Estimation of discrete choice models: extending BIOGEME

Michel Bierlaire

Mamy Fetiarison

STRC 2009 **September 2009**

Estimation of discrete choice models: extending BIOGEME

Michel Bierlaire	Mamy Fetiarison
TRANSP-OR	TRANSP-OR
EPFL	EPFL
Lausanne	Lausanne
phone: +41 21 693 25 37	phone: +41 21 693 81 00
fax: +41 21 693 80 60	fax: +41 21 693 80 60
michel.bierlaire@epfl.ch	mamynirina.fetiarison@epfl.ch

September 2009

Abstract

Discrete choice models are constantly in evolution in the literature. Since they enable to capture wide range of situations, they have been widely used by researchers and also practitioners in several fields of applications including econometrics and transportation demand analysis. However, estimation procedures are complicated and not always easily available to researchers.

BIOGEME is a free software package for estimating by maximum likelihood a broad range of random utility models. It can estimate particularly Multivariate Extreme Value (MEV) models including the logit model, the nested logit model, the cross-nested logit model, and the network MEV model, as well as continuous and discrete mixtures of these models. Biogeme has been designed to provide modelers with tools to investigate a wide variety of discrete choice models without worrying about the estimation algorithm itself.

In this paper, we present some new features and capabilities of Biogeme. To make it more flexible, we allow explicitly the user to specify the random utility model to be estimated and the associated likelihood function. With simple formulations, it will be able to handle more sophisticated models such as latent variable models, latent class models, dynamic models, etc. required by modern modeling practice, in particular in transportation.

Keywords

logit model, cross-nested model, GEV model, latent variable model, latent class model

1 Introduction

Random utility models (RUM) have been intensively used by researchers and practitioners in several fields of applications including econometrics and transportation demand analysis. The development of a great deal of new models, designed to model complex behavioral aspects, has characterized the research activities during the last decades.

The development of Biogeme (Bierlaire (2003)) has been motivated by the need to estimate the parameters of these new models from real data. The first version of Biogeme, released in 2001, was designed to estimate the models from the family of Multivariate Extreme Value (MEV) models (called "Generalized Extreme Value" models by McFadden (1978)). The version 0.7, released in 2003, introduced random parameters in the models, so that the parameters of mixtures of MEV models could be estimated. Originally designed for the courses and research at the Ecole Polytechnique Fédérale de Lausanne (EPFL), Biogeme is now widely used in the research community. As of today, more than 900 persons are registered on the users group.

Nowadays, more complex models are being proposed and used. Among them, hybrid choice models (Ben-Akiva *et al.* (2002), Walker (2001)) have received a considerable attention. In particular, the possibility to include latent variables and latent classes into the discrete choice framework allows to exploit psychometric data. Also, dynamic models accounting for panel data are more and more considered (see, among many, Train (2003)). The current version of Biogeme allows only the estimation of very simple latent class models (based on discrete random variables), and of simple models with panel data.

Software development requires significant resources. This is why it is difficult for a software like Biogeme to catch up with recent model developments. We describe in this paper a new version of Biogeme based on a different philosophy. The software is divided into two parts. The first part, implemented in C++, is taking care of the estimation itself and, in particular, of the optimization algorithms. Most of this code is adapted from the previous code. The second part, implemented in Python, is taking care of the models and the likelihood function. The Python code is automatically translated into a C++ code that computes the loglikelihood function and its derivatives, so that it can be fed into the optimizer.

For existing models, the type of input required by the user is essentially the same as for the former version of Biogeme: description of the parameters, specification of the utility functions, and choice of the model. Therefore, it does not necessarily require knowledge of the Python language. But if the user knows Python, she can benefit from all the features of this programming language (loops, complex data structures, if-then-else statements, etc.). And for new models, the user can write the full specification of the model and the likelihood function in Python, and use the software in the exact same way.

2 Model specification

The specification of the model with the new Biogeme is based on the following idea. First, the specification of simple models follows almost the same structure as the previous version of the software. Minor differences, due to the usage of the Python language, have been included and are discussed below. But regular users of Biogeme should be able to use the new version with a minimum adjustment. Second, the user can exploit the power of the Python language to write more complex models. A typical example is when the number of alternatives is large, and loops are convenient to write the specification. Finally, new models, or set of models, with associated likelihood functions, can be added in a flexible way to Biogeme.

