Chapter 1

Linear Mixed Model Implementation

The LMM implementation is available as C source code. It depends on the standard C libraries CHOLMOD (Davis, 2008) and GSL (Galassi et al., 2009). It can be found at `http://www.mcw.edu/biostatistics/Research/Software`

1.1 Data Preparation

A program has been provided for typical data preparation: `onetime.c` . It depends on two header files. The first `gsl_cholmod.h` defines two C preprocessor (`cpp`) macros that allow GSL to read and write to CHOLMOD matrices present in memory. `MAT(A, B)` is used for matrices and `VEC(A, B)` is used for vectors. They both take two parameters.

`A`: A pointer to an already existing matrix created by CHOLMOD.

`B`: The name of a GSL object to create. No new memory will be allocated since `A` already has memory allocated to it. Therefore, you will not need to "free" the memory associated with `B`; that will be taken care of when you "free" `A` instead. The new `B` object will need

Table 1.1: The definitions of the values found in `onetime.h` .

Variable	Definition
`p`	the dimension of the fixed parameters $\boldsymbol{\beta}$
`q1`	the number of primary clusters (hospitals)
`q2`	the number of secondary clusters (surgeons)
`q`	the number of total clusters
`N`	the number of subjects (patients)

to be operated on as a memory location. For example, if the second parameter is `X`, then you will need to access `&X` .

It is important to note that CHOLMOD has a column-major definition of dense matrices: adjacent memory locations represent two adjacent cells in a column. GSL has a row-major definition of dense matrices: adjacent memory locations represent two adjacent cells in a row. These differing definitions are easily handled; a column-major matrix is the same as the transpose of a row-major matrix.

The second header file is `onetime.h` . You need to create this header file based on your data. For example, for the XS Scenario, the file looks like Figure 1.1.

```
const int p=4, q1=5, q2=25, q=q1+q2, N=2500;
```

Figure 1.1: An example of `onetime.h` based on the XS Scenario.

See Table 1.1 for the definition of the variable names in `onetime.h`.

The program will read and write several input files in the Matrix Market format (Boisvert et al., 1996); see Table 1.2.

After compiling `onetime.c` into `onetime.out` (or whatever you are calling your executable), then running `onetime.out` will produce the output files and it will print $\boldsymbol{y}'\boldsymbol{y}$ on standard output (`stdout`).

Table 1.2: List of input/output files for `onetime.c` and their definitions.

Filename:	content definition
`Y.mtx`	the input file for the outcome vector $\boldsymbol{y}$
`uX.mtx`	the input file for the “uncentered” covariates $\boldsymbol{X}$
`X.mtx`	the output file for the “centered” covariates $\boldsymbol{X}$
`XtX.mtx`	the output file for $\boldsymbol{X}'\boldsymbol{X}$ where $\boldsymbol{X}$ has been “centered”
`XtY.mtx`	the output file for $\boldsymbol{X}'\boldsymbol{y}$ where $\boldsymbol{X}$ has been “centered”
`uZ.mtx`	the input file for the “unordered” $\boldsymbol{Z}$
`ZtZ.mtx`	the output file for $\tilde{\boldsymbol{Z}}'\tilde{\boldsymbol{Z}}$
`ZtX.mtx`	the output file for $\tilde{\boldsymbol{Z}}'\boldsymbol{X}$ where $\boldsymbol{X}$ has been “centered”
`ZtY.mtx`	the output file for $\tilde{\boldsymbol{Z}}'\boldsymbol{y}$
`P.mtx`	the output file for the permutation $\boldsymbol{P}$
`u1mask.mtx`	An output file for a vector of 1s and 0s. The 1s represent the locations of re-ordered hospitals in $\tilde{\boldsymbol{u}}$; the 0s, re-ordered surgeons.
`D1.mtx`	A similar definition to `u1mask.mtx` . It is the hospital portion of $\tilde{\boldsymbol{D}}$ with the surgeon portion zeroed out.
`D2.mtx`	A similar definition to `D1.mtx` . It is the surgeon portion of $\tilde{\boldsymbol{D}}$ with the hospital portion zeroed out.

1.2 Conjugate Priors

The version of the program created for conjugate priors is `Normal.c` . It depends on two header files. The first `gsl_cholmod.h` has already been discussed. The second header file is `Normal.h` . You need to create this header file based on your data, your prior parameters and your initial values. For example, for the XS Scenario, the file looks like Figure 1.2.

```
const int p=4, q1=5, q2=25, q=q1+q2, N=2500, M=20000;
const double a1=(q1+0.1)/2., b1=0.1,
             a2=(q2+0.1)/2., b2=0.1,
             ae=(N+0.1)/2.,  be=0.1,
             am=0.,          bm=0.001,
             yty=43137.609891;
double mu=0., taue=1., tau1=1., tau2=1.;
gsl_vector *beta=gsl_vector_calloc(p); // initialized to zeros
```

Figure 1.2: An example of `Normal.h` based on the XS Scenario.

