Building General Random Effect Models with R and ADMB

Jeff Laake, Hans Skaug and Devin Johnson

January 23, 2013

The ADMB and Otter Research web sites provide examples of specific random effect models. See http://admb-project.org/examples and http://www.admb-project.org/examples/miscellaneous/otter-reseach-collection. However, none of these examples discuss how to build more general TPL files to accommodate a class of random effect models. The R package glmmadmb (http://glmmadmb.r-forge.r-project.org/) is an example where R is used with a single TPL file that can fit generalized linear mixed models for a wide range of distributions. By examining the R code and TPL file for glmmadmb, you can work out how to develop a general TPL file and use R code to build the structures ADMB needs to fit mixed effect models. However, glmmadmb is fairly complex and sorting through the code can be challenging. Here we provide a simpler example with comments to demonstrate the process so others can incorporate this into their model building software.

We use the general modeling tools in R to create a DAT file that contains the structures needed for the TPL file for ADMB. We use the R2admb package as the R and ADMB interface. R and R2admb could be replaced with equivalent software that builds the DAT file but R is freely available and has a wealth of modeling and data manipulation tools. Using mixed logistic regression as an example, we'll begin by describing the TPL file structure to fit a random effects model for a user-specified dataframe and formula.

ADMB Code

We will describe the contents of the file mixed.tpl given in the Appendix. This simple skeleton file could be modified to compute a different likelihood or could be extended to incorporate mixed-effects modeling for several parameters contained in a single likelihood. The TPL file is generalized to work with any data file or formula by specifying the model via design matrices that are passed as data. For mixed effect models, there is a design matrix for the fixed effect and one for the random effects. In addition, there is a vector of indices for the random effects that are passed as data as well.

We'll describe the DAT structure with a simple illustrative example that is too small to use in practice. The variable y is the response and id, sex and time are all factor variables.

y	id	sex	time
0	1	F	1
1	1	F	2
0	2	M	1
1	2	M	2
1	3	F	1
0	3	F	2

Consider building a logistic regression model for the response y in which sex is a fixed effect and id (individual heterogeneity) is a random intercept for the response. We begin by describing the DATA_SECTION of the TPL file and show the contents below of the DAT file for this example. The variables in parentheses are the names used in the TPL file (Appendix) and we have used underscore to show the range of the vector or matrix (e.g. y_i, i=1 to n):

number of rows in data (n)
6
vector of 6 responses (y_i, i=1 to n)
0 1 0 1 1 0
number of columns in the fixed effect design matrix (kfixed)
2
fixed effect design matrix (fixedDM_ij; i=1 to n; j=1 to kfixed) (first column is intercept; second column is male effect)
1 0
1 0
1 1
1 1
1 0
1 0
number of random effect values (nre) (for this model, one for each id)
3
number of columns in the random effect design matrix (krand)
1
random effect design matrix (randDM_ij; i=1 to n; j=1 to krand)
1
1
1
1
1
1
random effect indices (randIndex_ij; i=1 to n; j=1 to krand)(values from 1 to nre)
1
1
2
2
3
3

There is a single random effect variable with nre=3 values (ie. one for each of the 3 individuals). The indices specify which random effect is used for the value in the random design matrix. For this model there are 3 parameters ($kfixed(2) + krand(1)$): Intercept and Male fixed effects and sigma (standard deviation) for id random effect. The likelihood is:

$$L(\beta_1,\beta_2,\tau_1,\boldsymbol{u}|\,\boldsymbol{y}) = \prod_{i=1}^{6} p_i^{y_i}(1-p_i)^{(1-y_i)}$$

Below we specify the value of each p_i (as logit) in terms of the parameters and the 3 random effect values u_1, u_2, u_3 which are standard normal random variables. The variable mu (μ) is the logit of p_i and we compute the log of the likelihood which simplifies to $\sum_{i=1}^{6}[y_i\mu_i + \ln(1+\exp(\mu_i))]$.