We start below by describing the main modeling elements used for the model specification. We then illustrate their use on a simple example and provide a specification of a latent variable model.

2.1 Modeling elements

- **Fixed parameters** are the parameters of the model, that will be estimated. For each of them, a statement like the following must be provided:

  ```
  COST =  Beta( 'COST', 0.0, -10000, 10000, 0, 'Cost param.')
  ```

  where the Python variable on the left hand side can be reused in any future expression, and the six arguments of the `Beta` function are defined as follows:

  1. Name of the variable (used for the reporting),
  2. default value,
  3. lower bound,
  4. upper bound,
  5. status (0 if the parameter must be estimated, 1 if it must be maintained at its default value by the algorithm),
  6. Short description (optional, used for the reporting in LaTeX).

  This is exactly the same information provided in the Section `[Beta]` of the current version of the package.
- **Variables** are the headers of the data file, that is, the explanatory variables of the model. The statement

  ```
  dataFile = "sample.dat"
  ```

  defines all the headers in the data file as Python entities that can be used in further expressions. Clearly, any transform can be applied to these variables before being used in the specification itself, such as

```
logCost = log(cost).
```

Biogeme also creates an additional variables named

```
__rowId__
```

which contains the number of the observation in the data file, starting counting from 0. Note that the data file must have exactly the same format as in the former version of Biogeme.

- **Random numbers** Biogeme handles two types of random numbers. The first type follows a normal distribution with mean 0 and variance 1, that is $N(0, 1)$. The second follows a uniform distribution between -1 and 1, that is $U[-1, 1]$. The syntax is the following:

  ```
  bioNormal('aNormal','__rowId__')
  ```

  defines a normal distribution called `aNormal`. The software will generate a set of draws from this distribution for each different value of the identifier mentioned as the second argument. In this example, the keyword `__rowId__` refers to the row number in the data file, meaning that a different set of draws will be generated for each row.

  ```
  bioUniform('aUniform','individualId')
  ```

  defines a uniform distribution in $[-1, 1]$ called `aUniform` such that a set of draws is generated for each individual in the sample, not each observation. This feature is particularly used in the context of panel or stated preference data. Note that if a $[0, 1]$ uniform distribution is needed, it is obtained from the following transform:

  ```
  (1 + bioUniform('aUniform','individualId')) / 2
  ```

  A wide variety of random parameters can be derived from these random numbers. For instance,

  ```
  BETA1NORMAL = BETA1 + SIGMA1 * bioNormal('aNormal','__rowId__')
  ```

  defines a random parameter, normally distributed, with mean `BETA1` and standard deviation `SIGMA1`, where these two parameters have been properly defined with the function `Beta` as described above.

  ```
  zeroOne = (1 + bioUniform('aUniform','Id')) / 2
  BETA2EXTREME = A - B * log(-log(zeroOne))
  ```

  defines a random parameter following an extreme value distribution with location parameter `A` and scale parameter `B`.

- Elements for building **mathematical expressions** are also provided. These elements consist of common and expanded mathematical operators and functions:

  1. Numerical operations : `-` (unary minus), `+` (addition), `-` (substraction), `*` (multiplication), `/` (division), `abs` (absolute value), `log` (natural logarithm), `exp` (exponential), `**` (power)

2. Boolean operations : `&` (and), `|` (or), `<` (less than), `<=` (less or equal to), `>` (greater than), `>=` (greater or equal to), `==` (equal to), `<>` (not equal to).
3. Element of a dictionary: there are typical cases where several expressions are defined (like the utility functions for each alternative), but only one of them is relevant in a given expression (typically, the utility function corresponding to the chosen alternative). We call the set of expressions a "dictionary". Biogeme provides the function `Elem(dictionary, expression)` which enables to reference the item associated to the key `expression` in `dictionary`. In Python, a dictionary is a data structure organized like a set of (*key*, *value*) pairs, where keys must be unique. The following example presents how to get the utility of the chosen alternative:

   ```
   V = {1: V1,
        2: V2,
        3: V3,
        4: V4,
        5: V5,
        6: V6}
   Vchosen = Elem(V,choice)
   ```

   where `choice` is an expression that returns one of the values of the key, that is `1`, `2`, `3`, `4`, `5` or `6`, and `V1`, `V2`, `V3`, `V4`, `V5` and `V6` have been specified earlier.
4. Iterators[1] are important components that allow to iterate on data, either from the sample file or from generated draws for random number. There are 3 kinds of iterator:

   – 'Row' iterator: the element referenced by this iterator is a vector of numbers, typically a row from the data file;
   – 'Meta' iterator: defines an iterator on another iterator. This permits to describe a hierarchical structure of iterators and allows to model expression such as $\sum_n \prod_o \sum_k p(n, o, k)$. This is particularly useful in the specification of the loglikelihood function of models for panel data.
   – 'Draw' iterator: iterates through the draws of random numbers, and is useful for the computation of integrals by simulation.

