# Practicum 2 (part B)

## Choice model specification and estimation (MixL)

### 1. Estimation and assessment of the Mixed Logit (MixL) model

From the **part A** of this practicum, you estimated a Latent Class Multinomial Logit (LC-MNL) model. From this model, you obtained insights into preferences by assuming that respondents are heterogeneous in their preferences and that there is a discrete number of preference classes into which respondents can be probabilistically allocated.

Now, you are going to estimate another model that accounts for preference heterogeneity, which is the Mixed Logit Model. In this model, preferences are assumed to be a continuous variable and each respondent is assumed to have her/his preference parameters. The sources of heterogeneity are non-observable, i.e., random. Observed sources of heterogeneity can be included through interactions in the utility function, as in the MNL model.

Before moving on, please follow the steps below to set up your environment correctly:

1. Go to Canvas > Modules > Week 5 > Tutorial 2
2. Download the syntax files "**practicum2_MixL.R**" and "**practicum2_MixL_postestimation.R**"
3. Go to download and find these two files
4. Move the syntax file to the folder created for the last week's practical
5. Open the syntax file "**practicum2_MixL.R**", and let's build an Mixed Logit model

Let us start from Apollo's basic structure that we introduced last week.

**How does the Apollo package work?**

In the Apollo package, all types of choice models use the same code structure. Each choice model is created by using the same building blocks. As the model complexity increases, components are added to the standard building blocks.


The standard building blocks to build a Mixed Logit model are:

- *Building block: Inputs*

In the "Inputs" block (see steps 1-9 in the syntax file), the set of information must be explicitly defined to set up the R-environment correctly for the Apollo package. As we will see, there is one additional component (step 9) in the "Inputs" block compared to the MNL syntax.

We start by setting up the R-environment.

- *Step 1*: clear the memory of the R-environment

```
19  ### Step 1: Clear memory
20  rm(list = ls())
```

This step is optional but highly recommended (!). Clearing the memory removes all saved variables, data frames, and objects in your R-environment. When you do not clear the environment and you are (re)running a (adjusted) syntax file, it could result in error messages.

- *Step 2*: Set a working directory for the R-environment

```
22  ### Step 2: Set working directory for R initialization
23  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

This step is optional but highly recommended (!). In this line, we instruct the R-environment to use the working directory (i.e., folder) in which the syntax file is stored. By doing so, we do not need to specify the full path to specific files, such as the dataset file.

- *Step 3 + 4*: Load and initialize the Apollo code

```
### Step 3: Load Apollo library
library(apollo)

### Step 4: Initialise Apollo code
apollo_initialise()
```

Here, we load and initialize the Apollo package into the R-environment.

- *Step 5*: Set the core controls

```
### Step 5: Set core controls
###     ATTENTION: Your inputs must be enclosed in quotes like "this"
apollo_control = list(
  # USER ACTION: Specify model name
  ##    Note: Change the model name for every model that you run
  modelName       = ,
  # USER ACTION: Provide model description
  ##    Note: Change the model description to reflect the current model
  modelDescr      = ,
  # USER ACTION: Specify the column with the respondent id
  indivID         = ,
  # USER ACTION: Set logical variable to activate estimation of random parameters
  mixing          = ,
  # Define number of cores used during estimation (used to speed up estimation time)
  nCores          = 5,
  # USER ACTION: Set path to the folder on your PC where the model results will be stored
  ##    Note: Use the "outputs" folder that was created by the pre-processing syntax
  outputDirectory =
)
```

As we have loaded and initialized the Apollo package in steps 3 and 4, we need to instruct Apollo how to call our model (= *modelName*), what our model is (= *modelDescr*), which column of our dataset contains the respondent identifiers (= *indivID*), and where to save the model results (*outputDirectory*).

Two control variables were added compared to the MNL. The control "*mixing*" is a logical (True/False) variable that define if random coefficient estimate is active or not. The control "*nCores*" sets the number of cores that will be used from the computer CPU. We fixed this number at 5 because it is most common number cores available in today's personal computers. It means that we are allowing Apollo to use the full processing power of the computer. This detail is informative of the computing effort required to estimate a Mixed Logit model.

***Action required!*** Go to your "**practicum5_MixL.R**" file, read the user actions, and complete the code in *step 5*. But do not run the syntax file yet (!).

- *Step 6*: Load the dataset into the R-environment

```
### Step 6: Load data
# Set path to directory on your PC where the dataset is stored
path_data = paste0("data",sep=.Platform$file.sep,"dataset.csv")
# Load dataset into global environment
database = read.csv(path_data, header=TRUE)
```

Because we have set the working directory (see step 2) in which the syntax file and dataset are stored, we do not need to specify the path to the dataset. We only need to check the name of the dataset whether it is correct or not (see purple names).

- *Step 7*: Initialize all parameters that need to be estimated in the choice model

