SUPPLEMENTAL INSTRUCTIONS FOR REG-DIF

Supplemental Material for

Simplifying the Assessment of Measurement Invariance over Multiple Background Variables: Using Regularized Moderated Nonlinear Factor Analysis to Detect Differential Item Functioning

We provide details on running regularized differential item functioning (Reg-DIF) using the moderated nonlinear factor analysis model (MNLFA) presented in Bauer, Belzak & Cole. We cover data management, model fitting, and model selection and re-fitting.

Data Management

The data must be prepared before running Reg-DIF. We present the general set-up below.

We first import the data. In SAS, this can be accomplished by directly loading a data file from a specified directory. The directory with the data file is listed on the `INFILE` line: `"C:\Users\Smith\data.dat"`. This will need to be changed accordingly. In this data file, there are no column names. We give names on the `INPUT` line. We also define the data characteristics as macro variables. Note that in SAS, macro variables are symbols that refer to user-specific values and, once defined, can be used in subsequent code. Here, we define the sample size (`sample_size`), the number of scale items (`number_items`), and the number of DIF covariates to be analyzed with Reg-DIF (`number_dif`). These will be useful in later code.

```
*Import data;
DATA wide;
 INFILE "C:\Users\Smith\example.dat";
 INPUT ID Age Gender Study Item_1 Item_2 Item_3 Item_4 Item_5 Item_6;
RUN;

*Define data characteristics;
%LET sample_size = 500;
%LET number_items = 6;
%LET number_dif = 3;
```

Above, we named the imported data set `wide` (next to **`DATA`**) because it is in "wide" format: one row per case and one column per variable. This is shown below. For brevity, we use … to indicate continuing data.

```
ID   Age   Gender   Study   Item_1   Item_2   Item_3   Item_4   Item_5   Item_6

1    -2     -1       -1      0        0        0        0        0        0

2     0     -1       -1      0        0        0        0        0        0

3     3     -1       -1      0        0        0        0        0        0

4     1     -1       -1      0        1        1        1        1        1

5    -2     -1       -1      0        0        0        0        0        0

…     …      …        …      …        …        …        …        …        …
```

To run Reg-DIF in SAS, however, we need to stack the data into "long" format: multiple rows per case collapsed across items. This is done using an `ARRAY` statement that defines a set of elements to process, and a `DO` statement that repeatedly processes statements based on the indexing number (e.g., `1`) until the `END` statement is specified. This code creates a new data set called `long` using the `wide` data set, and stacks each item number into the `Item` variable and each item response into the `Response` variable.

```
*Transform data into long format;
DATA long; SET wide;
 ARRAY DV [6] Item_1 Item_2 Item_3 Item_4 Item_5 Item_6;
 DO i = 1 to 6;
    Item = i;
    Response = DV[i];
    OUTPUT;
 END;
 KEEP ID Age Gender Study Item Response;
RUN;
```

There are now 6 instances of each case's `ID`, `Age`, `Gender`, and `Study` alongside the item numbers (`Item`) and item responses (`Response`).

```
ID    Age    Gender    Study    Item    Response
1     -2     -1        -1       1       0
1     -2     -1        -1       2       0
1     -2     -1        -1       3       0
1     -2     -1        -1       4       0
1     -2     -1        -1       5       0
1     -2     -1        -1       6       0
2     0      -1        -1       1       0
2     0      -1        -1       2       0
2     0      -1        -1       3       0
2     0      -1        -1       4       0
2     0      -1        -1       5       0
2     0      -1        -1       6       0
3     3      -1        -1       1       0
3     3      -1        -1       2       0
3     3      -1        -1       3       0
3     3      -1        -1       4       0
3     3      -1        -1       5       0
3     3      -1        -1       6       0
4     1      -1        -1       1       0
4     1      -1        -1       2       1
…     …      …                  …       …
```

Model Fitting

Once the data are prepared, we may now proceed to model fitting. We show SAS code for this below.

Model 1: Find starting values for MNLFA.

Good starting values are important to have when using the NLMIXED procedure in SAS. Namely, the convergence time is much faster, and in complex models, some models may not even be able to find suitable estimates without starting values. Therefore, we estimate a preliminary model to find good starting values for Reg-DIF.