   The following declarations describe the construction of each iterator object in Python

   ```
   rowIterator(iteratorName,dataStructure,indexVariable)
   metaIterator(iteratorName,dataStructure,indexVariable)
   ```

   The 3 arguments of these functions are respectively:

[1]At the time this paper is written, this feature is being tested and improved. Its syntax may slightly vary in the future, but the logic will be as described in the paper.

   (a) `iteratorName`: name of the iterator;
   (b) `dataStructure`: data structure that the iterator is iterating on. It can be either the object `Datafile('myfile.dat')`, when the iterator scans the sample file, or a string of characters with the name of a 'meta' iterator.
   (c) `indexVariable` (optional) is the name of the column in the data file where the identifier on which the iterator iterates is defined. Typically, `__rowId__` or the identifier of an individual for panel data.

   The draw iterator is defined as `drawIterator(iteratorName)`.

5. Functions `Sum(term,iterator)` and `Prod(term,iterator)` express summation and product of terms (Eg.: $\sum_i x_i$ or $\prod_j x_j$). `iterator` is the name of the iterator used to access the successive values of the variables in the database, making possible the evaluation of `term` for each referenced element. `term` is a general expression. It can therefore include other summation or product operation in a recursive way. See examples below.

2.2 Example : Logit model and mixtures

A specification of a logit model is presented in this section. First, the specification of the model is described. Then the corresponding specification in Python is given as well as some variants to illustrate the use of loops and model for panel data.

2.2.1 Model

Assume there are 6 alternatives in the choice set for each individual. The deterministic part of the utility functions is defined as follows :

$$\begin{aligned}
V_1 &= \text{ASC}_1 + \beta_1 * \text{time}_1 + \beta_2 * \text{cost}_1 \\
V_2 &= \text{ASC}_2 + \beta_1 * \text{time}_2 + \beta_2 * \text{cost}_2 \\
V_3 &= \text{ASC}_3 + \beta_1 * \text{time}_3 + \beta_2 * \text{cost}_3 \\
V_4 &= \text{ASC}_4 + \beta_1 * \text{time}_4 + \beta_2 * \text{cost}_4 \\
V_5 &= \text{ASC}_5 + \beta_1 * \text{time}_5 + \beta_2 * \text{cost}_5 \\
V_6 &= \text{ASC}_6 + \beta_1 * \text{time}_6 + \beta_2 * \text{cost}_6
\end{aligned}$$

where $\text{ASC}_1 = 0$ (fixed value). ASC_i, $i \in \{2, ..., 6\}$, β_1 and β_2 are the parameters that must be estimated. time_j and cost_j, $j \in \{1, ..., 6\}$ are the explanatory variables. The probability of choosing alternative i by individual n within the choice set C_n is

$$P_n(i|C_n) = \frac{A_{in}e^{V_i}}{\sum_{j=1}^{6} A_{jn}e^{V_j}}, \tag{1}$$

where

$$A_{in} = \begin{cases} 1 & \text{if alternative } i \text{ belongs to } \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

For numerical reasons, it is useful to consider the equivalent formulation

$$P_n(i|C_n) = \frac{A_{in}e^{V_i - V_i}}{\sum_{j=1}^{6} A_{jn}e^{V_j - V_i}} = \frac{A_{in}}{\sum_{j=1}^{6} A_{jn}e^{V_j - V_i}}. \quad (2)$$

First, it is less likely that the argument of the exponential will generate an overflow. Second, we save the computation of an exponential at the numerator. The loglikelihood for a sample with N observations is given by

$$\mathcal{L} = \sum_{n=1}^{N} \ln P_n(i_n|C_n), \quad (3)$$

where i_n is the alternative chosen by individual n.