```
### Step 7: Initialise all parameters that needs to be estimated in your Mixed Logit model
# USER ACTION: Define the (1) mu parameters that estimate the sample mean, and
#                         (2) sigma parameters that estimate the sample distribution
#              Please, complete the list with the parameters that are missing.
#              Provide names for each parameter following by assigning a
#              starting value.
apollo_beta=c(mu_asc_out         = 0,
              sigma_asc_out      = 0)
```

Note the differences between this code and the MNL. Only the *betas* were initialized in the MNL code, and the same vector of *betas* was applied to every respondent in the sample. Now, for every[1] taste coefficient to be estimated, we initialize

- *mu*, which is the average preference of the sample
- *sigma*, which identifies the dispersion of preferences (heterogeneity) around the mean

***Action required!*** Go to your "**practicum5_MixL.R**" file, read the user actions, and complete the code in **step 7**. But do not run the syntax file yet (!).

[1] For pedagogical purposes we use random parameters for every attribute. However, this is not mandatory. The researcher may have reasons to believe that some parameters should be fixed while others are random.

- *Step 8*: Define which parameters should kept fixed during the estimation

```
### Step 8:
##  USER ACTION: Complete the list with parameters (as initialised above) that
##               should be kept fixed during estimation (in quotes); if none, keep empty
apollo_fixed = c()
```

Remember that "*apollo_fixed*" is a list of parameters that should be kept fixed at their initial values.

***Action required!*** Go to your "**practicum5_MixL.R**" file, read the user actions, and complete the code in *step 8*. But do not run the syntax file yet (!).

- *Step 9*: Define controls for generating draws of the random coefficients

```
### Step 9: Set parameters for generating draws
# USER ACTION: Define the number of one random variable for each sigma in apollo_beta
# Use the command line interNormDraws

apollo_draws = list(
  interDrawsType = "mlhs",
  interNDraws    = 200,
  interUnifDraws = c(),
  interNormDraws = c(inter_1,...),
  intraDrawsType = "mlhs",
  intraNDraws    = 0,
  intraUnifDraws = c(),
  intraNormDraws = c()
)
```

Since the MixL identifies heterogeneity based on simulation, in this step the characteristics of simulation process to generate the random coefficients of the Mixed Logit model are specified. There are two blocks of controls in *step 9*.

- Controls starting with "*inter"* refers to heterogeneity across respondents. These are the relevant one to our goals. The control variables in this block are:
  - "*interDrawsType*": Apollo offers different methods to generate random variables. Keep this control as it is. Since the discussion of these methods are beyond the scope of the course, the interested student can find more information in the Apollo manual.
  - "*interNDraws*" controls the number of draws used in the simulation process. The stability of the Mixed Logit model depends on this number, and the larger it is the more robust the results are. However, as we increase this number, more time is needed for model estimation. Therefore, keep it as it is for the practical, and use at 500 draws for the workgroup project.
  - Apollo allows the choice of uniform ("*interUnifDraws*") or normal ("*interNormDraws*") random variables. In our course we use normal draws, and

a discussion of statistical distributions is beyond our scope. For each *sigma* create in **step 7**, you need to create one variable in "*interNormDraws*".

- Controls starting with "*intra"* refers to heterogeneity within respondents, i.e., one respondent taste heterogeneity across choice occasions. This idea is interesting behaviorally (can you imagine why?) but also beyond our scope.

***Action required!*** Go to your "**practicum5_MixL.R**" file, read the user actions, and complete the code in *step 9*. But do not run the syntax file yet (!).

- *Step 10*: Create random parameters

```
### Step 10: Create random parameters
# USER ACTION: Write every random coefficient function
# If necessary check the lecture slides
apollo_randCoeff = function(apollo_beta, apollo_inputs){
  randcoeff = list()

  randcoeff[["asc_out"]] = mu_asc_out + sigma_asc_out * inter_1
  randcoeff[["b_eff"]] =
  randcoeff[["b_fneg"]] =
  randcoeff[["b_freq_2yr"]] =
  randcoeff[["b_freq_3yr"]] =
  randcoeff[["b_wdiag_2wks"]] =
  randcoeff[["b_wdiag_3wks"]] =
  randcoeff[["b_wfup_4wks"]] =
  randcoeff[["b_wfup_8wks"]] =

  return(randcoeff)
}
```

Comparing to the MNL model, note that the random parameters are specified for each *asc*/*beta*. Here it is possible to see that $b_k = mu_k + sigma_k$*$inter\_k$ which is a distribution.

***Action required!*** Go to your "**practicum5_MixL.R**" file, read the user actions, and complete the code in *step 10*. But do not run the syntax file yet (!).

- *Building block: Validate inputs*

The final step in preparing the code and data for model estimation is to make a call to:

- *Step 11*: Validation of the inputs