The code below fits a MNLFA model with **`PROC NLMIXED`**. We specify `long` in the `DATA` statement. The `GCONV` and `QPOINTS` statements are the convergence criterion and the number of adaptive quadrature points for estimation. `MAXITER` and `MAXFUNC` are optional statements that increase the number of possible iterations and function calls in the optimization process. These model specifications may be useful in most applications. The `PARMS` statement gives the names and initial values of the parameters. We list the baseline intercepts and loadings, intercept and loading DIF, and latent distribution parameters with reasonable starting values. For instance, `k1_1=0` means that we give the intercept DIF parameter for the `Study` variable on item 1 a starting value of 0.

Next, we write the likelihood function. First, we use `IF THEN DO` statements to write inverse logistic link functions for each of the binary items. This is done by specifying the linear function `gmu` as combination of the baseline intercept (e.g., `v_1`), the intercept DIF parameters and covariates (e.g., `k1_1*Age`, `k2_1*Gender`, `k3_1*Study`), the baseline loading (e.g., `l_1`), and the loading DIF parameters and covariates (e.g., `w1_1*Age`, `w2_1*Gender`, `w3_1*Study`) multiplied to the latent variable. Second, we use an `IF THEN ELSE` statement to input the linear function into the inverse link function `mu` based on whether `Response=1` or not. That is, if the item was endorsed, this means the expected value `mu` is equal to the probability `1/(1+exp(-gmu))`. If the item was not endorsed, `mu` is equal to the probability `1 - 1/(1+exp(-gmu))`. Lastly, we specify another `IF THEN ELSE` statement as an error trap to avoid any convergence problems. That is, the likelihood function `ll` is computed only if `mu` is not too close to zero. If so, it's set to an arbitrarily small number: `-1e100`. We finalize the `IF THEN DO` statement with `END`. This is repeated for each item in the measure. To save space, we use … to indicate code for items 3-10.

The final step is specifying the model and the latent distribution. We first incorporate impact on the latent mean `alpha` and latent variance `psi`. Corresponding to Bauer and Hussong (2009), the latent mean is defined as deterministic linear function of the DIF covariates: `g1*Age + g2*Gender + g3*Gender`; and the latent variance is defined as a deterministic log-linear function of the DIF covariates: `exp(b1*Age + b2*Gender + b3*Gender)`. We then specify the `MODEL` statement. This is where we input our user-specified log-likelihood function with `GENERAL(ll)`, defining `Response` as the outcome. Next, the `RANDOM` statement defines the latent variable and its distribution. We specify a normal distribution with the `NORMAL` statement, including mean `alpha` and variance `psi` which were defined above. The `SUBJECT` statement indicates the outcome variable is grouped by `ID`. Finally, the `ODS OUTPUT` statement saves the model parameter estimates into a new data set we call `estimates`. This will be important for implementing Reg-DIF, as these will supply the starting values for each new model fit.

```
PROC NLMIXED DATA = long GCONV = .000001 QPOINTS = 15 MAXITER = 500 MAXFUNC
= 3000;
PARMS /*Baseline Intercepts*/
       v_1=0 v_2=0 v_3=0 v_4=0 v_5=0 v_6=0

       /*Baseline Loadings*/
       l_1=.7 l_2=.7 l_3=.7 l_4=.7 l_5=.7 l_6=.7

       /*Intercept DIF*/
       /*Age*/
       k1_1=0 k1_2=0 k1_3=0 k1_4=0 k1_5=0 k1_6=0
       /*Gender*/
       k2_1=0 k2_2=0 k2_3=0 k2_4=0 k2_5=0 k2_6=0
       /*Study*/
       k3_1=0 k3_2=0 k3_3=0 k3_4=0 k3_5=0 k3_6=0

       /*Loading DIF*/
       /*Age*/
       w1_1=0 w1_2=0 w1_3=0 w1_4=0 w1_5=0 w1_6=0
       /*Gender*/
       w2_1=0 w2_2=0 w2_3=0 w2_4=0 w2_5=0 w2_6=0
       /*Study*/
       w3_1=0 w3_2=0 w3_3=0 w3_4=0 w3_5=0 w3_6=0

       /*Latent Distribution (Eta)*/
       g1=0 g2=0 g3=0 b1=0 b2=0 b3=0
;

IF (item=1) THEN DO; *Bernoulli w/ Logit Link;
      gmu = (v_1 + (k1_1*Age + k2_1*Gender + k3_1*Study)) + (l_1 + (w1_1*Age
      + w2_1*Gender + w3_1*Study))*Eta;
      IF (Response=1) THEN mu = 1/(1+exp(-gmu));
      ELSE mu = 1 - ( 1/(1+exp(-gmu)) );
      IF (mu > 1e-8) THEN ll = log(mu);
      ELSE ll = -1e100;
END;

…

IF (item=6) THEN DO;
      gmu = (v_6 + (k1_6*Age + k2_6*Gender + k3_6*Study)) + (l_6 + (w1_6*Age
      + w2_6*Gender + w3_6*Study))*Eta;
      IF (Response=1) THEN mu = 1/(1+exp(-gmu));
      ELSE mu = 1 - ( 1/(1+exp(-gmu)) );
      IF (mu > 1e-8) THEN ll = log(mu);
      ELSE ll = -1e100;
END;
alpha = g1*Age + g2*Gender + g3*Study;
psi = exp(b1*Age + b2*Gender + b3*Study);
MODEL Response~GENERAL(ll);
RANDOM Eta~NORMAL(alpha,psi) SUBJECT=ID;
ODS OUTPUT ParameterEstimates=estimates;
RUN;
```