$\text{logit}(p_1) = \mu_1 =$	$\beta_1 + u_1 e^{\tau_1}$
$\text{logit}(p_2) = \mu_2 =$	$\beta_1 + u_1 e^{\tau_1}$
$\text{logit}(p_3) = \mu_3 =$	$\beta_1 + \beta_2 + u_2 e^{\tau_1}$
$\text{logit}(p_4) = \mu_4 =$	$\beta_1 + \beta_2 + u_2 e^{\tau_1}$
$\text{logit}(p_5) = \mu_5 =$	$\beta_1 + u_3 e^{\tau_1}$
$\text{logit}(p_6) = \mu_6 =$	$\beta_1 + u_3 e^{\tau_1}$

The `PARAMETER_SECTION` contains a vector `Beta` which are the values of the fixed parameters from 1 to `kfixed` that are estimated in phase 1. It also contains a vector `Tau` which are the random parameters from 1 to `krand` and a `random_effects_vector` from 1 to `nre`. The random effect parameters and values are estimated in phase 2. If there is no random component in the model, then the phase is set to -1 so it will be ignored and a fixed effects model is fitted. The negative log-likelihood value (`objective_function_value`) is held in `g`.

The `PROCEDURE_SECTION` of the `TPL` file for computation of the objective function is fairly simple. It contains a loop over each of the `nre` random effects that calls the `SEPARABLE_FUNCTION n01_prior` which sums the negative log-likelihood of a standard normal distribution. That loop is only called if there are any random effects. The next loop is over each row in the data and it calls the `SEPARABLE_FUNCTION ll_i` to compute and sum the negative log-likelihood contribution for each response. In the call to `ll_i` you must pass the parameters and the random effects used in the calculation. Notice that the code specifies `u(randIndex(i))` which passes only the random effect values needed for the each row in the data. When used with the `-shess` run argument this improves execution speed (Skaug and Fournier 2011).

The `SEPARABLE_FUNCTION ll_i` computes the value of mu which is `fixedDM(i)*Beta` plus `randDM(i,j)*u(j)*mfexp(Tau(j))` for each column in the random design matrix (`krand`). For our simple example, `krand`=1 and `randDM(i,j)`=1, so the code simplifies to the calculations shown above.

Now let's expand on the example with a model that has both an individual and temporal random intercept. The `TPL` file is the same and only the `DAT` file changes. Now we have `nre`=5 random values (3 for `id` and 2 for `time`) and we have `krand`=2 columns in the random design matrix, one for the intercept of each random effect.

number of rows in data (n)
6
vector of 6 responses (y_i, i=1 to n)
0 1 0 1 1 0
number of columns in the fixed effect design matrix (kfixed)
2
fixed effect design matrix (fixedDM_ij; i=1 to n; j=1 to kfixed) (first column is intercept; second column is male effect)
1 0
1 0
1 1
1 1
1 0
1 0
number of random effect values (nre) (for this model, one for each id)
5
number of columns in the random effect design matrix (krand)
2
random effect design matrix (randDM_ij; i=1 to n; j=1 to krand)
1 1
1 1
1 1
1 1
1 1
1 1
random effect indices (randIndex_ij; i=1 to n; j=1 to krand)(values from 1 to nre)
1 4
1 5
2 4
2 5
3 4
3 5

The important design aspect is the use of a single random effect vector to contain the values for each type of random effect (e.g. `id` and `time`). This is important because random effects must be passed as arguments to the `SEPARABLE_FUNCTION`. Had we decided to use a different vector for each type of random effect, we would have to change the definition of the `SEPARABLE_FUNCTION` and its call in the `PROCEDURE_SECTION`. Modification of the code is avoided by using a single vector and using indices to specify which to use with the random design matrix values. For this model there are 4 parameters (β_1,β_2,τ_1,τ_2) and the values of `mu` are:

$\text{logit}(p_1) = \mu_1 =$	$\beta_1 + u_1 e^{\tau_1} + u_4 e^{\tau_2}$
$\text{logit}(p_2) = \mu_2 =$	$\beta_1 + u_1 e^{\tau_1} + u_5 e^{\tau_2}$
$\text{logit}(p_3) = \mu_3 =$	$\beta_1 + \beta_2 + u_2 e^{\tau_1} + u_4 e^{\tau_2}$
$\text{logit}(p_4) = \mu_4 =$	$\beta_1 + \beta_2 + u_2 e^{\tau_1} + u_5 e^{\tau_2}$
$\text{logit}(p_5) = \mu_5 =$	$\beta_1 + u_3 e^{\tau_1} + u_4 e^{\tau_2}$
$\text{logit}(p_6) = \mu_6 =$	$\beta_1 + u_3 e^{\tau_1} + u_5 e^{\tau_2}$

