European University Viadrina
Department of Information and Operations Management
Business Analytics

Advanced Business Analytics 2019/2020

Comparison of mlogit and apollo packages for discrete choice modelling

Kirill Degtaryev, André Didriksen

Abstract

We examine Apollo, a new R package for choice modelling. We compare it to a popular alternative, mlogit, using the examples provided in an exercise set accompanying the mlogit package. Our analysis encompasses applications of standard multinomial models, nested logit models and mixed logit models. We describe how models are estimated in Apollo and mlogit from the user's perspective and evaluate the usability and versatility of both packages. We find that the versatility of Apollo is superior. Furthermore, we find that the user experience of working with Apollo and mlogit is very different. Whereas mlogit features extensive use of default options and automatic declaration of variables, Apollo offers no automatic correction to the code and requires the user to specify every detail of the model. The implication of this is that mlogit lends itself better to cases in which code needs to be reused to create models with minor differences with similar datasets, while Apollo has the advantage of increased transparency and decreased risk of specification error.

1. Introduction

This paper will discuss choice modelling with R. Specifically, and we will discuss and evaluate a new choice modelling package for R, called Apollo. We will compare and contrast this package with a popular existing alternative, named mlogit.

1.1. Structure

The paper is outlined as follows: we provide a brief background explanation of choice modelling with R, followed by a description of the focal points of our examination. We will provide a summary of the basic elements of the code, including its general structure and components. This is followed by a description of how adjusting the mlogit code for Apollo was done and an evaluation of the differences between the two packages.

1.2. Introduction to choice modelling with R

Discrete choice models are tools that aim to use data to make predictions about the way people make decisions. Prominent discrete choice models include (among others) the multinomial logit models, including the nested logit model and mixed logit model. Fundamentally, these models are designed to take information about explanatory variables such as income, price and age, and use them to predict the probability that individuals will choose, for instance, one product over the other.

Because of the inherent complexity and the large amounts of data going into such a model, the calculation is usually done with the help of a computer. There is a number of available software options that vary both in terms of user-friendliness, versatility and affordability. One widely used alternative is to use one of several packages for the statistical programming software R.

R is an open-source language and environment for statistical computing and graphics (The R Foundation, 2020). Because users are free to publish packages that expand the functionality of R. In this way, R can be seen as a platform for people to make new techniques for statistical analysis available to the public. As it is free, encourages the addition and sharing of new functions and techniques and has become one of the most widely used programming languages in the world, it is natural that practitioners of statistical analysis have also developed packages for discrete choice models.

One of the current popular options for discrete choice models is the package called mlogit, based on Kenneth Train (2009) Discrete Choice Methods with Simulation (Croissant, 2019). Its main contribution lies in making it possible to estimate multinomial logit-, nested logit- and mixed logit models in R. These models build on each other. Whereas the multinomial logit model already provides a model for utility in discrete choice, its validity is based on several restrictions. The mixed logit model, on the other hand, can be used to approximate utility in any choice model (McFadden & Train, 2000).

The focus of this paper will be a new package under development by the Choice Modelling Centre at the University of Leeds, named Apollo. Basically, everything mlogit can do, Apollo aims to do it as well, only more organized and with more options for customization.

With this paper, we aim to compare and contrast the capabilities, and practical reality of using, Apollo and mlogit. Of particular interest to us are differences in the complexity of the code required to write equivalent models. In addition, we pay special attention to the perspective of instructing inexperienced to intermediate users of R with basic knowledge of choice modelling to the packages.

This paper is aimed at readers who already have some familiarity with using mlogit to estimate multinomial logit models. After reading this paper, you will have learned about what to expect when estimating a model in Apollo contra estimating it in mlogit. You will also have learned about our recommendations for the advantages and disadvantages of choosing one package over the other, and to whom we recommend using Apollo over mlogit.

2. Basis of comparison

Since mlogit and Apollo are used to estimate the same model and given the same specification will yield identical results, the code will, to some degree follow the same basic structure in both packages. However, a key difference between Apollo and mlogit is that mlogit 'assists' the user by extensive use of default options and automatic specification, whereas Apollo offers more flexibility then mlogit. In other words, mlogit assumes that most users will create the standard format of the model that the package was designed for most of the time. Apollo, on the other hand, assumes nothing.

Apollo aims to offer choice modelling with a modular structure (Hess & Palma, 2019). The code for calling a choice model is supposed to have a proverbial skeleton that is the same for every model. The user is able to "flesh out" the skeleton to create a unique model, hopefully perfectly suited what the model is supposed to do and the available data. This skeleton consists of initialization, data preparation, parameter definition, model definition and model estimation.

2.1. Case study

In our paper, we will look at the practical execution of estimating a standard multinomial logit model, as well as nested logit models and mixed logit models, in Apollo and mlogit. We use the problem set from Kenneth Train and Yves Croissant's mlogit examples (Train & Croissant, 2012). Equation 1 contains an example of the generalized model we are trying to estimate.

$$V_{n,i} = \beta_i + \beta_{ic} \cdot ic + \beta_{oc} \cdot oc \tag{1}$$

Equation 1 is the generalized notation of the model estimated in Exercise 4 of the multinomial logit exercises from the mlogit problem set (Train & Croissant, 2012). $V_{n,i}$ is the discrete part of utility of choice i for individual n. β_i refers to the intercept of alternative i, $\beta_{ic} \cdot ic$ is the effect of installation cost on utility, $\beta_{oc} \cdot oc$ is the effect of operation costs on utility.

The notation is reflective of how the mlogit set is designed to be used. mlogit requires you to estimate the multinomial logit model in a generalized way. In contrast, Apollo requires you to declare a separate function of discrete utility for each alternative (see Figure 1). On the one hand, this means that Apollo requires you to at the very least declare five more variables (if there are five alternatives). On the other hand, this means that if you want to specify a model in which the utility of one particular alternative is not affected by its installation costs, mlogit would require you to remove the installation costs from the variable set.

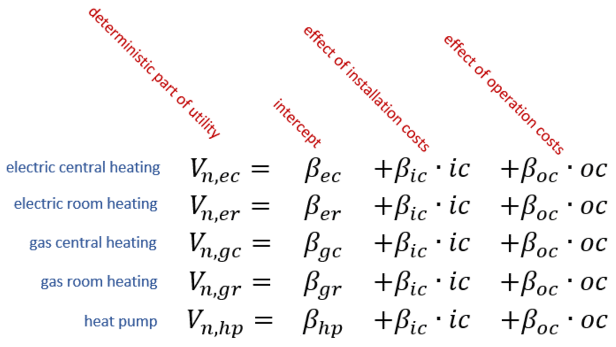

deterministic part of utility
intercept
effect of installation costs
effect of operation costs
electric central heating
electric room heating
gas central heating
gas room heating
heat pump
$V_{n,ec} = \beta_{ec} + \beta_{ic} \cdot ic + \beta_{oc} \cdot oc$
$V_{n,er} = \beta_{er} + \beta_{ic} \cdot ic + \beta_{oc} \cdot oc$
$V_{n,gc} = \beta_{gc} + \beta_{ic} \cdot ic + \beta_{oc} \cdot oc$
$V_{n,gr} = \beta_{gr} + \beta_{ic} \cdot ic + \beta_{oc} \cdot oc$
$V_{n,hp} = \beta_{hp} + \beta_{ic} \cdot ic + \beta_{oc} \cdot oc$

Figure 1: Model specification of multinomial logit exercise 4 in Apollo.

With this in mind, let us look at the general structure of a choice model in R.

2.2. General code structure and components of choice models in R

The first step of using any choice modelling functionality in R is to load the respective package. The user must then load the data that they will use for the estimation. The data usually requires additional preparation to be compatible with the specific package. Some packages provide functions that allow the user to easily manipulate the data so that it can be used for choice modelling. Other packages fail to supply this, in which case the user must usually rely on other available options for data manipulation, such as the *tidyr* package.

After preparing the dataset, model parameters must be defined. R packages usually use default options and automatic name generation to help the user with this stage, since defining model parameters is usually a straightforward task that without automatic help can become time-consuming. The user must then specify the model to such a degree that, at the bare minimum, the software knows what the dependent variable is and what kind of relationship is assumed with the independent variables. Choice modelling software then typically requires the user to add any additional options and then run the estimation function.

2.3. Basis of comparison

In our evaluation, we will carry out exercises for multinomial logit models, nested logit models and mixed logit models from the mlogit problem set (Train & Croissant, 2012). Our comparison will mainly focus on the following qualities: usability, meaning how easy the package is to use, what level of proficiency in using R the user needs to have in order to apply the package to their own problems; and versatility, meaning the degree to which the package allows user to customize their own models.

3. Implement Specifications

3.1. *Loading Data*

Both packages require preprocessing and some common special rules for loading data to properly construct a data model. The data should contain a column of alternatives, choices that every individual have made and the determinants of these choices being variables that can be alternative specific or purely individual specific. It is also possible for both packages to work with data containing several choices made by one individual, where each row of the data set shows data related to the specific choice of the individual, but then the dataset should also have a column indicating the sequential number of the participant.

In our case, datasets were taken from Kenneth Train and Yves Croissant's paper (Train & Croissant, 2012) fully respect rules that were mentioned before. For example, after calling a function `data( "Heating", package = "mlogit")` we get the data, where indicates an alternative by the column "depvar", which have been chosen by a respondent, sequential number of a case by "idcase", the determinants by other columns. For the mlogit package, a dataset in this format, where each row is an observation, is not acceptable for further modelling and requires reshaping. This shape of data is called the *wide* shape (Croissant, 2019), see Table 1.

Table 1 An example of the wide shape dataset.

idcase	depvar	ic.gc	ic.gr	ic.ec	ic.er	ic.hp	oc.gc	oc.gr	oc.ec	oc.er	oc.hp	income	agehed
1	gc	866	962.64	859.9	995.76	1135.5	199.69	151.72	553.34	505.6	237.88	7	25
2	gc	727.93	758.89	796.82	894.69	968.9	168.66	168.66	520.24	486.49	199.19	5	60
3	gc	599.48	783.05	719.86	900.11	1048.3	165.58	137.8	439.06	404.74	171.47	4	65
4	er	835.17	793.06	761.25	831.04	1048.7	180.88	147.14	483	425.22	222.95	2	50
5	er	755.59	846.29	858.86	985.64	883.05	174.91	138.9	404.41	389.52	178.49	2	25
6	gc	666.11	841.71	693.74	862.56	859.18	135.67	140.97	398.22	371.04	209.27	6	65
7	gc	670.1	941.25	633.63	952.3	1086.8	191.84	147.57	478.36	445.97	236.99	4	35

As we can see from the Table 1, the variables “ic” and “oc” are alternative specific, whereas “income” and “agehed” are individual specific as theirs values do not change across alternatives in each observation.