Model 2: Run Reg-DIF on MNFLA.

Running Reg-DIF in SAS requires a macro environment in which multiple iterations of code are completed based on an index variable. To set up this up, some preliminary coding is required. Namely, we create separate macro variables to be used in a macro environment. This is shown below.

In the first chunk of code, we save the starting values from the previous model, keeping only the parameter name (`Parameter`) and the parameter estimate (`Estimate`). The second chunk creates a range of tuning parameter values and places them into the `tuning` data set. This range was determined with trial and error. Other ranges with varying increments may be more useful with different data sets. The third chunk creates a macro variable called `values` that lists these tuning values, separated by a space `' '`. The fourth code chunk creates separate macro variables for the parameter names. Doing this will become more evident in the Reg-DIF macro code (shown next). The fifth and sixth chunks of codes also define macro variables that will be used in Reg-DIF.

```
*CODE CHUNK 1: Save starting values;
DATA start;
 SET estimates;
 KEEP Parameter Estimate;
RUN;

*CODE CHUNK 2: Create range of tuning parameter values;
DATA tuning;
 DO sequence = 2 TO 100 BY 2; OUTPUT; END;
 DO sequence = 105 TO 500 BY 5; OUTPUT; END;
 DO sequence = 510 TO 800 BY 10; OUTPUT; END;
 DO sequence = 900 TO 10000 BY 100; OUTPUT; END;
RUN;

*CODE CHUNK 3: Create macro variable of tuning values;
PROC SQL NOPRINT;
 SELECT sequence INTO :values separated BY ' '
 FROM tuning;
quit;

*CODE CHUNK 4: Create macro variables for parameter names;
DATA start;
 SET start;
 CALL symput(Parameter,Parameter);
RUN;

*CODE CHUNK 5: Define macro variable 'finish' to equal 0;
%LET finish = 0;

*CODE CHUNK 6: Define macro for the number of tuning values;
%LET count = %SYSFUNC(countw(&values," "));
```

Having created the necessary macro variables, we run Reg-DIF in a macro environment. We describe this code now.

In the first code chunk below, the macro environment is specified with the `%MACRO` statement. We note that `%` signs before statements indicate macro-styled statements. This macro must be named, which we call `Search()`. Next, we use a `%DO %TO` macro statement to iterate through the tuning parameter values. The `&xxx.` notation indicates a macro variable, and here we use the previously defined `&count.` variable to indicate the maximum number of tuning values (see seventh code chunk above). Because we are iterating through non-sequential numbers (see second code chunk above), we must pass the macro variable `&values` (see third code chunk above) through the `%SCAN` statement. In other words, this scans/iterates across the non-sequential tuning parameter values.