See Table 1.3 for the definition of the variable names in `Normal.h`.

The program will read several input files in the Matrix Market format (Boisvert et al., 1996); see Table 1.4.

After compiling `Normal.c` into `Normal.out` (or whatever you are calling your executable), then running `Normal.out` produces an R source file `Normal.R` containing the Gibbs samples for $\boldsymbol{\beta}$, μ, τ_1, τ_2 and τ_ϵ (in that order). `Normal.R` is the default name for the output unless you pass a file name as an argument such as `Normal.out example.R`

Table 1.3: The definitions of the values found in `Normal.h` .

Variable	Definition
`p`	the dimension of the fixed parameters $\boldsymbol{\beta}$
`q1`	the number of primary clusters (hospitals)
`q2`	the number of secondary clusters (surgeons)
`q`	the number of total clusters
`N`	the number of subjects (patients)
`M`	the number of Gibbs samples to perform
`a1`	the 1st posterior parameter to the Gamma distribution for τ_1
`b1`	the 2nd prior parameter to the Gamma distribution for τ_1
`a2`	the 1st posterior parameter to the Gamma distribution for τ_2
`b2`	the 2nd prior parameter to the Gamma distribution for τ_2
`ae`	the 1st posterior parameter to the Gamma distribution for τ_ϵ
`be`	the 2nd prior parameter to the Gamma distribution for τ_ϵ
`am`	the prior mean for μ
`bm`	the prior precision for μ
`yty`	$\boldsymbol{y}'\boldsymbol{y}$
`mu`	the initial value of μ
`taue`	the initial value of τ_ϵ
`tau1`	the initial value of τ_1
`tau2`	the initial value of τ_2
`beta`	the initial value of $\boldsymbol{\beta}$

Table 1.4: List of input files for `Normal.c` and their definitions.

Filename:	content definition
`C.mtx`	the prior precision of $\boldsymbol{\beta}$
`Cc.mtx`	the prior mean of $\boldsymbol{\beta}$
`XtX.mtx`	$\boldsymbol{X}'\boldsymbol{X}$ where $\boldsymbol{X}$ has been "centered"
`XtY.mtx`	$\boldsymbol{X}'\boldsymbol{Y}$ where $\boldsymbol{X}$ has been "centered"
`ZtZ.mtx`	$\tilde{\boldsymbol{Z}}'\tilde{\boldsymbol{Z}}$
`ZtX.mtx`	$\tilde{\boldsymbol{Z}}'\boldsymbol{X}$ where $\boldsymbol{X}$ has been "centered"
`ZtY.mtx`	$\tilde{\boldsymbol{Z}}'\boldsymbol{Y}$
`u1mask.mtx`	A vector of 1s and 0s. The 1s represent the locations of re-ordered hospitals in $\tilde{\boldsymbol{u}}$; the 0s, re-ordered surgeons.
`D1.mtx`	A similar definition to `u1mask.mtx` . It is the hospital portion of $\tilde{\boldsymbol{D}}$ with the surgeon portion zeroed out.
`D2.mtx`	A similar definition to `D1.mtx` . It is the surgeon portion of $\tilde{\boldsymbol{D}}$ with the hospital portion zeroed out.

1.3 Noninformative Prior

The Noninformative prior version of the program, `Normal-Uniform.c`, is very similar to the conjugate prior version. The header file is now named `Normal-Uniform.h` . The definition of the parameters is the same except for those associated with τ_1 and τ_2: `a1`, `b1`, `a2` and `b2`. For example, for the XS Scenario, the file looks like Figure 1.3.

```
const int p=4, q1=5, q2=25, q=q1+q2, N=2500, M=20000;
const double a1=(q1-1)/2.,  b1=0.01,
             a2=(q2-1)/2.,  b2=0.01,
             ae=(N+0.1)/2., be=0.1,
             am=0.,         bm=0.001,
             yty=43137.609891;
double mu=0., taue=1., tau1=1., tau2=1.;
gsl_vector *beta=gsl_vector_calloc(p); // initialized to zeros
```

Figure 1.3: An example of `Normal-Uniform.h` based on the XS Scenario.

