector of the proposal distribution is chosen by performing an auto-tuning step before the first iteration. After each iteration the rejection rate of the chain is checked and if it is either too large (> 0.40) or to small (< 0.15) the package performs an auto-tuning step before the next iteration.

3. After obtaining the sample, 5 is maximized with respect to the parameters using the `trust` function from the `trust` package. The maximizers are set as the current value of the estimator of the MLEs.

4. Steps 2 and 3 are repeated until the condition

$$\max_i \left\{ \frac{|\theta_i^{(t)} - \theta_i^{(t-1)}|}{|\theta_i^{(t)}| + \delta} \right\} < \epsilon \tag{6}$$

   for specified values of δ and ϵ is met three consecutive times or a maximum number of iterations have been performed. This is a stopping rule recommended by [2].

   The default values in the package are $\delta = 0.025$ and $\epsilon = 0.02$ but these can be easily changed by the user. The default number of iterations is 40 and this value can also be changed by the user.

5. After terminating the iterative process an additional sample from the conditional distribution of $U|Y$ to estimate Fisher's information matrix.

One condition for convergence of the MCEM algorithm is that $\sum_{t=0}^{\infty} m_t^{-1} < \infty$. However there is no consensus about a best way to approach this. The package starts by choosing a starting Monte Carlo sample size m_1 (with default value $m_1 = 3000$) and this is increased by a multiplicative factor $f > 1$ at each step of the algorithm, i.e. $m_t = f \cdot m_{t-1}$. Common experience is that at the start of algorithm small sample sizes are adequate at the beginning of the algorithm but larger sample sizes are required towards the end. Our approach is to increase f at two times during the algorithm. First after 15 steps we go from 1.025 to 1.2. After another 15 iterations or if we have met condition 6 twice we increase it to 1.5.

One issue that can arise with a fitted model is that it is possible that the Monte Carlo sample size at the last iteration of the algorithm to be inadequate to calculate the observed Fisher's information matrix due to Monte Carlo error [3, 2, 4]. In this case the package will return a warning and will suggest the user to run the algorithm for longer. The package offers functionality to continue the MCEM procedure from an already fitted model.

The last MCMC iteration of the algorithm is saved and returned to help with convergence assessment and can be used to predict the random effects. In addition to the sample from the distribution $U|Y$, $u_1, \ldots, u_{m_T}$, the package also returns the evaluated complete log-likelihood function for each u_i which can also be used to assess convergence for MCMC step.

In addition to MLE estimation, the package also offers Wald tests for model terms, as well as contrast, prediction, and residual estimation. These functions are similar in use to R's built in linear model functionality. Now we turn and look at examples of the use of the package for each type of supported model.

2 Using the mcemGLM package

2.1 Bernoulli model example

```
> require(mcemGLM)
> data("salamander")
```

```
> summary(salamander)
```

```
 Cross       Female          Male           Mate
 RR:90   1      :  6   1      :  6   Min.   :0.000
 RW:90   2      :  6   2      :  6   1st Qu.:0.000
 WR:90   3      :  6   3      :  6   Median :1.000
 WW:90   4      :  6   4      :  6   Mean   :0.525
         5      :  6   5      :  6   3rd Qu.:1.000
         6      :  6   6      :  6   Max.   :1.000
         (Other):324   (Other):324
```

Our first example comes from [5]. The data consists of three experiments, each consisting in salamander mating in two closed groups. Both groups contained 10 males and females each with five species "R" and five species "W". Each experiment resulted in 120 binary observations indicating which matings were successful and which were not.