The second code chunk is `PROC NLMIXED` where most of the optimization conditions remain the same. A peculiarly about SAS is that non-macro code within a macro environment does not show the same colored delineation between SAS statements and user-defined variables. For instance, `PROC NLMIXED` is in black text here, whereas outside of the macro environment, **`PROC NLMIXED`** is in blue text. Nevertheless, the code still works. For Reg-DIF, we define the tuning parameter `tau`. This tuning parameter takes the current value in the `&value.` macro variable (based on the iteration) and divides by **`100000`** . We identified this scaling factor through trial and error, although this scaling may be suitable for most applications. Next, in the `PARMS` statement, we used the starting values from the first model (`start`) for starting values in Reg-DIF. This was simply given with the `/ DATA = start` specification. These starting values will be updated after each model run (see below). The `penalty` estimate was defined by taking the sum of the absolute values of the DIF parameter estimates and multiplying by `tau`, the tuning value. Defining the log-likelihood function and the model set-up remained approximately the same from above. That is, we used `IF THEN DO` and `IF THEN ELSE` statements to specify the logistic inverse link functions, as well as the `MODEL` and `RANDOM` statements to specify the latent distribution. There were some notable differences, however. First, we used macro variables for the parameter names (see fourth code chunk above) so that when parameter estimates were penalized to zero, the parameter names (e.g., `&k1_1.`) would also be set to 0, therefore removing the covariate from the model. Second, we subtracted the `penalty` from the log-likelihood function `ll` to obtain `ll_reg`, and then optimized over this new penalized function. We also used the `ESTIMATE` statement to print the `penalty` estimate in the model results. This required a restatement of the penalty function. Finally, in addition to saving model estimates in the data set `estimates`, we also saved fit statistics `fit` and the `penalty` estimate using the `ODS OUTPUT` statement.

The third chunk of code uses the parameter estimates from the previous model fit to update the macro variables and starting values for the next model fit. Essentially, this code takes those parameter estimates that are close to zero, between `-`**`.00001`** and **`.00001`**, and sets them to equal zero. These parameters estimates that are extremely close to zero are saved in the `zero` data set. This code also ensures that any estimate in `zero` will make the corresponding parameter macro variable (e.g., `&k1_1.`) also equal to zero, which in turn will remove the corresponding DIF covariate from the model. We also added some code that would prevent the impact parameters (e.g., `&g1.`) from being removed from the model regardless of how close to zero the parameters are. Finally, the non-zero parameter estimates are saved for starting values in the next model fit (i.e., next tuning value in `&value.`).

In the fourth code chunk, fit statistics, the penalty estimate, the tuning value, and parameter estimates are saved for each model. These model results are appended to the previous model results, resulting in a large chain of results for the full Reg-DIF routine. Specifically, we use `%IF %THEN %DO` statements to save these model results and merge them together. You may notice that there are two replications of the same code after each `%IF %THEN %DO` statement. Although admittedly clunky, this allows us to append each model result to the preceding model results.

The fifth and final code chunk of this macro environment provides a stopping rule: when the number of parameters in the model equals the number of baseline intercepts, baseline loadings, and impact parameters, the macro stops. Because this implies that all the DIF covariates have been removed from the model, there is no need to continue Reg-DIF. We use the `&finish.` macro variable that was defined in the sixth code chunk above. More specifically, the first set of code determines whether the stopping criterion has been met: that is, is `&finish. = 1`? If so, the macro stops (`%THEN %RETURN`). If not, `PROC SQL` in the second set of code determines the number of parameters that are still in the model. The third and final set of code then determines whether the number of parameters in the model equals the minimum possible number of parameters. In other words, have all the DIF parameters in the `penalty` been penalized out of the model. We use two macro variables to do this. The first one is `&number_items.` and the second is `&number_dif.`, defined in the first code on page 1. We find the sum of the number of items (e.g., 6) and the number of DIF covariates (e.g., 3), that is, 6 + 3 = 9, and multiply by 2 to get the minimum number of parameters (without any DIF parameters): 2 * 9 = 18. We multiply by 2 because there are two sets of DIF parameters (slopes and intercepts) and because there are two sets of latent distribution functions (mean and variance) that depend on the DIF covariates. If researchers wish only penalize the DIF intercept parameters, for instance, there will only be one set of 6 parameters to penalize. Thus, this part of the code would need to change to reflect that (e.g., 6 intercept DIF parameters + [2 * 3 DIF covariates] = 12 minimum parameters).