Another type of data shape commonly used by packages for discrete choice modelling is the *long* shape, where each row is an alternative (Croissant, 2019), see Table 2.

Table 2 An example of the long shape dataset

	idcase	depvar	income	agehed	rooms	region	alt	ic	oc	chid
1.ec	1	FALSE	7	25	6	ncostl	ec	859.9	553.34	1
1.er	1	FALSE	7	25	6	ncostl	er	995.76	505.6	1
1.gc	1	TRUE	7	25	6	ncostl	gc	866	199.69	1
1.gr	1	FALSE	7	25	6	ncostl	gr	962.64	151.72	1
1.hp	1	FALSE	7	25	6	ncostl	hp	1135.5	237.88	1
2.ec	2	FALSE	5	60	5	scostl	ec	796.82	520.24	2

Mlogit can deal with these formats using the `mlogit.data` function, which takes as first argument a **data.frame** and returns a **data.frame** in “long” format with some information about the structure of the data. For the “Heating” data, we would use:

```
H <- mlogit.data(Heating, shape = "wide", choice = "depvar", varying = c(3:12))
```

Snippet 1 R syntax for mlogit.data

The mandatory arguments are **choice**, which is the variable that indicates which alternative is selected in each observation, the shape of the **data.frame**, and **varying** if there are existing alternative specific variables. The output of this function is mlogit.data type variable, which is very similar to the data.frame type in *long* shape. First seven rows of the variable “H” can be seen in Table 2. For preprocessing datasets stored in long format, a user should also set up an argument at least for alt.var or for alt.levels. Using this function makes the user experience of this package more friendly and make it possible for the person, who is not much experienced in R programming, to construct models with a simple data structure.

The apollo package, unlike the mlogit package requires the data to be in the wide format. It might take a lot of time to adapt dataset, which was given in a *long* shape since apollo package does not have its function to do such reshaping. Also, the data should be named in the environment exactly as “database”, so that function `apollo_validateInputs()` find this variable. Another mandatory requirement for a dataset is the availability of columns of choice for each observation coded in numeric form and “ID” column indicating a sequential number. The strict name of the dataset for modelling could cause collisions during working with several scripts, so it is recommended to include the command `rm(list = ls())` in order to delete all variables in the environment before running a script. The corresponding code to the Snippet 1 in the apollo looks like:

```
data("Heating", package = "mlogit")
database <- data.frame(Heating)
# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]
```

Snippet 2 R syntax for loading and preprocessing data for the apollo package

Both packages require loading data into a data frame of a specific structure and other rules about the content of the dataset. However, while the mlogit package has built-in function responsible for reshaping data into *long* shape appropriate for further mlogit modelling, the Apollo package does not have it. It requires more experience in data preprocessing for apollo users than for mlogit one. The **mlogit.data()** function is not able to preprocess data in all cases, such as dealing with unstructured datasets, but it can reduce some routine work for analysts. On the contrary, apollo's manual approach leads to a better understanding of the task during preprocessing data, and this can be valuable in further modelling.

3.2. *Model parameters*

The next step in modelling after loading and preprocessing data is setting of model parameters. Coefficients might be found during evaluation of a model or fixed to a specific value. The process of evaluation is iterative, and the user could define start values of unfixed coefficients so that an algorithm converges more quickly. In the case of mixed logit modelling, random parameters should also be set, as well as the method in which they will be generated.

In the **mlogit** package for all these purposes, mlogit() function is responsible. This function is used for evaluation of all kinds of models that are supported by the mlogit package. There are two mandatory arguments for these functions: *formula* and *data* (Croissant, 2019). *Formula* is a symbolic descriptionof the model to be estimated, and *data* is the name of a variable where is stored dataset for modelling.

Mlogit works with three different types of variables and coefficients (Croissant, 2012):

- alternative specific variables x_{ij} with a generic coefficients β
- individual specific variables z_i with alternative specific coefficients γ_j
- alternative specific variables ω_{ij} with an alternative specific coefficients δ_j

Where i is a number of an observation and j is a number of an alternative.

For an illustration how to specify utility function in the **mlogit** package, let's use the most general example, that would be included all three types:

```
m <- mlogit(depvar ~ oc | income | ic, H, reflevel = "ec")
```

Snippet 3 R syntax for defining utility's function specification

In this example variables, *oc* is alternative specific with a generic coefficient, *income* is individual specific with alternative specific coefficients, and *ic* is an alternative specific variable with alternative specific coefficients.

Also, it may be necessary to add some additional parameters in order to estimate a model. In MNL models, analysts often set a reference level of a model or, put differently, which alternative's intercept would be fixed to zero. In **mlogit**, it could be done easily by adding an argument to the mlogit function `ref.level` with the name of an alternative after the sign of assignment. Another useful argument is `na.action`, which receives a function that indicates what should happen when the data contains NA, e.g. `na.action = na.omit`.

For **nested logit models**, a user should also add an argument responsible for the setting of a nested structure – `nests`. It should be a named list of characters vectors, each name representing a nest, the corresponding vector being the set of alternatives that belong to this nest. For example, for the first exercise from the section of Nested logit models from the mlogit problem set (Train & Croissant, 2012) nest structure looked like:

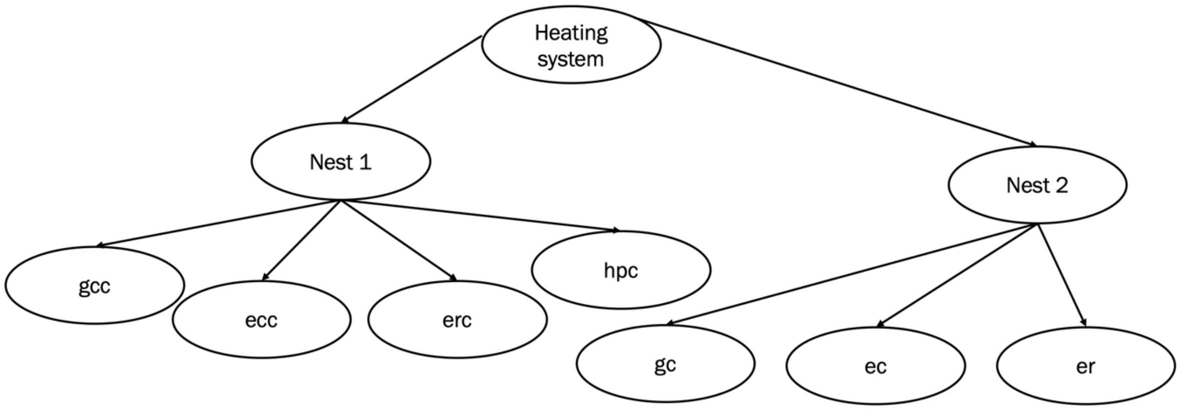


Figure 2 An illustration of the nest structure

```
nl <- mlogit(depvar ~ ich + och +icca + occa + inc.room + inc.cooling + int.cooling | 0,
HC,nests = list(cooling = c('gcc','ecc','erc','hpc'), other = c('gc', 'ec', 'er')),
un.nest.el = TRUE)
```

Snippet 4 R syntax for defining nested logit model in the mlogit package

In nested logit models, it is also important to decide the elasticity of every nest in advance. In the case that the analyst decides to evaluate a model with the same elasticity for all nests, this would mean that a change in the variable of an alternative has the same effect on all other alternatives. In the mlogit package, the argument responsible for this is `un.nest.el`, which receive a logical value *TRUE* in case of equal elasticity among all nests, or *FALSE* otherwise.

Mixed logit modelling also requires to include additional parameters related to random coefficients, that are describedby specifying the distribution from which coefficients would be generated and the number of draws of pseudo-random numbers. Furthermore, a method for generating these draws should be indicated. The `rpar` argument is a named vector whose names are the random parameters and values of the distribution: ‘n’ for normal, ‘l’ for log-normal, ‘t’ for truncated normal, ‘u’ for uniform. The number of draws is set in the argument `R`. The way in which these pseudo-random numbers would be generated described in the argument `halton`, which is relevant if rpar in not NULL. In case of a NA value, some default values are used for the prime of the sequence (actually, the primes are used in the order) and for the number of elements dropped. Otherwise, halton should be a list with elements prime (the primes used) and drop (the number of elements dropped). More details can be read in Train (2009). An example of the mixed logit model in the mlogit package from the first exercise from a Mixed logit section in the mlogit problem set (Train & Croissant, 2012):

```
Elec.mxl <- mlogit(choice ~ pf + cl + loc + wk + tod + seas | 0, Electr, rpar=c(pf = 'n',
cl = 'n', loc = 'n', wk = 'n',
tod = 'n', seas = 'n'), R = 100, halton = NA, panel = TRUE)
```

Snippet 5 R syntax for mixed logit model in the mlogit package

In the **apollo** package, unlike the mlogit package, there is no only one function that would be responsible for defining model parameters. Instead of this, the user should define several variables with specific reserved names and a fucntion. It is done for flexibility, that user could easily invoke his own function into the package on the one hand, and additionally leads to transparency, so that the analyst can explain in detail how the algorithm works.