2.2.2 A simple specification

The full specification in Python of this model is written below. Comments begin with #, so that characters following this symbol on the same line will be ignored by Biogeme. In this specification, we assume that the file `'sample.dat'` contains the data in the appropriate format.

```
# Import modules
from biogeme import *
from headers import *
from logit import *
from loglikelihood import *

# File containing a sample
dataFile = "sample.dat"

# Parameters
ASC1  =  Beta( 'ASC1', 0.0, -10000, 10000, 1, 'Cte for alt. 1')
ASC2  =  Beta( 'ASC2', 0.0, -10000, 10000, 0, 'Cte for alt. 2')
ASC3  =  Beta( 'ASC3', 0.0, -10000, 10000, 0, 'Cte for alt. 3')
ASC4  =  Beta( 'ASC4', 0.0, -10000, 10000, 0, 'Cte for alt. 4')
ASC5  =  Beta( 'ASC5', 0.0, -10000, 10000, 0, 'Cte for alt. 5')
ASC6  =  Beta( 'ASC6', 0.0, -10000, 10000, 0, 'Cte for alt. 6')
```

```
BETA1 =  Beta( ’BETA1’, 0, -10000, 10000, 0, ’\beta_1’)
BETA2 =  Beta( ’BETA2’, 0, -10000, 10000, 0, ’\beta_2’)

# Utility.  Note that it is not necessary anymore to write ASC2 * one.
V1 =   ASC1 + BETA1 * time1 + BETA2 * cost1
V2 =   ASC2 + BETA1 * time2 + BETA2 * cost2
V3 =   ASC3 + BETA1 * time3 + BETA2 * cost3
V4 =   ASC4 + BETA1 * time4 + BETA2 * cost4
V5 =   ASC5 + BETA1 * time5 + BETA2 * cost5
V6 =   ASC6 + BETA1 * time6 + BETA2 * cost6

# Dictionary containing the utilities. The index must correspond
# to the values that the choice variable may take
V = {1: V1,
     2: V2,
     3: V3,
     4: V4,
     5: V5,
     6: V6}

# Dictionary containing the definition of availability.
# Here, they are taken directly from the data file
av = {1: av1,
      2: av2,
      3: av3,
      4: av4,
      5: av5,
      6: av6}

# Model
prob = logit(V,av,Choice)

# Likelihood function
# Definition of the iterator
rowIterator(’obsIter’, Datafile(dataFile))
# Loglikelihood function
BIOGEME_OBJECT.FORMULA = Sum(log(prob),’obsIter’)
```

`prob = logit(V,av,choice)` specifies the model to be estimated. This Python function implements equation (2) and is defined as follows :

```
def logit(V,availability,choice) :
    chosen = Elem(V,choice)
    den = 0
    for i,v in V.iteritems() :
        den += availability[i] * exp(v-chosen)

    a = Elem(availability,choice)
    P = a / den
    return P
```

We refer the interested reader to the Python documentation (`python.org`) to decrypt this piece of code.

2.2.3 Using loops

Assume that the population is segmented, and we want a specification where one parameter is segment-specific. For the sake of the example, we assume that the identifier of the individual characterizes the group she belongs to in the following way:

- Group 0: ids from 0 to 20,
- Group 1: ids from 21 to 100,
- Group 2: ids from 101 to 150,
- Group 3: ids from 151 and above.

Groups may be defined based on income, age, or any appropriate socio-economic characteristics. The following specification involves the use of loops in Python, and should be self-explanatory:

```
BETA1 = {}
BETA1[0] = Beta( 'BETA_time_g0', 0, -10000, 10000, 0, 'beta time group 0')
BETA1[1] = Beta( 'BETA_time_g1', 0, -10000, 10000, 0, 'beta time group 1')
BETA1[2] = Beta( 'BETA_time_g2', 0, -10000, 10000, 0, 'beta time group 2')
BETA1[3] = Beta( 'BETA_time_g3', 0, -10000, 10000, 0, 'beta time group 3')

group = {}
group[0] = ((Id >= 0) & (Id <= 20))
group[1] = ((Id >= 21) & (Id <= 100))
group[2] = ((Id >= 101) & (Id <= 150))
group[3] = (Id >= 151)
```

```
V1 = ASC1 + BETA2 * cost1
for i in range(4):  # <=> for i in [0,1,2,3]
    V1 += BETA1[i] * group[i] * time1
```

By convention, the result of a logical expression is 1 if the expression is true, and 0 otherwise. Therefore, for each individual, exactly one term involving `time1` is non-zero.