Let y_{ij} be the indicator of a successful mating between females i and male j for $i, j = 1, \ldots, 60$. Since the salamanders were divided in groups only 360 of these pairs are of interest. Let u_m and u_f be the vectors of random effects for males and females. Each component corresponds to a single salamander, therefore each of these is a 60×1 vector. The conditional mean can be written as

$$\mu_{ij} = \log\left(\frac{p_{ij}}{1 - p_{ij}}\right) = x_{ij}\beta + z_{f,i}\, u_f + z_{m,j}^T\, u_m.$$

Where x_{ij} is a 1×4 row vector indicating the type of cross, $\beta = (\beta_{RR}, \beta_{RW}, \beta_{WR}, \beta_{WW})$, $z_{f,i}^T$ a 60×1 row vector indicating the female involved in the cross, and $z_{m,j}$ a 60×1 row vector indicating the male involved in the cross.

We can fit the model as

```
> fitBernoulli <- mcemGLMM(fixed  = Mate ~ 0+Cross,
+                          random = list(~ 0+Female, ~ 0+Male),
+                          data   = salamander,
+                          family = "bernoulli",
+                          vcDist = "normal")
```

These are the basic arguments to fit model:

- The `fixed` argument specifies the fixed effects. Since we want to estimate the effects for each type of cross we specify that we do not wish to fit an intercept in the model.
- The `random` argument specifies the random effects. In case of more than one random effect these have to be in a list. Each random effect must be specified to not have an intercept.
- The `data` arguments states the name of the data frame that contains the data.
- The `family` argument specifies the type of response we wish to fit.
- The `vcDist` argument specifies the distribution of the random effects.

Given a fitted model we can look at the MLEs, standard errors, and default hypothesis tests with the `summary` command.

```
> summary(fitBernoulli)

Call:
  mcemGLMM(fixed = Mate ~ 0 + Cross, random = list(~0 + Female,
    ~0 + Male), data = salamander, family = "bernoulli", vcDist = "normal")


   Two sided Wald tests for fixed effects coefficients:


          Estimate Std. Error    z value   Pr(>|z|)
CrossRR  1.0199914  0.4192513  2.4328877 0.01497895
CrossRW  0.3247652  0.3973106  0.8174089 0.41369480
CrossWR -1.9538935  0.4806576 -4.0650423 0.00004802
CrossWW  1.0010712  0.4248659  2.3562050 0.01846273



   One sided Wald tests for variance components:


       Estimate Std. Error  z value     Pr(>z)
```

```
Female 1.409038  0.6309231 2.233296 0.01276472
Male   1.259087  0.5867664 2.145807 0.01594420
```

This command displays the original call used to fit the model, and tables of point estimates, standard errors, and Wald tests for the fixed effects and variance components respectively.

We can test multiple contrasts with the `contrasts.mcemGLMM` command. To do so we first set up a contrast matrix. For example if we want to compare all possible pairs of means for each mating groups the matrix is

```
> ctr0 <- matrix(c(1, -1, 0, 0,
+                  1, 0, -1, 0,
+                  1, 0, 0, -1,
+                  0, 1, -1, 0,
+                  0, 1, 0, -1,
+                  0, 0, 1, -1), 6, 4, byrow = TRUE)
> rownames(ctr0) <- c("RR - RW", "RR - WR", "RR - WW",
+                     "RW - WR", "RW - WW", "WR - WW")
>
```

Once we have the contrast matrix we use the `contrasts.mcemGLMM` command. The first argument is the `mcemGLMM` object that contains the model and the second argument the contrast matrix to be tested.

```
> contrasts.mcemGLMM(fitBernoulli, ctr0)

           Estimate Std. Err.         Wald Adj. p-value
RR - RW  0.69522622 0.4815769  2.084112082 8.930325e-01
RR - WR  2.97388491 0.5792061 26.362225357 1.698131e-06
RR - WW  0.01892017 0.5854869  0.001044277 1.000000e+00
RW - WR  2.27865868 0.6245109 13.313077276 1.581369e-03
RW - WW -0.67630605 0.4889340  1.913313109 9.995742e-01
WR - WW -2.95496474 0.5877126 25.279854177 2.975165e-06
```

The table returns the contrast estimates, standard errors and Bonferroni adjusted p-values.