A set of coefficient with their start values that would be founded during the work of an algorithm is set in **apollo_beta** variable. Those, of them, that should be fixed to their start values, also should be indicated in a list **apollo_fixed**. Specifications of the utility functions are set inside of the `apollo_probabilities` function, that returns the probabilities of the model using the log-likelihood approach. The reference level in apollo is set differently: in case of alternative specific variables with generic coefficients in the model, the corresponding intercept should be excluded or their value fixed to zero, in case of individual specific variables with alternative specific coefficients, corresponding coefficients should be fixed to zero. The code for the same specification that was in the snippet 3 in the apollo package looks like:

```
### Vector of parameters, including any that are kept fixed in estimation

apollo_beta <- c(oc_coef = 0,
                 ic_coef.gc = 0,
                 ic_coef.er = 0,
                 ic_coef.gr = 0,
                 ic_coef.hp = 0,
                 ic_coef.ec = 0,
                 income_coef_gc = 0,
                 income_coef_er = 0,
                 income_coef_gr = 0,
                 income_coef_hp = 0,
                 income_coef_ec = 0,
                 c_ec = 0,
                 c_er = 0,
                 c_gc = 0,
                 c_gr = 0,
                 c_hp = 0)


apollo_fixed = c("c_ec", "income_coef_ec")
```

Snippet 6 R syntax for defining a set of parameters in the apollo package

Specifications of utility functions are set inside the `apollo_probabilities` function:

```
apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

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

  ### List of utilities: these must use the same names as in mnl_settings, order is
irrelevant
  V <- list()

  V[['ec']]  <-  c_ec + (ic_coef.ec * ic.ec + income_coef_ec * income + oc_coef * oc.ec)
  V[['er']]  <-  c_er + (ic_coef.er * ic.er + income_coef_er * income + oc_coef * oc.er)
  V[['gc']]  <-  c_gc + (ic_coef.gc * ic.gc + income_coef_gc * income + oc_coef * oc.gc)
  V[['gr']]  <-  c_gr + (ic_coef.gr * ic.gr + income_coef_gr * income + oc_coef * oc.gr)
  V[['hp']]  <-  c_hp + (ic_coef.hp * ic.hp + income_coef_hp * income + oc_coef * oc.hp)
```

Snippet 7 R syntax for defining utility functions in the apollo package

For **nested logit modelling** a user should also include a descriptionof the nest structure inside of the `apollo_probabilities` function by adding lists `nlNests` and `nlStructure` and a new parameter into `apollo_beta` – *lambda*. For receiving the same model in the apollo package as in mlogit when the value of an argument `un.nest.el` was set to *TRUE*, in the list `nlNests` all nests should be assigned to the same *lambda*. The mandatory requirement for this list is that the name of the one nest should be “root”. The corresponding code to the snippet 4 in the apollo package looks like:

```
### Specify nests for NL model
  nlNests = list(root=1, nest1=lambda, nest2=lambda)

  ### Specify tree structure for NL model
  nlStructure= list()
  nlStructure[["root"]]    = c("nest1","nest2")
  nlStructure[["nest1"]]    = c("gcc","ecc", "erc", "hpc")
  nlStructure[["nest2"]]    = c("gc","ec","er")
```

Snippet 8 R syntax for defining nested structure in the apollo package

Mixed logit models are defined in the Apollo package by introducing the list `apollo_draws`, which indicates how the draws are generated, and new function `apollo_randCoeff`, which returns randomly generated coefficients that would be used in the list of utility functions, `V`. In this function should be written the way how random coefficients is received using a given standard deviation and expected value. For example, the normal distribution could be specified as:

$$X = \mu + \sigma Z \qquad (2)$$

, where μ – expected value, σ – standard deviation, Z - standard normal random variable.

Log-normal distribution:

$$X = e^{\mu + \sigma Z} \qquad (3)$$

The corresponded code to the *snippet 5* in the Apollo that specifies distributions would be:

```
### Create random parameters
apollo_randCoeff = function(apollo_beta, apollo_inputs){
  randcoeff = list()

  randcoeff[["c_pf"]] = mu_pf + sd_pf * draws_pf
  randcoeff[["c_cl"]] = mu_cl + sd_cl * draws_cl
  randcoeff[["c_loc"]] = mu_loc + sd_loc * draws_loc
  randcoeff[["c_wk"]] = mu_wk + sd_wk * draws_wk
  randcoeff[["c_tod"]] = mu_tod + sd_tod * draws_tod
  randcoeff[["c_seas"]] = mu_seas + sd_seas * draws_seas

  return(randcoeff)
}
```

Snippet 9 R syntax for defining distribution in the apollo package

The list `apollo_draws` is a set of settings that are required for mixed logit modelling, where you can specify inter draws that differ only across observations as in the mlogit package, but also intra draws that differ across alternatives or use them both at the same time. This approach allows for generating values from a wide variety of distributions.

The Apollo package suggests seven pre-defined types of draws (Hess et al., 2019):

- **pmc:** for pseudo-Monte Carlo draws
- **halton:** for Halton draws Halton
- **mlhs:** for MLHS draws
- **sobol:** for Sobol draws
- **sobolOwen:** for Sobol draws with Owen scrambling
- **sobolFaureTezuka:** for Sobol draws with Faure-Tezuka scrambling
- **sobolOwenFaureTezuka:** for Sobol draws with both Owen and Faure-Tezuka scrambling

The code for the mixed logit model that specifies draws in Apollo looks like:

```
### Set parameters for generating draws
apollo_draws = list(
  interDrawsType = "pmc",
  interNDraws    = 100,
  interUnifDraws = c(),
  interNormDraws = c("draws_pf","draws_cl","draws_loc","draws_wk", "draws_tod",
"draws_seas"),
  intraDrawsType = "pmc",
  intraNDraws    = 0,
  intraUnifDraws = c(),
  intraNormDraws = c()
)
```

Snippet 10 R syntax for settings of draws in the apollo package

The mlogit and Apollo packages allow users to estimate trivial models in general easily, but when it comes to modelling tasks with a complicated specification using non-built-in functions, the Apollo package has advantages due to its modular structure. Also, explicitly defining a model step-by-step gives a better understanding of the details. At the same time, it might seem like routine work to set a big number of the same settings every time before estimating the new model. This could also increase the risk of making a mistake.

3.3 Model definition

The last part of modelling before the estimation is defining the settings that are responsible for getting the algorithm working correctly. With these parameters, the algorithm is able to determine whether techniques for multinomial logit models or another model should be used. Also, these settings could indicate a way of implementing an algorithm.

This topic is not related to the mlogit package since the algorithm is defined unambiguously by parameters. For instance, if the `mlogit` function does not include any other arguments besides formula and data arguments, then a multinomial logit model would be estimated. Arguments are also included related to the nest structure (in the case of a nested logit model) or the random coefficients (in the case of a mixed logit model).

In the Apollo package settings that are responsible for controlling the running of the code are located in the list `apollo_control`. The most important settings are indicated below, while others could be found in the Apollo documentation (Hess et al., 2019):

- **mixing**: Boolean. TRUE for models that include random parameters. It is a mandatory parameter in case of a constructing Mixed logit model

- **nCores**: Numeric>0. Number of threads (processors) to use in estimation of the model.
- **seed**: Numeric. Seed for random number generation. Could be useful for comparing results with other algorithms.
- **panelData**: Boolean. TRUE if using panel data.

The corresponding code for the mixed logit models:

```
### Set core controls
apollo_control = list(
  modelName ="Mixed logit",
  modelDescr ="Mixed logit model1",
  indivID   ="id",
  mixing    = TRUE,
  nCores = 4
)
```

Snippet 11 R syntax of apollo_control settings

Settings that contain details of the algorithm's operation are included in the list model_settings, where the *model* is substituted by the name of the type of model that is to be estimated, for example:

- **mnl**: multinomial logit model and mixed logit model
- **nl**: nested logit model
- **cnl**: cross nested logit model
- **ol**: ordered logit model

There might be some different setting for each type of model, but common settings are:

- **alternatives**: Named numeric vector. Names of alternatives and their corresponding value in choiceVar.
- **avail**: Named list of numeric vectors or scalars. Availabilities of alternatives, one element per **alternative**. Names of elements must match those in alternatives. Values can be 0 or 1.
- **choiceVar**: Numeric vector. Contains choices for all observations. It will usually be a column from the database. Values are defined in alternatives.
- **V**: Named list of deterministic utilities. Utilities of the alternatives. Names of elements must match those in alternatives.

The code for the mixed logit model looks like:

```
### Define settings for MNL model component
  mnl_settings = list(
    alternatives  = c(alt1=1, alt2=2, alt3=3, alt4=4),
    avail         = list(alt1=1, alt2=1, alt3=1, alt4=1),
    choiceVar     = choice,
    V             = V
  )
```

Snippet 12 R syntax of mnl_settings for the mixed logit model

As we can see, the Apollo package suggests many options, and most of them should be set; unlike the mlogit package, where mandatory settings defined automatically by model parameters' arguments. It might seem better in terms of usability, but some details of implementing the model might be hidden with this approach and lead to a misunderstanding of how the algorithm would be working.

3.4 Estimation and model output

Estimation of the mlogit model happens after calling the `mlogit` function, and then output in the console mightbe displayed if the **print.level** argument is set to any value from 1, 2. By default, this argument is set to 0, so nothing id displayed during the evaluation of the model. At level 1, the user could see a log-likelihood value on each iteration as well as chi-square statistic value. At level 2: additionally, values of coefficients and the gradient for each parameter. Also, this function could receive arguments concerning computational methods:

- **method**: the method used, one of 'nr' for Newton-Ralphson, 'bhhh' for Berndt-HausmanHall-Hall and 'bfgs',
- **start:** the initial value of the vector of coefficients
- **iterlim:** the maximum number of iterations

To see an output of the model, the user could call the `summary` function:

Table 3 mlogit output of the multinomial logit model

	Dependent variable:
	depvar
ec:(intercept)	1.659***
	(0.448)
er:(intercept)	1.853***
	(0.362)
gc:(intercept)	1.711***
	(0.227)
gr:(intercept)	0.308
	(0.207)
ic	-0.002**
	(0.001)
oc	-0.007***
	(0.002)
Observations	900
R^2	0.014
Log Likelihood	-1,008.229
LR Test	27.990*** (df = 6)
Note:	*p<0.1; **p<0.05; ***p<0.01

The package does not have a built-in function to save this output into some file, but the format of the output is standardized and can be handled by functions from other packages without problems.

The Apollo package has the separate functions for estimating a model using the likelihood function – apollo_estimate, which receives arguments **apollo_beta**, **apollo_fixed, apollo_inputs** and a function `apollo_probabilities`. An optional argument in this function is `estimate_settings`, which is a list containing controlling options for the estimation process:

- **estimationRoutine**: Character. Estimation method. Can take values "bfgs", "bhhh", or "nr". Used only if apollo_control$HB is FALSE. Default is "bfgs"
- **maxIterations**: Numeric. Maximum number of iterations of the estimation routine before stopping. Default is 200.
- **printLevel**: Higher values render more verbose
- outputs. Can take values 0, 1, 2 or 3.
- **silent**: Boolean. If TRUE, no information is printed to the console during estimation. Default is FALSE

For displaying the results of computations, the apollo has a dedicated function called `apollo_modelOutput`. This function receives a variable assigned to results of the `apollo_estimate` function and has its own settings:

- **printClassical**: Boolean. TRUE for printing classical standard errors. TRUE by default.
- **printPVal**: Boolean. TRUE for printing p-values. FALSE by default.
- **printT1**: Boolean. If TRUE, t-test for H0: apollo_beta=1 are printed. FALSE by default

The results could be saved into the file with `apollo_saveOutput`, which receives the same arguments as the `apollo_estimate` function, but has an additional one related to the files, that this function would produce:

- **saveEst**: Boolean. TRUE for saving estimated parameters and standard errors to a CSV file. TRUE by default.
- **saveCov**: Boolean. TRUE for saving estimated correlation matrix to a CSV file. TRUE by default.
- **saveCorr**: Boolean. TRUE for saving estimated correlation matrix to a CSV file. TRUE by default.
- **saveModelObject**: Boolean. TRUE to save the R model object to a file (use apollo_loadModel to load it to memory). TRUE by default.
- **writeF12**: Boolean. TRUE for writing results into an F12 file (ALOGIT format). FALSE by default.