2.2.4 Panel data

We illustrate here the use of iterators to specify a model for panel data. Assume again that there are 6 alternatives as above. The utility functions are now defined as:

$$\begin{aligned}
V_{1nt} &= \text{ASC}_1 + \beta_1 * \text{time}_{1nt} + \beta_2 * \text{cost}_{1nt} \\
V_{2nt} &= \text{ASC}_2 + \beta_1 * \text{time}_{2nt} + \beta_2 * \text{cost}_{2nt} + \xi_n \\
V_{3nt} &= \text{ASC}_3 + \beta_1 * \text{time}_{3nt} + \beta_2 * \text{cost}_{3nt} + \xi_n \\
V_{4nt} &= \text{ASC}_4 + \beta_1 * \text{time}_{4nt} + \beta_2 * \text{cost}_{4nt} + \xi_n \\
V_{5nt} &= \text{ASC}_5 + \beta_1 * \text{time}_{5nt} + \beta_2 * \text{cost}_{5nt} + \xi_n \\
V_{6nt} &= \text{ASC}_6 + \beta_1 * \text{time}_{6nt} + \beta_2 * \text{cost}_{6nt} + \xi_n
\end{aligned}$$

where ξ_n is an error component distributed across individuals (not observations). If ξ_n were known, the probability for individual n to make choice i_{nt} at time t is given by the logit model:

$$P_{nt}(i_{nt}|\xi_n) = \frac{A_{i_{nt}nt} e^{V_{i_{nt}nt}}}{\sum_j A_{jnt} e^V_{jnt}}. \tag{4}$$

The probability that this individual makes the sequence of choices at each time period, knowing ξ_n, is given by

$$P_n(\{i_{n1}, \ldots, i_{nT}\}|\xi_n) = \prod_{t=1}^{T} P_{nt}(i_{nt}|\xi_n). \tag{5}$$

As ξ_n is distributed, we have

$$P_n(\{i_{n1}, \ldots, i_{nT}\}) = \int_\xi P_n(\{i_{n1}, \ldots, i_{nT}\}|\xi) f(\xi) d\xi, \tag{6}$$

where f is the probability density function of ξ. This is approximated by

$$P_n(\{i_{n1}, \ldots, i_{nT}\}) \approx \frac{1}{R} \sum_r P_n(\{i_{n1}, \ldots, i_{nT}\}|\xi_r), \tag{7}$$

where ξ_r are random draws from the appropriate distribution. The loglikelihood function is therefore

$$\mathcal{L} = \sum_n \log P_n(\{i_{n1}, \ldots, i_{nT}\}). \tag{8}$$

If we assume that ξ_n follows a normal distribution of mean 0 and variance σ^2, the above model can be specified as follows.

```
from biogeme import *
from logit import *
from headers import *

dataFile = "sample.dat"

ASC1 =  Beta( 'ASC1', 0.0, -10000, 10000, 1, 'Cte for alt. 1')
ASC2 =  Beta( 'ASC2', 0.0, -10000, 10000, 0, 'Cte for alt. 2')
ASC3 =  Beta( 'ASC3', 0.0, -10000, 10000, 0, 'Cte for alt. 3')
ASC4 =  Beta( 'ASC4', 0.0, -10000, 10000, 0, 'Cte for alt. 4')
ASC5 =  Beta( 'ASC5', 0.0, -10000, 10000, 0, 'Cte for alt. 5')
ASC6 =  Beta( 'ASC6', 0.0, -10000, 10000, 0, 'Cte for alt. 6')
BETA1 =  Beta( 'BETA1', 0, -10000, 10000, 0, '\beta_1')
BETA2 =  Beta( 'BETA2', 0, -10000, 10000, 0, '\beta_2')
SIGMA = Beta( 'SIGMA', 1.0, -10000, 10000, 0, '\sigma')

# Id is the identifier of the individual. Draws will be
# generated for each individual, and not for each observation.
ERRORCOMP = SIGMA * bioNormal('aNormal','Id')

V1 =     ASC1 + BETA1 * x11 + BETA2 * x12
V2 =     ASC2 + BETA1 * x21 + BETA2 * x22 + ERRORCOMP
V3 =     ASC3 + BETA1 * x31 + BETA2 * x32 + ERRORCOMP
V4 =     ASC4 + BETA1 * x41 + BETA2 * x42 + ERRORCOMP
V5 =     ASC5 + BETA1 * x51 + BETA2 * x52 + ERRORCOMP
V6 =     ASC6 + BETA1 * x61 + BETA2 * x62 + ERRORCOMP

V = {1: V1,
     2: V2,
     3: V3,
     4: V4,
     5: V5,
     6: V6}
```

```
av = {1: av1,
      2: av2,
      3: av3,
      4: av4,
      5: av5,
      6: av6}

metaIterator('personIter',Datafile(dataFile),'Id')
rowIterator('panelObsIter','personIter','__rowId__')
drawIterator('drawIter')

prob = logit(V,av,choice)
condProbIndiv = Prod(prob,'panelObsIter')
probIndiv = Sum(condProbIndiv,'drawIter')
loglikelihood = Sum(log(probIndiv),'personIter')
BIOGEME_OBJECT.FORMULA = loglikelihood
BIOGEME_OBJECT.DRAWS = 1000
```