```
105  ### Step 11: Checkpoint for model inputs
106  apollo_inputs = apollo_validateInputs()
```

The function runs several checks and produces a consolidated list of model inputs. This function takes no arguments but looks at the R-environment for the various inputs required to build and estimate the model.

- *Building block: Likelihood function*

At this point, you have initialized all the essentials for building the Mixed Logit model. Now, it is time to the final specifications. So, let's go to *step 12*.

- *Step 12*: Define the choice model and likelihood function

```
### Step 12: Define model and likelihood function
apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

  ### Attach dataset inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

  ### Create list of choice probabilities P
  P = list()

  ### List of utility functions: these must use the same names as in mnl_settings (see below), order is irrelevant
  V = list()

  # USER ACTION: Define utility function for alternative 1
  # Code "effectiveness" and "risk false negative" attributes as numerical variables
  V[["ALT1"]]  =

  # USER ACTION: Define utility function for alternative 2
  # Code "effectiveness" and "risk false negative" attributes as numerical variables
  V[["ALT2"]]  =

  # USER ACTION: Utility function for alternative 3 (i.e., opt-out)
  V[["ALT3"]] =

  ### Define settings for MNL model component
  mnl_settings = list(
    # USER ACTION: Attach utility function to the choice alternative in your dataset
    alternatives  = c(ALT1=, ALT2=, ALT3=),
    # USER ACTION: Define which alternatives are "available" in each choice task
    #              In our study, all alternatives are "available"
    avail         = 1,
    # USER ACTION: Specify the column containing the chosen alternative
    choiceVar     = ,
    # USER ACTION: Attach list of utility functions
    utilities     =
  )

  ### Compute choice probabilities using MNL model
  #### functionality="estimate" as the parameters will be updated for estimating the MNL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality)

  ### Take product across observations for same individual
  ### (i.e., considering the panel structure of the data)
  P = apollo_panelProd(P, apollo_inputs, functionality)

  ## Average across inter-individual draws
  P = apollo_avgInterDraws(P, apollo_inputs, functionality)

  ### Prepare and return outputs of function
  P = apollo_prepareProb(P, apollo_inputs, functionality)
  return(P)
}
```

The output of the *apollo_probabilities* function is the choice probabilities. To compute the choice probabilities, we need to define the utility functions and set the correct settings for the MNL model.

***Action required!*** Go to your "**practicum5_MixL.R"** file, read the user actions, and complete the code in step 12. But do not run the syntax file yet (!).

***Tip***: the utility is exactly the same as in the MNL. However, the number of estimated parameters is different. Do you understand why?

- *Building block: Estimation*

When you have completed *step 12*, you have initialized all essentials and built your MixL model. Now, it is time to start the estimation procedure.

- Step 13: Model estimation

```
### Step 13: Model estimation
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs)
```

By running *step 13*, the estimation procedure will be started by Apollo. There are no actions required. It will be all handled automatically for you.

- *Building block: Output*

The last building block is to instruct Apollo to print the model output and save the results in the folder that we have specified in *step 5* (see "*outputDirectory*").

- *Step 14*: Print model output with two-sided p-values

```
### Step 14: Print model output with two-sided p-values
#### Note: if one-sided p-values are needed, set "printPVal=1" (p-values are not reported if set to "0")
modelOutput_setting=list(printPVal=2)
apollo_modelOutput(model, modelOutput_setting)
```

After completing the estimation procedure, we instruct Apollo on how to process the model results. For now, you do not need to adjust anything. However, in some situations, you might want additional information. The Apollo manual explains what information you can extract and how you can do this (see here, page 37).

Once you have completed all 14 steps, go to the "**practicum5_MixL.R"** file and run the full syntax. The model results will be shown in your console, which should be used to answer the questions below.

**Question 2.1** Briefly describe how the mean (*mu*) coefficients can be interpreted. Compare the interpretation of the mean MIXL parameter estimates the interpretation of the *betas* from the Multinomial Logit (MNL) model?

| *Answer*: |
|---|

**Question 2.2** How can you interpret the parameters and standard errors of the sigma parameters?

| *Answer:* |
|---|

**Question 2.3** In which attributes/attribute levels is significant random heterogeneity observed?

| *Answer*: |
|---|

**Question 2.4** How can you interpret the mean (*mu*) and the standard deviation (*sigma*) of the CRC screening frequency every 2 years?

| *Answer*: |
|---|