The final step is finishing the macro with the **`%MEND`** statement and `%`***`Search`***`()` name.

```
*CODE CHUNK 1: Specify macro environment;
%MACRO Search();
%DO i = 1 %TO &count.;
%LET value = %SCAN(&values,&i," ");

*CODE CHUNK 2: Run Reg-DIF in NLMIXED;
PROC NLMIXED DATA = long GCONV = .000001 QPOINTS = 15 MAXITER = 500 MAXFUNC
= 3000;
tau = &value./100000;
PARMS / DATA = start;

penalty = tau*(abs(&k1_1.) + abs(&k2_1.) + abs(&k3_1.) +
               abs(&k1_2.) + abs(&k2_2.) + abs(&k3_2.) +
               abs(&k1_3.) + abs(&k2_3.) + abs(&k3_3.) +
               abs(&k1_4.) + abs(&k2_4.) + abs(&k3_4.) +
               abs(&k1_5.) + abs(&k2_5.) + abs(&k3_5.) +
               abs(&k1_6.) + abs(&k2_6.) + abs(&k3_6.) +
               abs(&w1_1.) + abs(&w2_1.) + abs(&w3_1.) +
               abs(&w1_2.) + abs(&w2_2.) + abs(&w3_2.) +
               abs(&w1_3.) + abs(&w2_3.) + abs(&w3_3.) +
               abs(&w1_4.) + abs(&w2_4.) + abs(&w3_4.) +
               abs(&w1_5.) + abs(&w2_5.) + abs(&w3_5.) +
               abs(&w1_6.) + abs(&w2_6.) + abs(&w3_6.))
;

IF (item=1) THEN DO;
      gmu = (&v_1. + (&k1_1.*Age + &k2_1.*Gender + &k3_1.*Study)) + (&l_1. +
      (&w1_1.*Age + &w2_1.*Gender + &w3_1.*Study))*eta;
      IF (Response=1) THEN mu = 1/(1+exp(-gmu));
      ELSE mu = 1 - ( 1/(1+exp(-gmu)) );
      IF (mu > 1e-8) THEN ll = log(mu);
      ELSE ll = -1e100;
END;

…

IF (item=6) THEN DO;
      gmu = (&v_6. + (&k1_6.*Age + &k2_6.*Gender + &k3_6.*Study)) + (&l_6. +
      (&w1_6.*Age + &w2_6.*Gender + &w3_6.*Study))*eta;
      IF (Response=1) THEN mu = 1/(1+exp(-gmu));
      ELSE mu = 1 - ( 1/(1+exp(-gmu)) );
      IF (mu > 1e-8) THEN ll = log(mu);
      ELSE ll = -1e100;
END;
alpha = &g1.*Age + &g2.*Gender + &g3.*Study;
psi = exp(&b1.*Age + &b2.*Gender + &b3.*Study);
ll_reg = ll - penalty;
MODEL Response~GENERAL(ll_reg);
RANDOM eta~NORMAL(alpha,psi) SUBJECT=id;
ESTIMATE 'penalty' tau*(abs(&k1_1.) + abs(&k2_1.) + abs(&k3_1.) +
               abs(&k1_2.) + abs(&k2_2.) + abs(&k3_2.) +
               abs(&k1_3.) + abs(&k2_3.) + abs(&k3_3.) +
               abs(&k1_4.) + abs(&k2_4.) + abs(&k3_4.) +
               abs(&k1_5.) + abs(&k2_5.) + abs(&k3_5.) +
               abs(&k1_6.) + abs(&k2_6.) + abs(&k3_6.) +
               abs(&w1_1.) + abs(&w2_1.) + abs(&w3_1.) +
               abs(&w1_2.) + abs(&w2_2.) + abs(&w3_2.) +
               abs(&w1_3.) + abs(&w2_3.) + abs(&w3_3.) +
               abs(&w1_4.) + abs(&w2_4.) + abs(&w3_4.) +
               abs(&w1_5.) + abs(&w2_5.) + abs(&w3_5.) +
               abs(&w1_6.) + abs(&w2_6.) + abs(&w3_6.));
ODS OUTPUT ParameterEstimates=estimates FitStatistics=fit
AdditionalEstimates=penalty;
run;

*CODE CHUNK 3: Update macro variables and starting values;

   *Rename penalty estimate (for later use);
   DATA penalty(KEEP=Estimate RENAME=(Estimate=penalty)); SET penalty;
   RUN;

   *Grab parameters with estimate close to 0;
   PROC SQL;
    CREATE TABLE zero AS
    SELECT Parameter FROM estimates WHERE Estimate <= .00001 and Estimate >=
   -.00001;
   QUIT;
   *For those estimates close to 0, change macros of parameter names to
   equal 0 (but not impact parameters);
   DATA zero;
    SET zero;
    IF first(left(Parameter))='b' THEN DELETE;
    IF first(left(Parameter))='g' THEN DELETE;
    CALL symput(Parameter, 0);
   RUN;

   *Save (nonzero) estimates for starting values;
   DATA start;
    SET estimates;
    WHERE Estimate not between .00001 and -.00001 or
    (left(Parameter))='b' or first(left(Parameter))='g';
    KEEP Parameter Estimate;
   RUN;

*CODE CHUNK 4: Save fit statistics, penalty estimate, tuning parameter
value, and parameter estimates for each model and append to previous models;

%IF &value. = 2 %THEN %DO;
DATA fit; SET fit(obs=1);
 tau = &value./100000;
RUN;

DATA estimates; SET estimates;
 group_var = &value.;
RUN;

DATA fit; MERGE fit penalty estimates;
RUN;

DATA all; SET fit;
RUN;
%END;

%ELSE %DO;
DATA fit; SET fit(obs=1);
 tau = &value./100000;
RUN;

DATA estimates; SET estimates;
 group_var = &value.;
RUN;

DATA fit; MERGE fit penalty estimates;
RUN;

DATA all; SET all fit;
RUN;
%END;

DATA fit; SET _NULL_;
RUN;

DATA estimates; SET _NULL_;
```

```
RUN;

*CODE CHUNK 5: Stopping rule once all DIF covariates have been removed;

   *Once all DIF covariates have been removed, finish the macro;
   %IF &finish. = 1 %THEN %RETURN;

   *Determine whether all parameters have been removed;
   PROC SQL NOPRINT;
    select count(*) into :last from start;
   QUIT;

   *If number of parameters equals only those estimated in equals 2*(number
   of items) + (impact parameters) then fit one more model;
   %IF &last. = 2*(&number_items. + &number_dif.) %THEN %LET finish = 1;

%END;

%MEND; %Search();
```