R Code

Now you certainly would not want to construct the `DAT` file by hand for anything other than a simple illustrative example like this one. It would be tedious and error prone. This is where `R` and the `R` package `R2admb` become an invaluable tool. We have used functions in `R` listed in the Appendix: 1) `proc.form` - parses a formula using the structure specified in the `R` package `lme4`, 2) `mixed.model.admb` - calls `proc.form` with a formula and then with a data frame constructs the design matrices and indices needed for the `DAT` file, and 3) `mixed.model.dat` - writes the model components structures to the `DAT` file. The function `mixed.model.dat` only writes out a mixed effect model structure and not the complete `DAT` file. We chose to do this because it may be included as part of a `DAT` file that may have other components.

Below are code that construct a simulated data set, fit a mixed-effects logistic model with `ADMB` using the code described and then fit the same model with the function `lmer` in the package `lme4` to demonstrate that the same results are obtained.

```
> # Load R2admb package from the library
> library(R2admb)
> # Set environment variables PATH and ADMB_HOME to enable ADMB connection
```

```
> Sys.setenv(PATH = paste("c:/admb/bin;c:admb/utilities;c:/MinGW/bin;",
+                Sys.getenv("PATH"), sep = ";"))
> Sys.setenv(ADMB_HOME = "c:/admb")
> # create simulated data with 2 factor variables f and g with
> # N(0,.3) and N(0,1) random effects
> fn=rnorm(10,0,.3)
> gn=rnorm(20,0,1)
> data=cbind(data.frame(y=rbinom(400,1,
+       p=plogis(log(.3/.7)+rowSums(expand.grid(fn,gn))))),
+       rbind(expand.grid(f=factor(1:10),g=factor(1:20)),
+       expand.grid(f=factor(1:10),g=factor(1:20))))
> # Use R2admb functions to compile the ADMB TPL file
> compile_admb("mixed",re=TRUE)
> # Remove any leftover files
> clean_admb("mixed")
> # Create a connection to a file mixed.dat (DAT file)
> con=file("mixed.dat",open="wt")
> # Write out n and responses y to DAT file
> write(nrow(data),con,append=FALSE)
> write(data$y,con,append=TRUE)
> clean_admb("mixed")
> con=file("mixed.dat",open="wt")
> write(nrow(data),con,append=FALSE)
> write(data$y,con,append=TRUE)
> # Write mixed model structures to DAT file
> mixed.model.dat(mixed.model.admb(~1 +(1|f) +(1|g),data),con)
> # Close connection
> close(con)
> # Run program
> run_admb("mixed",extra.args="-shess")
> # Read results into R object
> results=read_admb("mixed")
> results

Model file: mixed
Negative log-likelihood: 230.449
Coefficients:
     Beta     Tau.1     Tau.2
-0.904190 -0.743352  0.038662
```

```
> # The value of exp(Tau)^2 should match the variances from lmer
> exp(results$coeflist$Tau*2)

[1] 0.2261167 1.0803921

> # Compare to lmer from lme4
> library(lme4)
> lme4res=lmer(y~1 +(1|f) +(1|g),data,family="binomial",REML=FALSE)
> lme4res

Generalized linear mixed model fit by the Laplace approximation
Formula: y ~ 1 + (1 | f) + (1 | g)
   Data: data
   AIC   BIC logLik deviance
 466.9 478.9 -230.4    460.9
Random effects:
 Groups Name        Variance Std.Dev.
 g      (Intercept) 1.08039  1.03942
 f      (Intercept) 0.22613  0.47553
Number of obs: 400, groups: g, 20; f, 10

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.9042     0.3020  -2.994  0.00275 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

>
```

References

[1] Skaug, H. and D. Fournier. 2011. Random effects in AD Model Builder: ADMB-RE User Guide. Version 10.0. http://admb-project.googlecode.com/files/admbre-10.0-rev1.pdf.

