Quantitative Genomics and Genetics 2016
Computer Lab 10

Jin Hyun Ju (jj328@cornell.edu)

April 28, 2016

1 Linear Mixed Models

Linear mixed models are named so because unlike linear models with only fixed effects, they use a "mix" of fixed effects and random effects. To directly compare the difference between the two models, let's review the linear model in its simplest form we have used to model genotype effects on the phenotype.

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\epsilon}$$
$$\vec{\epsilon} \sim N(\vec{0}, \sigma_e^2\mathbf{I})$$

$\vec{y}$ is a vector with n measurements of the phenotype, $\mathbf{X}$ is the $n \times j$ matrix with $j-1$ indenpendent variables and one column with 1s for the mean, $\vec{\beta}$ is an j dimensional vector of the beta values, and $\vec{\epsilon}$ is the normally distributed error with a variance of σ_e^2. We can also represent the model as a multivariate normal distribution as:

$$\vec{y_i} \sim N(\mathbf{X}\vec{\beta}, \sigma_e^2\mathbf{I})$$

where the mean of the phenotypes depend on $\mathbf{X}\vec{\beta}$ and the error has no structure. In this case, we had to only worry about the likelihood of $\vec{\beta}$ and σ_e^2 given the data. This had a nice closed form solution to calculate the maximum likelihood esitmators for the β values which were the center of attention and the variance σ_e^2 which we did not focus on that much.

In comparison, linear mixed models have additional random effect terms in the model. The most commonly used form in genomics will look something like this:

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{a} + \vec{\epsilon}$$
$$\vec{a} \sim N(\vec{0}, \sigma_a^2\mathbf{A})$$
$$\vec{\epsilon} \sim N(\vec{0}, \sigma_e^2\mathbf{I})$$

where $\vec{a}$ is a n dimensional vector representing the random effect drawn from a multivariate normal distribution with a variance structure defined by the $n \times n$ covariance matrix $\mathbf{A}$. In a multivariate normal distribution form the model can be shown as:

$$\vec{y} \sim N(\mathbf{X}\vec{\beta}, \sigma_a^2\mathbf{A} + \sigma_e^2\mathbf{I})$$

As you can see the key difference between the linear model and the linear mixed model lies in the modeling of the variance. Intuitively, $\mathbf{A}$ represents the pairwise similarity between the n samples, which has an affect on the residual variance. Simply put, the random effect is included to account for samples that are not independent.

Now that we have three parameters to estimate $(\vec{\beta}, \sigma_a^2, \sigma_e^2)$ there is no closed form solution anymore. This week we will learn how to get the maximum likelihood estimators for those parameters using an EM algorithm

2 EM algorithm

Roughly speaking, the process of the EM algorithm can be outlined in two steps. During the first step it fixes the parameters (betas, and sigmas) to calculate the best estimate for `alpha, V` in the code, which are used in the next step to update the betas and the sigmas until the likelihood does not change much.

The technical aspect of the EM algorithm is beyond the scope of this course, so in this exercise we are going to focus on understanding the concept, inputs and outputs.

2.1 Exercise

- Using the data files for this lab, inspect the input. Try to visualize the given A matrix. What does the structure shown in the A matrix imply?
- To check that our algorithm actually works, modify the function to print out the iteration step and the corresponding log likelihood. Does the log-likelihood increase after each step? Generate a plot that shows the log likelihood for each iteration.

```
library(MASS)  # load MASS package to use the ginv() function

X = as.matrix(read.table("QG16_Lab11_EM_X.txt"))
Y = as.matrix(read.table("QG16_Lab11_EM_Y.txt"))
A = as.matrix(read.table("QG16_Lab11_EM_A.txt"))



EM_algorithm = function(Y, X_j, A, max.iter = 100) {

    # Calculate the inverse of A once since it is used repeatedly in the
```

```
# algorithm
solve_A = ginv(A)

n = length(Y)

I = diag(1, n)

log_L = c()
# set starting values
sigma_sq_a = 70
sigma_sq_e = 10
beta = as.vector(rep(0, ncol(X_j)))


C = A * sigma_sq_a + I * sigma_sq_e
log_L[1] = -1/2 * determinant(C)$modulus - 1/2 * t(Y - X_j %*% beta) %*%
    ginv(C) %*% (Y - X_j %*% beta)
iter = 2

while (iter < max.iter) {

    # calculate S, alpha and V using the previous beta, sigma_a, sigma_e values
    S = ginv(I + solve_A * sigma_sq_e/sigma_sq_a)

    alpha = S %*% (Y - X_j %*% beta)

    V = S * sigma_sq_e

    # update beta, sigma_a, sigma_e
    beta = ginv(t(X_j) %*% X_j) %*% t(X_j) %*% (Y - alpha)

    sigma_sq_a = as.numeric(1/n * (t(alpha) %*% solve_A %*% alpha + sum(diag(solve_A %*%
        V))))

    sigma_sq_e = as.numeric(1/n * (t(Y - X_j %*% beta - alpha) %*% (Y -
        X_j %*% beta - alpha) + sum(diag(V))))


    C = A * sigma_sq_a + I * sigma_sq_e
    log_L[iter] = -1/2 * determinant(C)$modulus - 1/2 * t(Y - X_j %*% beta) %*%
        ginv(C) %*% (Y - X_j %*% beta)

    if (log_L[iter] - log_L[iter - 1] < 1e-05) {
        break
    }
```

```
        iter = iter + 1
    }

    return(list(beta = beta, sigma_sq_a = sigma_sq_a, sigma_sq_e = sigma_sq_e,
        log_L = log_L[iter - 1]))
}



############### Mixed model #

n_indivs = length(Y)

# Null model
One = as.matrix(rep(1, n_indivs))
log_L_null = EM_algorithm(Y, One, A)$log_L

p_values_EM = c()

# Full model
for (j in 1:ncol(X)) {
    X_j = cbind(1, X[, j])

    fit = EM_algorithm(Y, X_j, A)

    p_values_EM[j] = pchisq(-2 * (log_L_null - fit$log_L), 1, lower.tail = FALSE)

    cat(".")
}
## ..................................................................................
```