Model Selection and Re-Fitting

The last stage of Reg-DIF is selecting a final model and re-fitting it without the penalty (i.e., relaxed lasso). In the paper, we used the lowest BIC as our criterion for selection. We show code for doing this below.

We first select the final model based on BIC, which requires some "backtracking" computation. That is, we must calculate BIC using the log-likelihood value and the degrees of freedom for each model *as if* they were not penalized. We do so with the following. The first code chunk calculates the regularized log-likelihood function `Value` for each model by subtracting out the `penalty` value (scaled for sample size in the `long` data set, which is given with the macro variable `&long_sampsize.`). This results in a "purified" log-likelihood function value called `loglik_nopen`. The second code chunk finds the number of non-zero parameters for each model to be used for the new degrees of freedom. The third code chunk calculates BIC using the updated log-likelihood value from the first chunk and the new degrees of freedom from the second chunk. The fourth code chunk identifies the final model with the lowest BIC using the tuning parameter value, and saves this as the macro variable `&tune.`. The fifth and final code chunk saves the estimates for the final model in the data set called `final`.

```
*CODE CHUNK 1: Compute regularized log-likelihood function without penalty
parameter;
DATA all; SET all;
 loglik_nopen = Value/2 - penalty*(&sample_size.*&number_items.);
RUN;

*CODE CHUNK 2: Create variable that equals the number of nonzero parameters
for each model;
PROC SQL;
 CREATE TABLE select AS
 SELECT *, count(Parameter) AS num_par
      FROM all
      GROUP BY group_var
      ORDER BY group_var, Parameter;
```

```
RUN;

*CODE CHUNK 3: Drop duplicate values of nonzero parameters and compute BIC;
DATA select(DROP=num_par); SET select;
 IF not missing(Value) THEN pars=num_par;
 ELSE pars=Value;
 bic = 2*loglik_nopen + pars*log(&sample_size.);
RUN;

*CODE CHUNK 4: Find model where BIC is at a minimum;
PROC SQL;
 SELECT group_var
      INTO :tune TRIMMED
      FROM select
      HAVING bic=min(bic);
RUN;

*CODE CHUNK 5: Save estimates for final model;
PROC SQL;
 CREATE TABLE final AS
 SELECT Parameter,Estimate
      FROM select
      WHERE group_var=&tune.;
RUN;
```