Appendix - mixed.tpl

```
// Skeleton template for a mixed effects model structure; with logistic regression likelihood
// Jeff Laake; 22 Jan 2013
DATA_SECTION
    init_int n;                              // number of rows in data
    init_vector y(1,n);                      // vector of responses
    init_int kfixed;                         // number of columns in design matrix(DM) for fixed effects
    init_matrix fixedDM(1,n,1,kfixed);       // fixed effect DM
    init_int nre;                            // number of random effects
    int phase;                               // if no random effects set phase to -1 otherwise 2
        !! phase=2;
        !! if(nre==0)phase=-1;
    init_int krand;                          // number of columns in random effect DM
    init_matrix randDM(1,n,1,krand);         // random effect DM
    init_imatrix randIndex(1,n,1,krand);     // random effect indices for DM

PARAMETER_SECTION
    init_vector Beta(1,kfixed,1);            // parameter vector for fixed effects
    init_vector Tau(1,krand,phase);          // parameter vector for log(sigma)
    objective_function_value g;              // objective function - negative log-likelihood
    random_effects_vector u(1,nre,phase);    // random effects vector

PROCEDURE_SECTION
    int i;                                                    // index variable
    g=0;                                                      // initialize negative log-likelihood
        cout<<"nre = "<< nre <<endl;
        cout<<"kfixed = "<< kfixed <<endl;
        cout<<"krand = "<< krand <<endl;
    for (i=1;i<=nre;i++)                                      // if any random effects compute likelihood contribution for each
        n01_prior(u(i));                                      // u's are N(0,1) distributed
        cout<<"g = "<< g <<endl;

    for(i=1;i<=n;i++)                                         // loop over rows in data
          ll_i(i,Beta,Tau,u(randIndex(i)));                   //compute negative log-likelihood for each row with separable function
        cout<<"g = "<< g <<endl;

SEPARABLE_FUNCTION void n01_prior(const prevariable&  u)  // taken from glmmadmb.tpl; uses PI
 g -= -0.5*log(2.0*PI) - 0.5*square(u);

SEPARABLE_FUNCTION void ll_i(const int i, const dvar_vector& Beta,const dvar_vector& Tau,const dvar_vector& u)
    int j;
    dvariable mu;                                             // link value
    mu=fixedDM(i)*Beta;                                       // fixed portion
    for(j=1;j<=krand;j++)                                     // random portion (if any)
        mu+=randDM(i,j)*u(j)*mfexp(Tau(j));

    g-= y(i)*mu - log(1+exp(mu));                             // Bernoulli likelihood with logit link
```