After compiling `Normal-Uniform.c` into `Normal-Uniform.out` (or whatever you are calling your executable), then running `Normal-Uniform.out` produces an R source file `Normal-Uniform.R` containing the Gibbs samples for $\boldsymbol{\beta}$, μ, τ_1, τ_2 and τ_ϵ (in that order). `Normal-Uniform.R` is the default name for the output unless you pass a file name as an argument such as `Normal-Uniform.out example.R`

Chapter 2

Logistic Mixed Model Implementation

The Logistic Mixed Model implementation is available as C source code. It depends on the standard C libraries CHOLMOD (Davis, 2008) and GSL (Galassi et al., 2009). It can be found at `http://www.mcw.edu/biostatistics/Research/Software`

Although, there are fewer one-time calculations for the Logistic Mixed Model, you can still use the data preparation program previously discussed: `onetime.c` . The only difference is that you need to copy the file `V.mtx` to the non-existent file `Y.mtx` and the program will perform the necessary operations.

2.1 Conjugate Priors

The Logistic Mixed Model program, `Logistic.c`, is very similar to the LMM version. The header file is `Logistic.h` . The header file is nearly identical to `Normal.h`; the exceptions are that the parameters `ae`, `be`, `yty` and `taue` are unnecessary. For example, for the XS Scenario the file looks like Figure 2.1.

The program will read several input files in the Matrix Market format (Boisvert et al., 1996); see Table 2.1.

```
const int p=4, q1=5, q2=25, q=q1+q2, N=2500, M=20000;
const double a1=(q1+0.1)/2., b1=0.1,
             a2=(q2+0.1)/2., b2=0.1,
             am=0.,          bm=0.001;
double mu=0., tau1=1., tau2=1.;
gsl_vector *beta=gsl_vector_calloc(p); // initialized to zeros
```

Figure 2.1: An example of `Logistic.h` based on the XS Scenario.

Table 2.1: List of input files for `Logistic.c` and their definitions.

Filename:	content definition
`C.mtx`	the prior precision of $\boldsymbol{\beta}$
`Cc.mtx`	the prior mean of $\boldsymbol{\beta}$
`X.mtx`	$\boldsymbol{X}$ which has been “centered”
`V.mtx`	$\boldsymbol{V}$ which is the dichotomous outcome
`Z.mtx`	$\tilde{\boldsymbol{Z}}$
`ZtZ.mtx`	$\tilde{\boldsymbol{Z}}'\tilde{\boldsymbol{Z}}$
`u1mask.mtx`	A vector of 1s and 0s. The 1s represent the locations of re-ordered hospitals in $\tilde{\boldsymbol{u}}$; the 0s, re-ordered surgeons.
`D1.mtx`	A similar definition to `u1mask.mtx` . It is the hospital portion of $\tilde{\boldsymbol{D}}$ with the surgeon portion zeroed out.
`D2.mtx`	A similar definition to `D1.mtx` . It is the surgeon portion of $\tilde{\boldsymbol{D}}$ with the hospital portion zeroed out.

After compiling `Logistic.c` into `Logistic.out` (or whatever you are calling your executable), then running `Logistic.out` produces an R source file `Logistic.R` containing the Gibbs samples for $\boldsymbol{\beta}$, μ, τ_1 and τ_2 (in that order). `Logistic.R` is the default name for the output unless you pass a file name as an argument such as `Logistic.out example.R`

2.2 Noninformative Prior

The Noninformative prior version of the program, `Logistic-Uniform.c`, is very similar to the conjugate prior version. The header file is now named `Logistic-Uniform.h` . The definition of the parameters is the same except for those associated with τ_1 and τ_2: `a1`, `b1`, `a2` and `b2`. For example, for the XS Scenario, the file looks like Figure 2.2.

```
const int p=4, q1=5, q2=25, q=q1+q2, N=2500, M=20000;
const double a1=(q1-1.)/2., b1=0.25,
             a2=(q2-1.)/2., b2=0.25,
             am=0.,         bm=0.1;
double mu=0., tau1=1., tau2=1.;
gsl_vector *beta=gsl_vector_calloc(p); // initialized to zeros
```

Figure 2.2: An example of `Logistic-Uniform.h` based on the XS Scenario.

After compiling `Logistic-Uniform.c` into `Logistic-Uniform.out` (or whatever you are calling your executable), then running `Logistic-Uniform.out` produces an R source file `Logistic-Uniform.R` containing the Gibbs samples for $\boldsymbol{\beta}$, μ, τ_1 and τ_2 (in that order). `Logistic-Uniform.R` is the default name for the output unless you pass a file name as an argument such as `Logistic-Uniform.out example.R`

Bibliography

Boisvert, R., R. Pozo, and K. Remington (1996). The Matrix Market exchange formats: initial design. Technical Report NISTIR 5935, National Institute of Standards and Technology. [http://math.nist.gov/MatrixMarket].

Davis, T. (2008). *User guide for CHOLMOD: a sparse Cholesky factorization and modification package.* Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL. [http://www.cise.ufl.edu/research/sparse/cholmod].

Galassi, M., J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi (2009). *GNU Scientific Library Reference Manual* (3rd ed.). Network Theory Ltd. [http://www.gnu.org/software/gsl].