Once the final model has been identified and saved, re-fitting this model without the penalty term gives estimates that have been “de-biased” from Reg-DIF. We describe this code now.

We must first create macro variables to use in the re-fitted model. This is shown in two steps. In the first code chunk below, we define all parameter macro variables (e.g., `&k1_1.`) to equal zero so that the previous effects of Reg-DIF (model fitting section) on these variables do not affect our results here. The second code chunk then creates macro variables for only those parameters that were included in the final model (with the lowest BIC). Effectively, these two code chunks allows us to reset the macro variables for re-fitting of the final model.

```
*CODE CHUNK 1: Make all macro parameters zero;
data zero;
 set start;
 call symput(Parameter,0);
run;

*CODE CHUNK 2: Create macro parameter variables only for those variables
included in model with minimum BIC;
data final;
 set final;
 call symput(Parameter,Parameter);
run;
```

Having saved the stating values and updated the macro variables, we may now re-fit the final model. Below **`PROC NLMIXED`** has nearly the same specifications as shown above. The only difference is that `GCONV` has a much stricter criterion: **`.00000000000000001`**. This is to ensure no local optima on the likelihood surface are found. We use the `final` model estimates as starting values and save the parameter estimates and fit statistics.

```
PROC NLMIXED DATA = long GCONV = .0000000000000001 QPOINTS = 15 MAXITER =
500 MAXFUNC = 3000;
PARMS / DATA = final
;

IF (item=1) THEN DO;
      gmu = (&v_1. + (&k1_1.*Age + &k2_1.*Gender + &k3_1.*Study)) + (&l_1. +
      (&w1_1.*Age + &w2_1.*Gender + &w3_1.*Study))*eta;
      IF (Response=1) THEN mu = 1/(1+exp(-gmu));
      ELSE mu = 1 - ( 1/(1+exp(-gmu)) );
      IF (mu > 1e-8) THEN ll = log(mu);
      ELSE ll = -1e100;
END;

…

IF (item=6) THEN DO;
      gmu = (&v_6. + (&k1_6.*Age + &k2_6.*Gender + &k3_6.*Study)) + (&l_6. +
      (&w1_6.*Age + &w2_6.*Gender + &w3_6.*Study))*eta;
      IF (Response=1) THEN mu = 1/(1+exp(-gmu));
      ELSE mu = 1 - ( 1/(1+exp(-gmu)) );
      IF (mu > 1e-8) THEN ll = log(mu);
      ELSE ll = -1e100;
END;
alpha = &g1.*Age + &g2.*Gender + &g3.*Study;
psi = exp(&b1.*Age + &b2.*Gender + &b3.*Study);
MODEL Response~GENERAL(ll);
RANDOM Eta~NORMAL(alpha,psi) SUBJECT=ID;
ODS OUTPUT ParameterEstimates=estimates;
RUN;
```

To obtain final model results, we may simply merge `estimates` and `fit`, print this data set, and if desired, save the model results to a CSV file.

```
*Merge results;
DATA final; MERGE fit estimates;
RUN;

*Print results;
PROC PRINT DATA=final;
RUN;

*Save results;
PROC EXPORT DATA=final
 OUTFILE= "C:\Users\Smith\final.csv"
 DBMS=csv
 REPLACE;
RUN;
```