The code that estimates, displays and saves the output into a file in the apollo package looks like:

```
model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
                        estimate_settings = list(estimationRoutine = "nr", print.Level =
0))


apollo_modelOutput(model, modelOutput_settings = list(printPVal = TRUE, printT1 = TRUE))

apollo_saveOutput(model)
```

Snippet 13 R syntax for estimating and saving the output in the apollo package

And the output for the same task as in the mlogit package (Table 3) would be:

Table 4 apollo output for the multinomiallogit model

	Estimate	Std.err.	t.ratio(0)	p-val(0)	t.ratio(1)	p-val(1)	Rob.std.err.	Rob.t.ratio(0)	Rob.p-val(0)	Rob.t.ratio(1)	Rob.p-val(1)
ic_coef	-0.002	0.001	-2.470	0.014	-1,613.150	0	0.001	-2.530	0.012	-1,650.680	0
oc_coef	-0.007	0.002	-4.500	0	-647.970	0	0.002	-4.760	0	-685.760	0
c_ec	1.659	0.448	3.700	0	1.470	0.142	0.440	3.770	0	1.500	0.134
c_er	1.853	0.362	5.120	0	2.360	0.018	0.349	5.310	0	2.440	0.015
c_gc	1.711	0.227	7.550	0	3.140	0.002	0.221	7.730	0	3.210	0.001
c_gr	0.308	0.207	1.490	0.136	-3.350	0.001	0.206	1.490	0.135	-3.350	0.001

4. Conclusion

Instead of a one-size-fits-all solution, Apollo aims to offer a some-parts-fits-all, fill-out-the-details-yourself solution. The practical reality of this solution, as of today, is that users cannot easily take a set of the supplied example code, make simple adjustments, and apply the choice modelling technique to their own data. Our experience working with the package indicates that putting together a model with Apollo is more challenging than doing so with mlogit. However, the vision of creating choice models with modular code and having a level of flexibility that was previously not possible should, in principle, make Apollo an easy package to work with considering the alternative of adjusting packages that were not designed for flexibility. One might assume that the current problems with the execution can be attributed to the relatively early stage in which Apollo finds itself (as of today, version 0.1 has yet to be released). One can hope that the package, backed by the University of Leeds's *Choice Modelling Centre*, will continue to improve and eventually attract a community of proficient users.

Apollo requires the user to specify their model to the smallest detail explicitly. This can be a good thing or a bad thing. On the one hand, it prevents users from quickly copying and pasting code that worked in one model and applies it to another model. One certainly cannot hold it against Apollo that users are confronted with needing a thorough comprehension of their own models. However, in the cases where that level of comprehension is given, this requirement turns into an inconvenience. Where a mlogit user could easily just add variables to their dataset and run the same code, the Apollo user has to sit down for (hopefully) minutes, adjusting and debugging the code so that it works with the new dataset.

References

Croissant, Yves. "Estimation of multinomial logit models in R: The mlogit Packages." *R package version 0.2-2. URL: http://cran. r-project. org/web/packages/mlogit/vignettes/mlogit. pdf* (2012).

Croissant, Yves. *Package 'mlogit'*. Technical report, 2019.

Hess, Stephane, and David Palma. "Apollo: A flexible, powerful and customisable freeware package for choice model estimation and application." *Journal of choice modelling* 32 (2019): 100170.

McFadden, Daniel, and Kenneth Train. "Mixed MNL models for discrete response." *Journal of applied Econometrics* 15, no. 5 (2000): 447-470.

The R Foundation. *What is R?*. The R Project for Statistical Computing, viewed 11 January 2020, <https://www.r-project.org/about.html>

TIOBE Software BV. TIOBE *Index for January 2020*. TIOBE, viewed 11 January 2020, < https://www.tiobe.com/tiobe-index/>

Train, Kenneth E. *Discrete choice methods with simulation*. Cambridge university press, 2009.

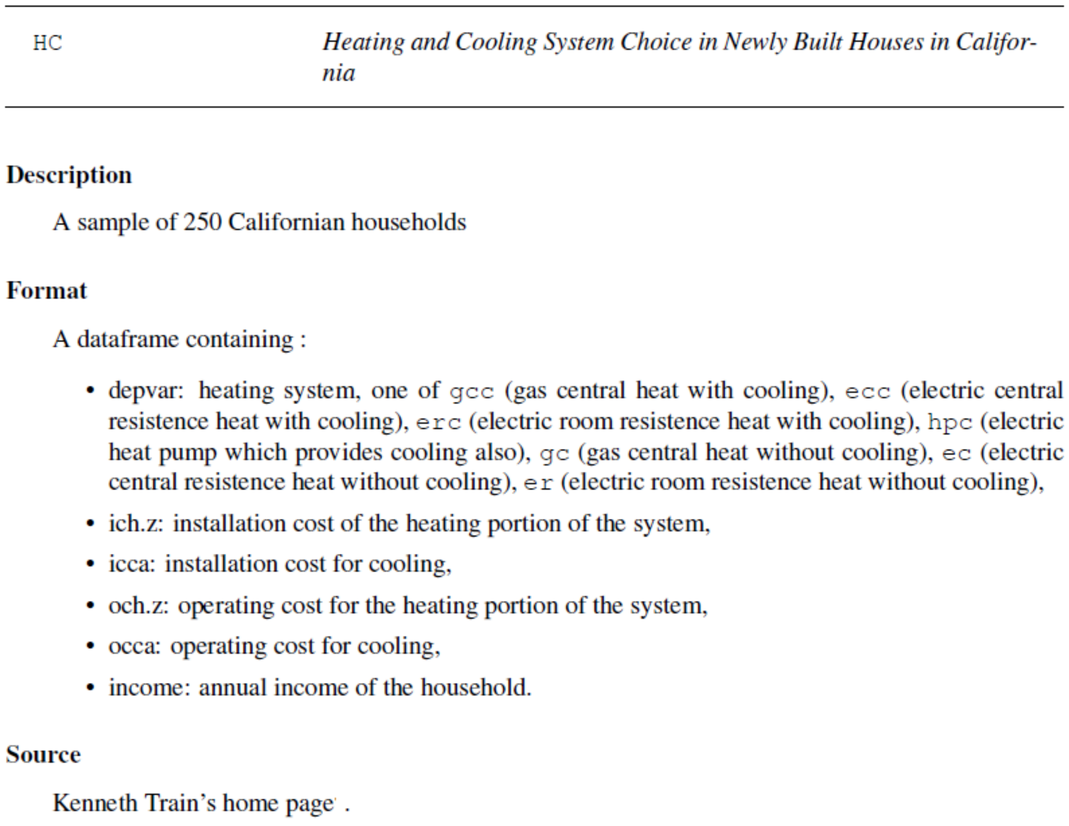
Train, Kenneth, and Yves Croissant. "Kenneth Train's exercises using the mlogit package for R." *R* 25 (2012): 0-2.

Appendix

The following appendix contains the R codes of our paper.

A.1. Dataset descriptions

The following dataset descriptions are taken from the mlogit documentation (Croissant, 2019, pp. 9, 12-13).



`HC`	*Heating and Cooling System Choice in Newly Built Houses in California*

Description

A sample of 250 Californian households

Format

A dataframe containing :

- depvar: heating system, one of `gcc` (gas central heat with cooling), `ecc` (electric central resistence heat with cooling), `erc` (electric room resistence heat with cooling), `hpc` (electric heat pump which provides cooling also), `gc` (gas central heat without cooling), `ec` (electric central resistence heat without cooling), `er` (electric room resistence heat without cooling),
- ich.z: installation cost of the heating portion of the system,
- icca: installation cost for cooling,
- och.z: operating cost for the heating portion of the system,
- occa: operating cost for cooling,
- income: annual income of the household.

Source

Kenneth Train’s home page .

`Heating`	*Heating System Choice in California Houses*

Description

A sample of 900 Californian households#'

Format

A dataframe containing:

- idcase: id,
- depvar: heating system, one of gc (gas central), gr (gas room), ec (electric central), er (electric room), hp (heat pump),
- ic.z: installation cost for heating system z (defined for the 5 heating systems),
- oc.z: annual operating cost for heating system z (defined for the 5 heating systems),
- pb.z: ratio oc.z/ic.z ,
- income: annual income of the household,
- agehed: age of the household head
- rooms: numbers of rooms in the house,

Source

Kenneth Train's home page .

`Electricity` *Stated preference data for the choice of electricity suppliers*

Description

A sample of 2308 households in the United States

Format

A dataframe containing :

- choice: the choice of the individual, one of 1, 2, 3, 4,
- id: the individual index,
- pfi: fixed price at a stated cents per kWh, with the price varying over suppliers and experiments, for scenario i=(1, 2, 3, 4),
- cli: the length of contract that the supplier offered, in years (such as 1 year or 5 years.) During this contract period, the supplier guaranteed the prices and the buyer would have to pay a penalty if he/she switched to another supplier. The supplier could offer no contract in which case either side could stop the agreement at any time. This is recorded as a contract length of 0,
- loci: is the supplier a local company,
- wki: is the supplier a well-known company,
- todi: a time-of-day rate under which the price is 11 cents per kWh from 8am to 8pm and 5 cents per kWh from 8pm to 8am. These TOD prices did not vary over suppliers or experiments: whenever the supplier was said to offer TOD, the prices were stated as above.
- seasi: a seasonal rate under which the price is 10 cents per kWh in the summer, 8 cents per kWh in the winter, and 6 cents per kWh in the spring and fall. Like TOD rates, these prices did not vary. Note that the price is for the electricity only, not transmission and distribution, which is supplied by the local regulated utility.