We have access to the residuals with the `residuals` command. The arguments needed are the `mcemGLMM` object that contains the model and `type`, a string that specifies the type of residual, i.e., deviance residuals (default value, `type = 'deviance'`) or Pearson (`type = 'pearson'`) residuals. 2.1 shows the deviance residuals for each observation and grouped by cross type.

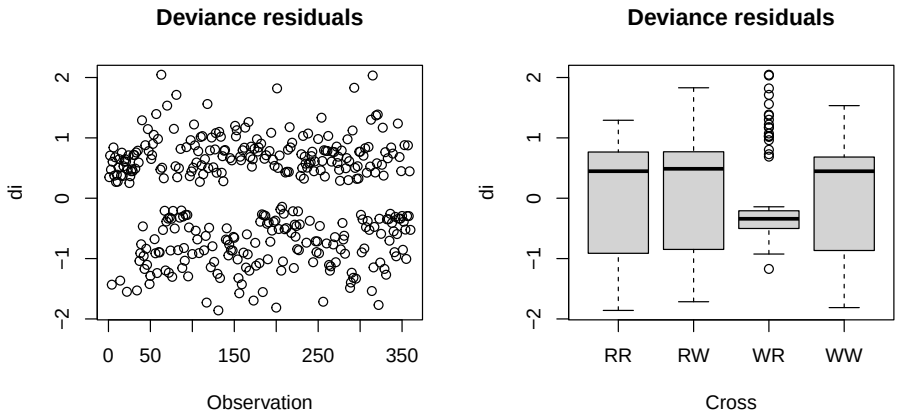

Figure 1: Left: Deviance residuals for each observation. Right: Deviance residuals grouped by cross type.

We can obtain mean predictions at the population level with the `predict` command. This command can take three arguments, the first argument is the `mcemGLMM` object that contains the model. The second argument, `newdata` is a list with vectors corresponding to the values of fixed effects where the predictions will be evaluated. If this argument is not provided the function will return a prediction for each observation in the model. The third argument `type` is a string that specifies the type of prediction to be returned, i.e., a link function evaluation (`type = 'link'`) or a prediction on the response mean (`type = 'response'`).

```
> predict(fitBernoulli, newdata=list(Cross = c("RR", "RW", "WR", "WW")),
+         type = "link", se.fit = TRUE)

       Estimate        SE
[1,]  1.0199914 0.1757717
[2,]  0.3247652 0.1578557
[3,] -1.9538935 0.2310317
[4,]  1.0010712 0.1805111
```

```
> predict(fitBernoulli, newdata=list(Cross = c("RR", "RW", "WR", "WW")),
+         type = "response", se.fit = TRUE)

      Estimate          SE
[1,] 0.7349709 0.006669239
[2,] 0.5804851 0.009361322
[3,] 0.1241294 0.002730870
[4,] 0.7312691 0.006970973
```

The "link" and "response" predictions correspond to the value of the link function and the probability of success respectively for the value of the random effects when the random effects are set equal to zero. If the argument `se.fit` is set to `TRUE`, the function will also return standard errors for the mean predictions.

We can obtain random effect predictions `ranef.mcemGLMM` command. The output of this command is not shown for space concerns but 2.1 shows Q-Q plots for the two variance components.