Appendix - proc.form, mixed.model.admb, mixed.model.dat

```
#' Mixed effect model formula parser
#'
#' Parses a mixed effect model in the lme4 structure of  ~fixed +(re1|g1) +...+(ren|gn)
#'
#' @param f formula for mixed effect mode in the form used in lme4; ~fixed +(re1|g1) +...+(ren|gn)
#' @return A list with elements fix.model and re.model. fix.model contains the formula for the fixed effects;
#' re.model contains elements sub, the grouping formula and model the design formula for the
#' random effect. Each formula is of type character and must be wrapped with as.formula in use with model.matrix
#' @author Devin Johnson <devin.johnson@@noaa.gov>
#'
proc.form <- function(f){
        tms <- terms(f)
        tms.lab <- attr(tms, "term.labels")
        tms.lst <- strsplit(tms.lab, rep(" | ",length(tms.lab)), fixed=TRUE)
        fix.var <- attr(tms, "term.labels")[sapply(tms.lst, "length")==1]
        if(length(fix.var)==0)
                fix.model="~1"
        else
                fix.model <- paste("~ ",paste(fix.var, collapse=" + "))
        re.lst <- tms.lst[sapply(tms.lst, "length")==2]
        if(length(re.lst)==0) re.model <- NULL
        else{
                re.model <- lapply(re.lst, function(x){list(model=paste("~",x[1]), sub=paste("~",x[2],"-1", collapse=""))})
        }
        return(list(fix.model=fix.model, re.model=re.model))
}
```

```
#' Mixed effect model construction
#'
#' Functions that develop structures needed for a mixed effect model
#'
#' mixed.model.admb - creates design matrices and supporting index matrices
#' for use of mixed model in ADMB
#'
#' mixed.model.dat - writes to data file (con) for fixed and random effect stuctures
#'
#' @aliases mixed.model.admb mixed.model.dat
#' @usage mixed.model.admb(formula,data)
#'        mixed.model.dat(x,con)
#' @param formula formula for mixed effect mode in the form used in lme4; ~fixed +(re1|g1) +...+(ren|gn)
#' @param data dataframe used to construct the design matrices from the formula
#' @param x list structure created by mixed.model.admb
#' @param con connection to data file which contents will be appended
#' @return mixed.model.admb returns a list with elements fixed.dm, the design matrix for
#' the fixed effects; re.dm, a combined design matrix for all of the random effects; and
#' re.indices, matrix of indices into a single vector of random effects to be applied to the
#' design matrix location.
#' @author Jeff Laake <jeff.laake@@noaa.gov>
#'
mixed.model.admb=function(formula,data)
{
# parse formula for fixed and random effects
  mlist=proc.form(formula)
# construct design matrix for fixed effects
  fixed.dm=model.matrix(as.formula(mlist$fix.model),data)
# remainder of code is for random effects unless NULL
  reindex=0
  re.dm=NULL
  re.indices=NULL
  if(!is.null(mlist$re.model))
  {
#    Loop over each random effect component
     for(i in 1:length(mlist$re.model))
         {
#      Make sure each variable used to define random effect group is a factor variable
           if(!all(sapply(model.frame(as.formula(mlist$re.model[[i]]$sub),data),is.factor)))
               stop(paste("\n one or more variables in",mlist$re.model[[i]]$sub,"is not a factor variable\n"))
       else
           {
#         Compute design matrix for grouping variables
          zz=model.matrix(as.formula(mlist$re.model[[i]]$sub),data)
#         Not all combinations of factor variable(s) may be used so only use those observed in the data
                  used.columns=which(colSums(zz)>0)
                  nre=length(used.columns)
#         Compute the indices for this particular grouping structure and reindex if any missing
          indices=rowSums(zz*col(zz))
                  if(nre!=ncol(zz))indices=match(indices,used.columns)
#         Compute the design matrix for the random effect formula
          zz=model.matrix(as.formula(mlist$re.model[[i]]$model),data)
#         Now shift indices to refer to a single vector of random effects across all re groupings
                  ng=max(indices)
          indices=matrix(indices,nrow=length(indices),ncol=ncol(zz))+reindex
          indices=t(t(indices)+cumsum(c(0,rep(ng,ncol(zz)-1))))
          reindex=max(indices)
#         Bind random effect design matrices (re.dm), indices into random effects vector (re.indices) and
#         index for the random effect sigma parameter (re.sigma)
                  re.dm=cbind(re.dm,zz)
```

```
                re.indices=cbind(re.indices,indices)
            }
          }
    }
    return(list(fixed.dm=fixed.dm,re.dm=re.dm,re.indices=re.indices))
}
mixed.model.dat=function(x,con)
{
        # number of columns of fixed dm
        write(ncol(x$fixed.dm),con,append=TRUE)
        # fixed dm
        write(t(x$fixed.dm),con,ncolumns=ncol(x$fixed.dm),append=TRUE)
    if(!is.null(x$re.dm))
        {
                # number of random effects
                write(max(x$re.indices),con,append=TRUE)
                # number of columns of re dm
                write(ncol(x$re.dm),con,append=TRUE)
                # re dm
                write(t(x$re.dm),con,ncolumns=ncol(x$re.dm),append=TRUE)
                # re indices
                write(t(x$re.indices),con,ncolumns=ncol(x$re.indices),append=TRUE)
        }
        else
        {
                # 0 no re
                write(0,con,append=TRUE)
                # number of re =0
                write(0,con,append=TRUE)
                # number of columns of re dm=0
                write(0,con,append=TRUE)
        }
    invisible()
}
```