A.2. MNL Exercises – mlogit solution

As supplied in the solutions to the problem set (Train & Croissant, 2012).

```
#### Exercise 1 ####

library("mlogit")
data("Heating", package = "mlogit")
H <- mlogit.data(Heating, shape = "wide", choice = "depvar", varying = c(3:12))
m <- mlogit(depvar ~ ic + oc | 0, H)
mc <- mlogit(depvar ~ ic + oc, H, reflevel = 'hp')
summary(mc)
apply(fitted(mc, outcome = FALSE), 2, mean)


#### Exercise 6 ####

X <- model.matrix(mc)
alt <- index(H)$alt
chid <- index(H)$chid
eXb <- as.numeric(exp(X %*% coef(mc)))
SeXb <- tapply(eXb, chid, sum)
P <- eXb / SeXb[chid]
P <- matrix(P, ncol = 5, byrow = TRUE)
head(P)

apply(P, 2, mean)

apply(fitted(mc, outcome = FALSE), 2, mean)

#### Exercise 7 ####

Hn <- H
Hn[Hn$alt == "hp", "ic"] <- 0.9 * Hn[Hn$alt == "hp", "ic"]
apply(predict(mc, newdata = Hn), 2, mean)
```

Snippet 14: MNL exercises – mlogit solution

A.3. MNL Exercise 4 – Apollo solution

```
#  Multinomial logit model:


library('apollo')
library('mlogit')

#### Exercise 4, page 5 ####
#### a) How well do the estimated probabilities match the shares of customers choosing
each alternative?####

rm(list = ls())

# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                      ####
# ################################################################# #
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
  modelName  ="Apollo_mnl_ex4",
  modelDescr ="example",
  indivID    ="idcase",
  panelData = TRUE
)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                    ####
# ################################################################# #

#### Loading data from mlogit package  and preprocessing####
data("Heating", package = "mlogit")
database <- data.frame(Heating)
# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# During transformation from mlogit data class into data.frame column depvar tranformed
into factor, transform
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]
# ################################################################# #
#### DEFINE MODEL PARAMETERS                                    ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in estimation

apollo_beta <- c(ic_coef   = 0,
                 oc_coef   = 0,
                 c_ec = 0,
                 c_er = 0,
                 c_gc = 0,
                 c_gr = 0)
```

```
### Vector with names (in quotes) of parameters to be kept fixed at their starting value
in apollo_beta, use apollo_beta_fixed = c() if none

apollo_fixed = c()

# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

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

  ### List of utilities: these must use the same names as in mnl_settings, order is
irrelevant
  V <- list()

  V[['ec']]  <-  c_ec + (ic_coef * ic.ec + oc_coef * oc.ec)
  V[['er']]  <-  c_er + (ic_coef * ic.er + oc_coef * oc.er)
  V[['gc']]  <-  c_gc + (ic_coef * ic.gc + oc_coef * oc.gc)
  V[['gr']]  <-  c_gr + (ic_coef * ic.gr + oc_coef * oc.gr)
  V[['hp']]  <-  (ic_coef * ic.hp + oc_coef * oc.hp)

 ### Define settings for mnl:
  mnl_settings = list(
    alternatives = c(gc=1, er=2, gr=3, hp=4, ec=5),
    avail        = list(ec=1, er=1, gc=1, gr=1, hp=1),
    choiceVar    = database$alt_num,
    V            = V,
    estimateRoutine = "maxLik"
  )

  ### Compute logit probabilities
  P[["model"]]=apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  P = apollo_panelProd(P, apollo_inputs, functionality)

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

# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
                        estimate_settings = list(estimationRoutine = "nr", print.Level =
0))
```

```
# ----------------------------------------------------------------- #
#---- FORMATTED OUTPUT (TO SCREEN)                               ----
# ----------------------------------------------------------------- #

apollo_modelOutput(model, modelOutput_settings = list(printPVal = TRUE, printT1 = TRUE))

fitted_values <- na.omit(data.frame(apollo_prediction(model, apollo_probabilities,
apollo_inputs, modelComponent = "model")))
apply(fitted_values, 2, mean)[4:ncol(fitted_values) - 1]


#### b) Calculate the wtp and discount rate r that is implied by the estimates. Are
these reasonable?####

wtp <- coef(model)["oc_coef"] / coef(model)["ic_coef"]
r <- 1 / wtp
r

#### c) Update reflevel in model from 'hp' to 'gr' ####

#### Modified model apollo_probabalities: ####
apollo_beta <- c(ic_coef   = 0,
                 oc_coef   = 0,
                 c_ec = 0,
                 c_er = 0,
                 c_gc = 0,
                 c_hp = 0)


apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

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

  ### List of utilities: these must use the same names as in mnl_settings, order is
irrelevant
  V <- list()

  V[['ec']]  <-  c_ec + (ic_coef * ic.ec + oc_coef * oc.ec)
  V[['er']]  <-  c_er + (ic_coef * ic.er + oc_coef * oc.er)
  V[['gc']]  <-  c_gc + (ic_coef * ic.gc + oc_coef * oc.gc)
  V[['gr']]  <-  (ic_coef * ic.gr + oc_coef * oc.gr)
  V[['hp']]  <-  c_hp + (ic_coef * ic.hp + oc_coef * oc.hp)

  ### Define settings for mnl:
  mnl_settings = list(
    alternatives = c(gc=1, er=2, gr=3, hp=4, ec=5),
    avail        = list(ec=1, er=1, gc=1, gr=1, hp=1),
    choiceVar    = database$alt_num,
    V            = V
  )


  ### Compute logit probabilities
  P[["model"]]=apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  P = apollo_panelProd(P, apollo_inputs, functionality)

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

```
#### ####
model2 = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
                         estimate_settings = list(estimationRoutine = "nr", print.Level =
0, silent = TRUE))

apollo_modelOutput(model2, modelOutput_settings = list(printPVal = TRUE, printT1 =
TRUE))
```

Snippet 15: MNL exercise 4 - Apollo solution

A.4. MNL Exercise 4 – output comparison

Table 5: mlogit output

	Estimate	Std. Error	z-value	Pr(> | z|)
ec:(intercept)	1.659	0.448	3.699	0.0002
er:(intercept)	1.853	0.362	5.121	0.00000
gc:(intercept)	1.711	0.227	7.546	0
gr:(intercept)	0.308	0.207	1.492	0.136
ic	-0.002	0.001	-2.469	0.014
oc	-0.007	0.002	-4.502	0.00001

Table 6: Apollo output

	Estimate	Std. error	t value	Pr(> t)
ic_coef	-0.002	0.001	-2.469	0.014
oc_coef	-0.007	0.002	-4.502	0.00001
c_ec	1.659	0.448	3.699	0.0002
c_er	1.853	0.362	5.121	0.00000
c_gc	1.711	0.227	7.546	0
c_gr	0.308	0.207	1.492	0.136

A.5. MNL Exercise 6 – Apollo solution

```
#  Multinomial logit model:

library('apollo')

#### Exercise 6, page 9 ####
#### Estimate a model with installation costs, operating costs, and alternative specific
constants: ####

rm(list = ls())

# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                       ####
# ################################################################# #
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
  modelName  ="Apollo_mnl_ex6",
  modelDescr ="example",
  indivID    ="idcase",
  panelData = TRUE
)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

#### Loading data from mlogit package  and preprocessing####
data("Heating", package = "mlogit")
database <- data.frame(Heating)
# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# During transformation from mlogit data class into data.frame column depvar tranformed
into factor, transform
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]
# ################################################################# #
#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #


### Vector of parameters, including any that are kept fixed in estimation

apollo_beta <- c(ic_coef  = 0,
                 oc_coef  = 0,
                 c_ec = 0,
                 c_er = 0,
                 c_gc = 0,
                 c_hp = 0)

### Vector with names (in quotes) of parameters to be kept fixed at their starting value
in apollo_beta, use apollo_beta_fixed = c() if none

apollo_fixed = c()
```

```
# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                    ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                         ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

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

  ### List of utilities: these must use the same names as in mnl_settings, order is
irrelevant
  V <- list()

  V[['ec']]  <-  c_ec + (ic_coef * ic.ec + oc_coef * oc.ec)
  V[['er']]  <-  c_er + (ic_coef * ic.er + oc_coef * oc.er)
  V[['gc']]  <-  c_gc + (ic_coef * ic.gc + oc_coef * oc.gc)
  V[['gr']]  <-  (ic_coef * ic.gr + oc_coef * oc.gr)
  V[['hp']]  <-  c_hp + (ic_coef * ic.hp + oc_coef * oc.hp)

### Define settings for mnl:
  mnl_settings = list(
    alternatives = c(gc=1, er=2, gr=3, hp=4, ec=5),
    avail        = list(ec=1, er=1, gc=1, gr=1, hp=1),
    choiceVar    = database$alt_num,
    V             = V
  )


  ### Compute logit probabilities
  P[["model"]]=apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  P = apollo_panelProd(P, apollo_inputs, functionality)

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

# ################################################################# #
#### MODEL ESTIMATION                                             ####
# ################################################################# #

model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
                        estimate_settings = list(estimationRoutine = "nr", print.Level =
0, silent = T))
```

```
# ----------------------------------------------------------------- #
#---- FORMATTED OUTPUT (TO SCREEN)                               ----
# ----------------------------------------------------------------- #

apollo_modelOutput(model)

# Calculate the probabilities for each house explicitly
fitted_values <- na.omit(data.frame(apollo_prediction(model, apollo_probabilities,
apollo_inputs, modelComponent = "model")))

apply(fitted_values, 2, mean)[4:ncol(fitted_values) - 1]
```

Snippet 16: MNL exercise 6 - Apollo solution

A.6. MNL Exercise 7 – Apollo solution

```
#  Multinomial logit model:

library('apollo')

#### Exercise 6, page 9 ####
#### Estimate a model with installation costs, operating costs, and alternative specific
constants: ####

rm(list = ls())

# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                       ####
# ################################################################# #
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
  modelName  ="Apollo_mnl_ex6",
  modelDescr ="example",
  indivID    ="idcase",
  panelData = TRUE
)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

#### Loading data from mlogit package  and preprocessing####
data("Heating", package = "mlogit")
database <- data.frame(Heating)
# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# During transformation from mlogit data class into data.frame column depvar tranformed
into factor, transform
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]

# ################################################################# #

#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in estimation

apollo_beta <- c(ic_coef  = 0,
                 oc_coef  = 0,
                 c_ec = 0,
                 c_er = 0,
                 c_gc = 0,
                 c_hp = 0)
```

```
### Vector with names (in quotes) of parameters to be kept fixed at their starting value
in apollo_beta, use apollo_beta_fixed = c() if none

apollo_fixed = c()

# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

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

  ### List of utilities: these must use the same names as in mnl_settings, order is
irrelevant
  V <- list()

  V[['ec']]  <-  c_ec + (ic_coef * ic.ec + oc_coef * oc.ec)
  V[['er']]  <-  c_er + (ic_coef * ic.er + oc_coef * oc.er)
  V[['gc']]  <-  c_gc + (ic_coef * ic.gc + oc_coef * oc.gc)
  V[['gr']]  <-  (ic_coef * ic.gr + oc_coef * oc.gr)
  V[['hp']]  <-  c_hp + (ic_coef * ic.hp + oc_coef * oc.hp)

  ### Define settings for mnl:
  mnl_settings = list(
    alternatives = c(gc=1, er=2, gr=3, hp=4, ec=5),
    avail        = list(ec=1, er=1, gc=1, gr=1, hp=1),
    choiceVar    = database$alt_num,
    V            = V
  )


  ### Compute logit probabilities
  P[["model"]]=apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  P = apollo_panelProd(P, apollo_inputs, functionality)

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

# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
                        estimate_settings = list(estimationRoutine = "nr", print.Level =
0, silent = T))

#### Using the estimated coefficients from the model in exercise 6,
#calculate new probabilities and predicted shares using the new installation cost of
heat pump: ####
```

```
database['ic.hp'] <- 0.9 * database['ic.hp']


predictions_new = na.omit(data.frame(apollo_prediction(model, apollo_probabilities,
apollo_inputs)))

apply(predictions_new, 2, mean)[4:ncol(predictions_new) - 1]
```

Snippet 17: MNL exercise 7 - Apollo solution

A.7. Nested logit exercise 1 – Apollo solution

```
#  Multinomial logit model:

library('apollo')
library('mlogit')

#### Exercise 1, page 12 ####
#### Estimate a nested model logit: ####


# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                       ####
# ################################################################# #
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
  modelName  ="Apollo_NL_ex1",
  modelDescr ="example",
  indivID    = "ID",
  panelData = TRUE)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

#### Loading data from mlogit package  and preprocessing####
data("HC", package = "mlogit")
database <- na.omit(data.frame(HC))
# Make id column, as is required for apollo package
database[["ID"]] <- seq(1, dim(database)[1])
# During transformation from mlogit data class into data.frame column depvar tranformed
into factor, transform
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)

cooling.modes <- (database[['depvar']]) %in% c('gcc', 'ecc', 'erc', 'hpc')
room.modes <- (database[['depvar']]) %in% c('erc', 'er')
# create income variables for two sets cooling and rooms
database$inc.cooling <- database$inc.room <- 0
database$inc.cooling <- database$income
database$inc.room <- database$income
# create an intercet for cooling modes
database$int.cooling <- 1

# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]

# Creating new columns related to alternative specific:
x_vars <- list("icca", "occa", "inc.room", "inc.cooling", "int.cooling")
y_alt <- as.character(unique(database$depvar))
```

```
for (i in x_vars){
  for (j in y_alt){
      database[[paste(i, ".", j, sep='')]] <- 0
  }
}

for (i in c("icca", "occa")){
  for (j in y_alt){
    if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
      database[[paste(i, ".", j, sep='')]] <- database[[i]]
    }
  }
}


for (j in y_alt){
  if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
    database[[paste("inc.cooling", ".", j, sep='')]] <- database[["inc.cooling"]]
    database[[paste("int.cooling", ".", j, sep='')]] <- database[["int.cooling"]]}
  if (j %in%  c('erc', 'er')){
    database[[paste("inc.room", ".", j, sep='')]] <- database[["inc.room"]]
  }
}

# ################################################################# #
#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in estimation

apollo_beta <- c(ich_coef  = 0,
                 och_coef  = 0,
                 icca_coef = 0,
                 occa_coef = 0,
                 inc.room_coef = 0,
                 inc.cooling_coef = 0,
                 int.cooling_coef = 0,
                 lambda = 0.5)
### Vector with names (in quotes) of parameters to be kept fixed at their starting value
in apollo_beta, use apollo_beta_fixed = c() if none

apollo_fixed = c()


# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

  ### Create list of probabilities P
  P = list()
  ### List of utilities: these must use the same names as in nl_settings, order is
irrelevant
  V <- list()
  V[['gcc']] <-  ich_coef * ich.gcc + och_coef * och.gcc + icca_coef * icca.gcc +
occa_coef * occa.gcc + inc.cooling_coef * inc.cooling.gcc + int.cooling_coef *
int.cooling.gcc
  V[['ecc']] <-  ich_coef * ich.ecc + och_coef * och.ecc + icca_coef * icca.ecc +
occa_coef * occa.ecc+ inc.cooling_coef * inc.cooling.ecc + int.cooling_coef *
int.cooling.ecc
  V[['erc']] <-  ich_coef * ich.erc + och_coef * och.erc + icca_coef * icca.erc +
occa_coef * occa.erc + inc.room_coef * inc.room.erc + inc.cooling_coef * inc.cooling.erc
+ int.cooling_coef * int.cooling.erc
  V[['hpc']]  <-  ich_coef * ich.hpc + och_coef * och.hpc + icca_coef * icca.hpc +
occa_coef * occa.hpc + inc.cooling_coef * inc.cooling.hpc + int.cooling_coef *
int.cooling.hpc
  V[['gc']] <-  ich_coef * ich.gc + och_coef * och.gc
  V[['ec']] <-  ich_coef * ich.ec + och_coef * och.ec
  V[['er']] <-  ich_coef * ich.er + och_coef * och.er + inc.room_coef * inc.room.er

  ### Specify nests for NL model
  nlNests = list(root=1, nest1=lambda, nest2=lambda)

  ### Specify tree structure for NL model
  nlStructure= list()
  nlStructure[["root"]]   = c("nest1","nest2")
  nlStructure[["nest1"]]   = c("gcc","ecc", "erc", "hpc")
  nlStructure[["nest2"]]   = c("gc","ec","er")

  ### Define settings for nl:
  nl_settings = list(
    alternatives = c(erc=1, hpc=2, gcc=3, gc=4, er=5, ecc=6, ec=7),
    avail        = list(ec=1, ecc=1, er=1, erc=1, gc=1, gcc=1, hpc=1),
    choiceVar    = database$alt_num,
    V            = V,
    nlNests      = nlNests,
    nlStructure  = nlStructure
  )

  ### Compute probabilities using NL model
  P[["model"]] = apollo_nl(nl_settings, functionality)

  ### Take product across observation for same individual
  P = apollo_panelProd(P, apollo_inputs, functionality)

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

# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
estimate_settings = list(print.level = 0, estimationRoutine = "bfgs"))



# ################################################################# #
#### MODEL OUTPUTS                                               ####
# ################################################################# #

# ----------------------------------------------------------------- #
#---- FORMATTED OUTPUT (TO SCREEN)                               ----
# ----------------------------------------------------------------- #

apollo_modelOutput(model)
```

```
# b) Test the hypothesis that the log-sum coefficient is 1.0:

(coef(model)['lambda'] - 1) / sqrt(vcov(model)['lambda', 'lambda'])
# The critical value of t for 95% confidence is 1.96. So we can reject the hypothesis at
95% confidence.
# We can also use a likelihood ratio test because the multinomial logit is a special
case of the nested model.
# First estimate the multinomial logit model

apollo_beta <- c(ich_coef  = 0,
                 och_coef  = 0,
                 icca_coef = 0,
                 occa_coef = 0,
                 inc.room_coef = 0,
                 inc.cooling_coef = 0,
                 int.cooling_coef = 0)


apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

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

  ### List of utilities: these must use the same names as in nl_settings, order is
irrelevant
  V <- list()
  V[['gcc']] <-  ich_coef * ich.gcc + och_coef * och.gcc + icca_coef * icca.gcc +
occa_coef * occa.gcc + inc.cooling_coef * inc.cooling.gcc + int.cooling_coef *
int.cooling.gcc
  V[['ecc']] <-  ich_coef * ich.ecc + och_coef * och.ecc + icca_coef * icca.ecc +
occa_coef * occa.ecc+ inc.cooling_coef * inc.cooling.ecc + int.cooling_coef *
int.cooling.ecc
  V[['erc']] <-  ich_coef * ich.erc + och_coef * och.erc + icca_coef * icca.erc +
occa_coef * occa.erc + inc.room_coef * inc.room.erc + inc.cooling_coef * inc.cooling.erc
+ int.cooling_coef * int.cooling.erc
  V[['hpc']]  <-  ich_coef * ich.hpc + och_coef * och.hpc + icca_coef * icca.hpc +
occa_coef * occa.hpc + inc.cooling_coef * inc.cooling.hpc + int.cooling_coef *
int.cooling.hpc
  V[['gc']] <-  ich_coef * ich.gc + och_coef * och.gc
  V[['ec']] <-  ich_coef * ich.ec + och_coef * och.ec
  V[['er']] <-  ich_coef * ich.er + och_coef * och.er + inc.room_coef * inc.room.er


  ### Define settings for nl:
  mnl_settings = list(
    alternatives = c(erc=1, hpc=2, gcc=3, gc=4, er=5, ecc=6, ec=7),
    avail        = list(ec=1, ecc=1, er=1, erc=1, gc=1, gcc=1, hpc=1),
    choiceVar    = database$alt_num,
    V            = V)


  ### Compute probabilities using NL model
  P[["model"]] = apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  P = apollo_panelProd(P, apollo_inputs, functionality)

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

```
model2 = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities, apollo_inputs,
estimate_settings = list(print.level = 0, estimationRoutine = "bfgs"))

apollo_saveOutput(model2)

# LR-test:

apollo_lrTest("Apollo_NL_ex1", model)
```