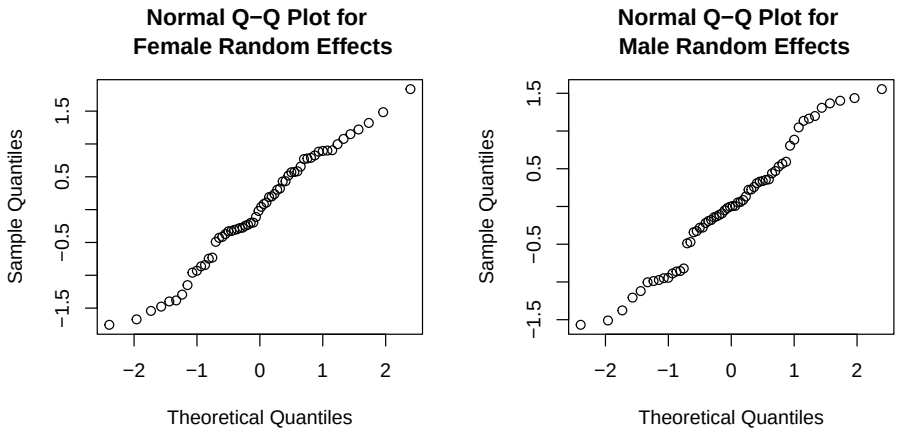

Figure 2: Left: Q-Q plot for Female random effects. Right: Q-Q plot for Male random effects.

We can assess convergence of the MCEM iterations by looking at trace plots of the EM sequence estimators. The field `mcemEST` contains a matrix with the MLE estimates at each iteration of the MCEM algorithm. 2.1 shows trace plots for all the parameters estimated in the model.

We can also obtain a trace plot for the value of $\hat{Q}$ function at each iteration, the field `QfunVal` contains these values. 2.1 shows a trace plot for the values of this estimate across the EM iterations.

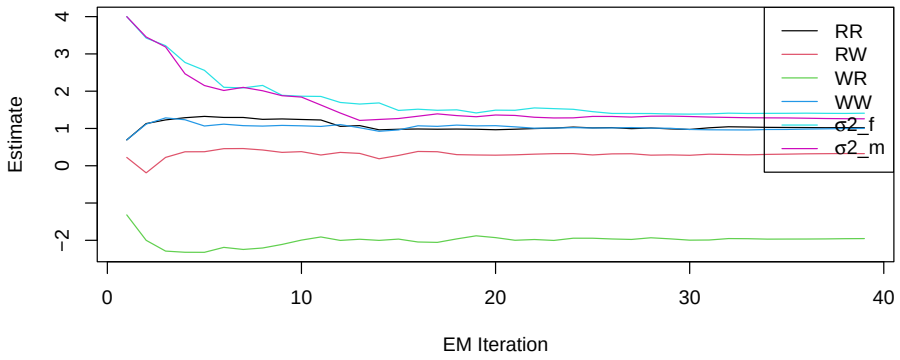

Figure 3: Trace plots for the EM sequences of each parameter.

To assess convergence at the MCMC level we can use the sample obtained at the last EM iteration of $U|Y$. In this problem we have 60 different chains so it is not feasible to look at trace plots of all of them. However it is recommended to the user to look at trace plots and autocorrelation functions of at least a handful of them. In addition to the sample from $U|Y$, we can find the complete log-likelihood function evaluated at each observation of the $U|Y$ chain in the field `QfunMCMC`. This can also be useful for convergence assessment since the maximization step of the algorithm is done on this function. 2.1 shows these assessment plots for the first female subject and the $\hat{Q}$ function.

2.2 Negative binomial model example

The second data set by [6], comes from an experiment with $i = 1, \ldots, 59$ epilepsy patients. Each of the patients was assigned to a control group or a treatment group. The experiment recorded the number of seizures experienced by each patient over four two-week periods. The experiment also recorded a baseline count of the number of seizures the patients had experienced during the previous eight weeks. In their analysis they used the following covariates.

- Base: The logarithm of baseline/4.
- Age: The logarithm of the patient's age in years.
- Trt: An indicator variable for the treatment group.

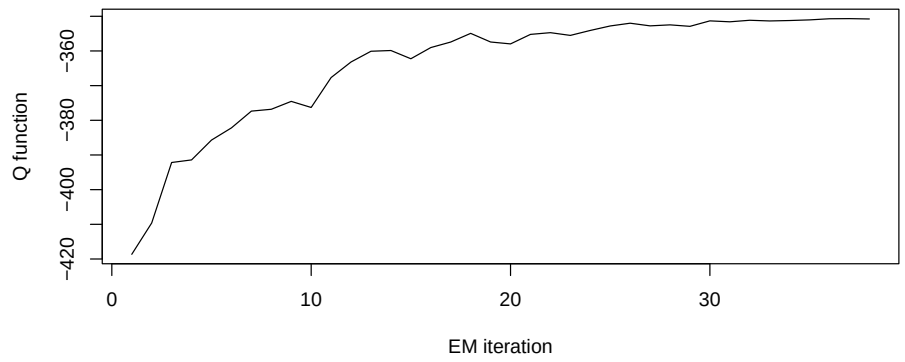

Figure 4: Trace plot for the $\hat{Q}$ function value at each EM iteration.

- V4: An indicator variable for the fourth time period.

This data is also analyzed in [7]. One of their models replaces the variable V4 with the variable Visit, defined as $(j - 2.5)/5$ for $j = 1, 2, 3, 4$ where each value corresponds to one of the four time periods. The analysis of [6] consists in a multiplicative model, while the analysis of [7] consists a Poisson loglinear model. [8] note that the Poisson loglinear model fails to account for over-dispersion and consider the use of a negative binomial model. This last model is the one we fit here in this example.

Let y_{ij} be the count for the ith subject at the jth period. Then we can write the model as

$$\log \mu_{ij} = \beta_0 + \beta_1 \text{Base} + \beta_2 \text{Trt} + \beta_3 \text{Base} \times \text{Trt} + \beta_4 \text{Age} + \beta_5 \text{Visit} + u_i.$$

We will fit this model using t distributed random effects.

```
> fitNegbin <- mcemGLMM(fixed  = count ~ base * group + age + visit,
+                   random = list( ~ 0 + id),
+                   data   = epilepsy,
+                   family = "negbinom",
+                   vcDist = "t",
+                   df     = 10)
```

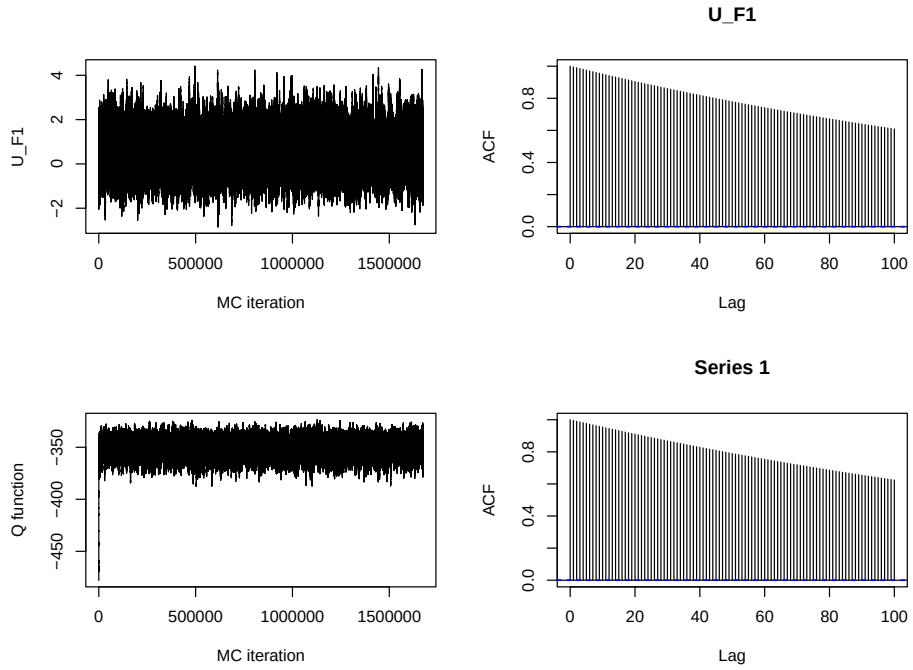

Figure 5: Top: Trace plot and autocorrelation function for the first female subject. Bottom: Trace plot and autocorrelation function for the $\hat{Q}$ function function.