Snippet 18: Nested logit exercise 1 - Apollo solution

A.8. Nested logit exercise 2 – Apollo solution

```r
apollo_control = list(
          modelName  ="Apollo_NL_ex2",
          modelDescr ="example",
          indivID    = "ID",
          panelData = TRUE)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

#### Loading data from mlogit package  and preprocessing####
data("HC", package = "mlogit")
database <- na.omit(data.frame(HC))
# Make id column, as is required for apollo package
database[["ID"]] <- seq(1, dim(database)[1])
# During transformation from mlogit data class into data.frame column depvar tranformed
into factor, transform
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)

cooling.modes <- (database[['depvar']]) %in% c('gcc', 'ecc', 'erc', 'hpc')
room.modes <- (database[['depvar']]) %in% c('erc', 'er')
# create income variables for two sets cooling and rooms
database$inc.cooling <- database$inc.room <- 0
database$inc.cooling <- database$income
database$inc.room <- database$income
# create an intercept for cooling modes
database$int.cooling <- 1
# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]
# Creating new columns related to alternative specific:
x_vars <- list("icca", "occa", "inc.room", "inc.cooling", "int.cooling")
y_alt <- as.character(unique(database$depvar))

for (i in x_vars){
          for (j in y_alt){
                    database[[paste(i, ".", j, sep='')]] <- 0
          }
}

for (i in c("icca", "occa")){
          for (j in y_alt){
                    if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
                              database[[paste(i, ".", j, sep='')]] <- database[[i]]
                    }
          }
}
```

```
for (j in y_alt){
          if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
                    database[[paste("inc.cooling", ".", j, sep='')]] <-
database[["inc.cooling"]]
          }
}


for (j in y_alt){
          if (j %in%  c('erc', 'er')){
                    database[[paste("inc.room", ".", j, sep='')]] <-
database[["inc.room"]]
          }
}


for (j in y_alt){
          if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
                    database[[paste("int.cooling", ".", j, sep='')]] <-
database[["int.cooling"]]
          }
}


# ################################################################# #
#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in estimation

apollo_beta <- c(ich_coef  = 0,
                 och_coef  = 0,
                 icca_coef = 0,
                 occa_coef = 0,
                 inc.room_coef = 0,
                 inc.cooling_coef = 0,
                 int.cooling_coef = 0,
                 lambda = 0.5)

### Vector with names (in quotes) of parameters to be kept fixed at their starting value
in apollo_beta, use apollo_beta_fixed = c() if none

apollo_fixed = c()


# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()


# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

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

          ### Create list of probabilities P
          P = list()
          ### List of utilities: these must use the same names as in nl_settings, order
is irrelevant
          V <- list()
          V[['gcc']] <-  ich_coef * ich.gcc + och_coef * och.gcc + icca_coef * icca.gcc
+ occa_coef * occa.gcc + inc.cooling_coef * inc.cooling.gcc + int.cooling_coef *
int.cooling.gcc
          V[['ecc']] <-  ich_coef * ich.ecc + och_coef * och.ecc + icca_coef * icca.ecc
+ occa_coef * occa.ecc+ inc.cooling_coef * inc.cooling.ecc + int.cooling_coef *
int.cooling.ecc
          V[['erc']] <-  ich_coef * ich.erc + och_coef * och.erc + icca_coef * icca.erc
+ occa_coef * occa.erc + inc.room_coef * inc.room.erc + inc.cooling_coef *
inc.cooling.erc + int.cooling_coef * int.cooling.erc
          V[['hpc']]  <-  ich_coef * ich.hpc + och_coef * och.hpc + icca_coef * icca.hpc
+ occa_coef * occa.hpc + inc.cooling_coef * inc.cooling.hpc + int.cooling_coef *
int.cooling.hpc
          V[['gc']] <-  ich_coef * ich.gc + och_coef * och.gc
          V[['ec']] <-  ich_coef * ich.ec + och_coef * och.ec
          V[['er']] <-  ich_coef * ich.er + och_coef * och.er + inc.room_coef *
inc.room.er

          ### Specify nests for NL model
          nlNests = list(root=1, central=lambda, room=lambda)

          ### Specify tree structure for NL model
          nlStructure= list()
          nlStructure[["root"]]   = c("central","room")
          nlStructure[["central"]]   = c("ec", "ecc", "gc", "gcc", "hpc")
          nlStructure[["room"]]   = c("er", "erc")

          ### Define settings for nl:
          nl_settings = list(
                    alternatives = c(erc=1, hpc=2, gcc=3, gc=4, er=5, ecc=6, ec=7),
                    avail        = list(ec=1, ecc=1, er=1, erc=1, gc=1, gcc=1, hpc=1),
                    choiceVar    = database$alt_num,
                    V            = V,
                    nlNests      = nlNests,
                    nlStructure  = nlStructure
          )


          ### Compute probabilities using NL model
          P[["model"]] = apollo_nl(nl_settings, functionality)

          ### Take product across observation for same individual
          P = apollo_panelProd(P, apollo_inputs, functionality)

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

```
# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

Ex2_base = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
apollo_inputs, estimate_settings = list(print.level = 0, estimationRoutine = "bfgs"))
```

Snippet 19: Nested logit exercise 2 - Apollo solution

A.9. Nested logit exercise 3 – Apollo solution

```
# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                       ####
# ################################################################# #
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
          modelName  ="Apollo_NL_ex3",
          modelDescr ="example",
          indivID    = "ID",
          panelData = TRUE)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

#### Loading data from mlogit package  and preprocessing####
data("HC", package = "mlogit")
database <- na.omit(data.frame(HC))
# Make id column, as is required for apollo package
database[["ID"]] <- seq(1, dim(database)[1])
# During transformation from mlogit data class into data.frame column
depvar tranformed into factor, transform
# this factor column into column of characters":
database$depvar <- as.character(database$depvar)

cooling.modes <- (database[['depvar']]) %in% c('gcc', 'ecc', 'erc',
'hpc')
room.modes <- (database[['depvar']]) %in% c('erc', 'er')
# create income variables for two sets cooling and rooms
database$inc.cooling <- database$inc.room <- 0
database$inc.cooling <- database$income
database$inc.room <- database$income
# create an intercet for cooling modes
database$int.cooling <- 1
# A list with numeric values of alternatives:
l1 <- seq(1, length(unique(database$depvar)))
names(l1) <- unique(database$depvar)
# Creating a column with a numeric version of alternatives:
database$alt_num <- l1[database$depvar]
# Creating new columns related to alternative specific:
y_alt <- as.character(unique(database$depvar))
```

```
for (i in x_vars){
          for (j in y_alt){
                    database[[paste(i, ".", j, sep='')]] <- 0
          }
}

for (i in c("icca", "occa")){
          for (j in y_alt){
                    if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
                              database[[paste(i, ".", j, sep='')]] <-
database[[i]]
                    }
          }
}


for (j in y_alt){
          if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
                    database[[paste("inc.cooling", ".", j, sep='')]] <-
database[["inc.cooling"]]
          }
}


for (j in y_alt){
          if (j %in%  c('erc', 'er')){
                    database[[paste("inc.room", ".", j, sep='')]] <-
database[["inc.room"]]
          }
}


for (j in y_alt){
          if (j %in%  c('gcc', 'ecc', 'erc', 'hpc')){
                    database[[paste("int.cooling", ".", j, sep='')]] <-
database[["int.cooling"]]
          }
}

# ################################################################# #
#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in
estimation

apollo_beta <- c(ich_coef  = 0,
                 och_coef  = 0,
                 icca_coef = 0,
                 occa_coef = 0,
                 inc.room_coef = 0,
                 inc.cooling_coef = 0,
                 int.cooling_coef = 0,
                 lambda1 = 0.5,
                 lambda2 = 0.5)
```

```
### Vector with names (in quotes) of parameters to be kept fixed at
their starting value in apollo_beta, use apollo_beta_fixed = c() if
none

apollo_fixed = c()


# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

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

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

          ### List of utilities: these must use the same names as in
nl_settings, order is irrelevant
          V <- list()
          V[['gcc']] <-  ich_coef * ich.gcc + och_coef * och.gcc +
icca_coef * icca.gcc + occa_coef * occa.gcc + inc.cooling_coef *
inc.cooling.gcc + int.cooling_coef * int.cooling.gcc
          V[['ecc']] <-  ich_coef * ich.ecc + och_coef * och.ecc +
icca_coef * icca.ecc + occa_coef * occa.ecc+ inc.cooling_coef *
inc.cooling.ecc + int.cooling_coef * int.cooling.ecc
          V[['erc']] <-  ich_coef * ich.erc + och_coef * och.erc +
icca_coef * icca.erc + occa_coef * occa.erc + inc.room_coef *
inc.room.erc + inc.cooling_coef * inc.cooling.erc + int.cooling_coef *
int.cooling.erc
          V[['hpc']]  <-  ich_coef * ich.hpc + och_coef * och.hpc +
icca_coef * icca.hpc + occa_coef * occa.hpc + inc.cooling_coef *
inc.cooling.hpc + int.cooling_coef * int.cooling.hpc
          V[['gc']] <-  ich_coef * ich.gc + och_coef * och.gc
          V[['ec']] <-  ich_coef * ich.ec + och_coef * och.ec
          V[['er']] <-  ich_coef * ich.er + och_coef * och.er +
inc.room_coef * inc.room.er
```

```
          ### Specify nests for NL model
          nlNests = list(root=1, nest1=lambda1, nest2=lambda2)

          ### Specify tree structure for NL model
          nlStructure= list()
          nlStructure[["root"]]   = c("nest1","nest2")
          nlStructure[["nest1"]]   = c("gcc","ecc", "erc", "hpc")
          nlStructure[["nest2"]]   = c("gc","ec","er")

          ### Define settings for nl:
          nl_settings = list(
                    alternatives = c(erc=1, hpc=2, gcc=3, gc=4, er=5,
ecc=6, ec=7),
                    avail        = list(ec=1, ecc=1, er=1, erc=1, gc=1,
gcc=1, hpc=1),
                    choiceVar   = database$alt_num,
                    V            = V,
                    nlNests      = nlNests,
                    nlStructure  = nlStructure
          )


          ### Compute probabilities using NL model
          P[["model"]] = apollo_nl(nl_settings, functionality)

          ### Take product across observation for same individual
          P = apollo_panelProd(P, apollo_inputs, functionality)

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

# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

Ex3_base = apollo_estimate(apollo_beta, apollo_fixed,
apollo_probabilities, apollo_inputs, estimate_settings =
list(print.level = 0, estimationRoutine = "bfgs"))
```

Snippet 20: Nested logit exercise 3 - Apollo solution

A.10. Mixed logit exercises – mlogit solution

As supplied in the solutions to the problem set (Train & Croissant, 2012).