The argument `df` specifies the degrees of freedom for the t distribution. The summary of the model is

```
> summary(fitNegbin)

Call:
  mcemGLMM(fixed = count ~ base * group + age + visit, random = list(~0 +
    id), data = epilepsy, family = "negbinom", vcDist = "t",
    df = 10)


   Two sided Wald tests for fixed effects coefficients:


              Estimate Std. Error    z value   Pr(>|z|)
(Intercept) -0.9920524  1.1068225 -0.8963066 0.37008903
base         0.8950982  0.1342273  6.6685284 0.00000000
group       -0.8754207  0.4003589 -2.1865897 0.02877249
```

```
age           0.3690062  0.3200139  1.1530942 0.24887171
visit        -0.2578519  0.1668548 -1.5453667 0.12225760
base:group    0.3016896  0.2065637  1.4605162 0.14414825

   Overdispersion parameter alpha:

      Estimate Std. Error
alpha 7.385486   1.795523



   One sided Wald tests for variance components:

    Estimate Std. Error  z value   Pr(>z)
id 0.2190529  0.1140393 1.920855 0.027375
```

3 Options for the mcemGLM package

The help file of the `mcemGLMM` function lists other possible options that can be passed through the `controlEM` argument. The complete list of options is:

EMit: Maximum number of EM iterations.

MCit: Initial number of Monte Carlo iterations for the MCMC step.

MCf: Factor in which the MC iterations increase in each EM iteration.

verb: Logical value. If set to TRUE, at each EM iteration the function will print convergence information and a trace plot for one of the random effects. This can be useful to assess the performance and tuning of the algorithm but it can impact the actual running time.

MCsd: Initial standard deviation for the proposal density of the MCMC step. If zero (default) an auto-tuning step will be performed.

EMdelta: constant for the EM error assessment, see 6.

EMepsilon: constant for the EM error assessment, see 6.

The object returned by the `mcemGLMM` function has the following fields:

mcemEST: A matrix with the value of the maximum likelihood estimators at the end of each EM step.

iMatrix: Fisher's information matrix.

QfunVal The of the Q function (up to a constant.)

QfunMCMC: The Q function evaluated at a sample from the distribution of $U|Y, \hat{\theta}_n$.

randeff: A sample from the distribution of $U|Y, \hat{\theta}_n$.

y: The vector of observations.

x: The design matrix for the fixed effects.

z: The design matrix for the random effects.

EMerror The relative error at the last iteration, see 6.

MCsd: The last value for the standard deviation of the proposal distribution of the MCMC step.

call: The original call used to fit the function.

References

[1] Greg C. G. Wei and Martin A. Tanner. "A Monte Carlo Implementation of the EM Algorithm and the Poor Man's Data Augmentation Algorithms". In: 85 (1990), pp. 699–704.

[2] James G. Booth and James P. Hobert. "Maximizing generalized linear mixed model likelihoods with an automated Monte Carlo EM algorithm". In: 61 (1999), pp. 265–285.

[3] Brian S. Caffo, Wolfgang Jank, and Galin L. Jones. "Ascent-Based Monte Carlo EM". In: *Journal of the Royal Statistical Society,* Series B 67 (2005), pp. 235–251.

[4] Ralitza V. Gueorguieva and Alan Agresti. "A Correlated Probit Model for Joint Modeling of Clustered Binary and Continuous Responses". In: 96 (455 2001), pp. 1102–1112.

[5] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. second. London: Chapman & Hall, 1989, p. 500.

[6] Peter F. Thall and Stephen C. Vail. "Some covariance models for longitudinal count data with overdispersion". In: *Biometrics* 46 (3 1990), pp. 657–671.

[7] Norman E. Breslow and D. G. Clayton. "Approximate Inference in Generalized Linear Mixed Models". In: 88 (1993), pp. 9–25.

[8] James G. Booth et al. "Negative binomial loglinear mixed models". In: *Statistical Modeling* 3 (2003), pp. 179–191.