```r
#### Load mlogit package and data ####

library(mlogit)
data("Electricity", package = "mlogit")
Electr <- mlogit.data(Electricity, id = "id", choice = "choice", varying = 3:26,
                      shape = "wide", sep = "")


#### Exercise 1 ####

Elec.mxl <- mlogit(choice ~ pf + cl + loc + wk + tod + seas | 0, Electr,
                   rpar = c(pf = 'n', cl = 'n', loc = 'n',
                            wk = 'n', tod = 'n', seas = 'n'),
                   R = 100, halton = NA, panel = TRUE)


#### Exercise 3 ####

Elec.mxl2 <- mlogit(choice ~ pf + cl + loc + wk + tod + seas | 0, Electr,
                    rpar = c(         cl = 'n', loc = 'n',
                             wk = 'n', tod = 'n', seas = 'n'),
                    R = 100, halton = NA, panel = TRUE)
```

Snippet 21: Mixed logit exercises - mlogit solutions

A.11. Mixed logit exercise 1 – Apollo solution

```
# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                       ####
# ################################################################# #

### Load Apollo and mlogit libraries
library(apollo)
library("mlogit")
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
  modelName ="Mixed logit",
  modelDescr ="Mixed logit model1",
  indivID   ="id",
  mixing    = TRUE,
  nCores = 4
)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

data("Electricity", package = "mlogit")
database <- data.frame(Electricity)

# ################################################################# #
#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in
estimation
apollo_beta = c(mu_pf                    =0,
                mu_cl                    =0,
                mu_loc                    =0,
                mu_wk                    =0,
                mu_tod                    =0,
                mu_seas                    =0,
                sd_pf                 = 0,
                sd_cl                 = 0,
                sd_loc                 = 0,
                sd_wk                 = 0,
                sd_tod                 = 0,
                sd_seas                 = 0)
```

```
### Vector with names (in quotes) of parameters to be kept fixed at
their starting value in apollo_beta, use apollo_beta_fixed = c() if
none
apollo_fixed = c()

# ################################################################# #
#### DEFINE RANDOM COMPONENTS                                    ####
# ################################################################# #

### Set parameters for generating draws
apollo_draws = list(
  interDrawsType = "pmc",
  interNDraws    = 100,
  interUnifDraws = c(),
  interNormDraws = c("draws_pf","draws_cl","draws_loc","draws_wk",
"draws_tod", "draws_seas"),
  intraDrawsType = "pmc",
  intraNDraws    = 0,
  intraUnifDraws = c(),
  intraNormDraws = c()
)

### Create random parameters
apollo_randCoeff = function(apollo_beta, apollo_inputs){
  randcoeff = list()

  randcoeff[["c_pf"]] = mu_pf + sd_pf * draws_pf
  randcoeff[["c_cl"]] = mu_cl + sd_cl * draws_cl
  randcoeff[["c_loc"]] = mu_loc + sd_loc * draws_loc
  randcoeff[["c_wk"]] = mu_wk + sd_wk * draws_wk
  randcoeff[["c_tod"]] = mu_tod + sd_tod * draws_tod
  randcoeff[["c_seas"]] = mu_seas + sd_seas * draws_seas

  return(randcoeff)
}

# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs,
functionality="estimate"){

  ### Function initialisation: do not change the following three
commands
  ### Attach inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))
  ### Create list of probabilities P
  P = list()

  ### List of utilities: these must use the same names as in
mnl_settings, order is irrelevant
  V = list()
  V[['alt1']] = c_pf * pf1 + c_cl * cl1 + c_loc * loc1 + c_wk * wk1 +
c_tod * tod1 + c_seas * seas1
  V[['alt2']] = c_pf * pf2 + c_cl * cl2 + c_loc * loc2 + c_wk * wk2 +
c_tod * tod2 + c_seas * seas2
  V[['alt3']] = c_pf * pf3 + c_cl * cl3 + c_loc * loc3 + c_wk * wk3 +
c_tod * tod3 + c_seas * seas3
  V[['alt4']] = c_pf * pf4 + c_cl * cl4 + c_loc * loc4 + c_wk * wk4 +
c_tod * tod4 + c_seas * seas4

  ### Define settings for MNL model component
  mnl_settings = list(
    alternatives  = c(alt1=1, alt2=2, alt3=3, alt4=4),
    avail         = list(alt1=1, alt2=1, alt3=1, alt4=1),
    choiceVar     = choice,
    V             = V
  )

  ### Compute probabilities using MNL model
  P[['model']] = apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  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)
}

# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

model = apollo_estimate(apollo_beta, apollo_fixed,
                        apollo_probabilities, apollo_inputs,
estimate_settings=list(hessianRoutine="maxLik", estimationRoutine =
"bfgs"))
```

```
# ################################################################# #
#### MODEL OUTPUTS                                               ####
# ################################################################# #

# ----------------------------------------------------------------- #
#---- FORMATTED OUTPUT (TO SCREEN)                               ----
# ----------------------------------------------------------------- #

apollo_modelOutput(model)
```

Snippet 22: Mixed logit exercise 1 - Apollo solution

A.12. Mixed logit exercise 1 – output comparison

Table 7: mlogit output

	Estimate	Std. Error	z-value	Pr(> | z|)
pf	-0.973	0.034	-28.359	0
cl	-0.206	0.013	-15.428	0
loc	2.076	0.080	25.808	0
wk	1.476	0.065	22.644	0
tod	-9.053	0.287	-31.518	0
seas	-9.104	0.289	-31.496	0
sd.pf	0.220	0.011	20.291	0
sd.cl	0.378	0.018	20.461	0
sd.loc	1.483	0.081	18.240	0
sd.wk	1.000	0.074	13.481	0
sd.tod	2.289	0.111	20.676	0
sd.seas	1.181	0.109	10.833	0

Table 8: Apollo output

	Estimate	Std. error	t value	Pr(> t)
mu_pf	-0.944	0.035	-27.224	0
mu_cl	-0.188	0.019	-10.000	0
mu_loc	2.136	0.098	21.729	0
mu_wk	1.519	0.074	20.567	0
mu_tod	-9.027	0.296	-30.506	0
mu_seas	-9.211	0.298	-30.928	0
sd_pf	0.223	0.015	14.866	0
sd_cl	-0.362	0.021	-17.416	0
sd_loc	1.508	0.103	14.640	0
sd_wk	-0.863	0.074	-11.662	0
sd_tod	2.127	0.155	13.681	0
sd_seas	-1.394	0.130	-10.688	0

A.13. Mixed logit exercise 3 – Apollo solution

```
# ################################################################# #
#### LOAD LIBRARY AND DEFINE CORE SETTINGS                       ####
# ################################################################# #

### Load Apollo and mlogit libraries
library(apollo)
library(mlogit)
### Initialise code
apollo_initialise()

### Set core controls
apollo_control = list(
  modelName ="Mixed logit",
  modelDescr ="Mixed logit model 3",
  indivID   ="id",
  mixing     = TRUE,
  nCores = 4
)

# ################################################################# #
#### LOAD DATA AND APPLY ANY TRANSFORMATIONS                     ####
# ################################################################# #

data("Electricity", package = "mlogit")
database <- data.frame(Electricity)

# ################################################################# #
#### DEFINE MODEL PARAMETERS                                     ####
# ################################################################# #

### Vector of parameters, including any that are kept fixed in estimation
apollo_beta = c(mu_pf                  =0,
                mu_cl                  =0,
                mu_loc                  =0,
                mu_wk                  =0,
                mu_tod                  =0,
                mu_seas                  =0,
#               sd_pf               = 0,
                sd_cl               = 0,
                sd_loc               = 0,
                sd_wk               = 0,
                sd_tod               = 0,
                sd_seas               = 0)

### Vector with names (in quotes) of parameters to be kept fixed at their starting value
in apollo_beta, use apollo_beta_fixed = c() if none
apollo_fixed = c()
```

```
# ################################################################# #
#### DEFINE RANDOM COMPONENTS                                    ####
# ################################################################# #

### Set parameters for generating draws
apollo_draws = list(
  interDrawsType = "pmc",
  interNDraws    = 100,
  interUnifDraws = c(),
  interNormDraws = c("draws_pf","draws_cl","draws_loc","draws_wk", "draws_tod",
"draws_seas"),
  intraDrawsType = "pmc",
  intraNDraws    = 0,
  intraUnifDraws = c(),
  intraNormDraws = c()
)


### Create random parameters
apollo_randCoeff = function(apollo_beta, apollo_inputs){
  randcoeff = list()

# randcoeff[["c_pf"]] = mu_pf + sd_pf * draws_pf
  randcoeff[["c_cl"]] = mu_cl + sd_cl * draws_cl
  randcoeff[["c_loc"]] = mu_loc + sd_loc * draws_loc
  randcoeff[["c_wk"]] = mu_wk + sd_wk * draws_wk
  randcoeff[["c_tod"]] = mu_tod + sd_tod * draws_tod
  randcoeff[["c_seas"]] = mu_seas + sd_seas * draws_seas

  return(randcoeff)
}

# ################################################################# #
#### GROUP AND VALIDATE INPUTS                                   ####
# ################################################################# #

apollo_inputs = apollo_validateInputs()

# ################################################################# #
#### DEFINE MODEL AND LIKELIHOOD FUNCTION                        ####
# ################################################################# #

apollo_probabilities=function(apollo_beta, apollo_inputs, functionality="estimate"){

  ### Function initialisation: do not change the following three commands
  ### Attach inputs and detach after function exit
  apollo_attach(apollo_beta, apollo_inputs)
  on.exit(apollo_detach(apollo_beta, apollo_inputs))

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

  ### List of utilities: these must use the same names as in mnl_settings, order is
irrelevant
  V = list()
  V[['alt1']] = mu_pf * pf1 + c_cl * cl1 + c_loc * loc1 + c_wk * wk1 + c_tod * tod1 +
c_seas * seas1
  V[['alt2']] = mu_pf * pf2 + c_cl * cl2 + c_loc * loc2 + c_wk * wk2 + c_tod * tod2 +
c_seas * seas2
  V[['alt3']] = mu_pf * pf3 + c_cl * cl3 + c_loc * loc3 + c_wk * wk3 + c_tod * tod3 +
c_seas * seas3
  V[['alt4']] = mu_pf * pf4 + c_cl * cl4 + c_loc * loc4 + c_wk * wk4 + c_tod * tod4 +
c_seas * seas4
```

```
  ### Define settings for MNL model component
  mnl_settings = list(
    alternatives  = c(alt1=1, alt2=2, alt3=3, alt4=4),
    avail         = list(alt1=1, alt2=1, alt3=1, alt4=1),
    choiceVar     = choice,
    V             = V
  )


  ### Compute probabilities using MNL model
  P[['model']] = apollo_mnl(mnl_settings, functionality)

  ### Take product across observation for same individual
  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)
}

# ################################################################# #
#### MODEL ESTIMATION                                            ####
# ################################################################# #

Ex3_model = apollo_estimate(apollo_beta, apollo_fixed, apollo_probabilities,
                        apollo_inputs, estimate_settings = list(hessianRoutine="maxLik",
estimationRoutine = "bfgs"))
```

Snippet 23: Mixed logit exercise 3 - Apollo solution