2.3 More complex models

We present here the specification of a complex model proposed by Abou Zeid (2009). It includes latent variables with indicators, ordered logit and correlated error components. We refer the reader to the thesis for the description and motivation of the model. For information, the model has been translated into Python in less than 20 minutes.

Structural Model

$$U_{\text{Car}} = \beta_0 + \beta_1 \text{Time}_{\text{Car}} + \beta_2 \frac{\text{Cost}_{\text{Car}}}{\text{Income}} + \varepsilon_{\text{Car}}. \tag{9}$$

$$U_{\text{PT}} = \beta_1 \text{Time}_{\text{PT}} + \beta_2 \frac{\text{Cost}_{\text{PT}}}{\text{Income}} + \varepsilon_{\text{PT}}. \tag{10}$$

$$\begin{bmatrix} \varepsilon_{\text{Car}} \\ \varepsilon_{\text{PT}} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \right), \; -1 \leq \rho \leq 1. \tag{11}$$

The Cholesky decomposition of the variance-covariance matrix is

$$\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix}'$$

so that the draws from this bivariate distribution can be generated from independent $N(0,1)$ draws r_1 and r_2 as

$$\begin{aligned} r_{\text{Car}} &= r_1 \\ r_{\text{PT}} &= r_1\rho + r_2\sqrt{1-\rho^2}. \end{aligned} \tag{12}$$

$$\Delta U = U_{\text{Car}} - U_{\text{PT}} \tag{13}$$

$$\begin{aligned} \tilde{U}_{\text{Car}} &= \beta_0 + \beta_1 \text{Time}_{\text{car}} + \varepsilon_{\text{Car}} \\ \tilde{U}_{\text{PT}} &= \beta_1 \text{Time}_{\text{PT}} + \varepsilon_{\text{PT}} \end{aligned} \tag{14}$$

Measurement Model

$$y = \begin{cases} 1 & (\text{Car}) \quad \text{if } \Delta U + \eta \geq 0, \eta \sim \text{Logistic}(0,1) \\ 0 & (\text{PT}) \quad \text{otherwise}. \end{cases}$$

If h^{*0}_{Car}, h^{*}_{Car} and h^{*}_{PT} are continuous latent response variables, the observed measures related to these variables through a threshold model is defined as follows:

$$h^0_{\text{Car}} = \begin{cases} 1 & \text{if} \quad -\infty < h^{*0}_{\text{Car}} \leq \tau_1 \\ 2 & \text{if} \quad \tau_1 < h^{*0}_{\text{Car}} \leq \tau_2 \\ 3 & \text{if} \quad \tau_2 < h^{*0}_{\text{Car}} \leq \tau_3 \\ 4 & \text{if} \quad \tau_3 < h^{*0}_{\text{Car}} \leq \tau_4 \\ 5 & \text{if} \quad \tau_4 < h^{*0}_{\text{Car}} \leq \infty \end{cases} \tag{15}$$

$$h_{\text{Car}} = \begin{cases} 1 & \text{if} \quad -\infty < h^{*}_{\text{Car}} \leq \tau_1 \\ 2 & \text{if} \quad \tau_1 < h^{*}_{\text{Car}} \leq \tau_2 \\ 3 & \text{if} \quad \tau_2 < h^{*}_{\text{Car}} \leq \tau_3 \\ 4 & \text{if} \quad \tau_3 < h^{*}_{\text{Car}} \leq \tau_4 \\ 5 & \text{if} \quad \tau_4 < h^{*}_{\text{Car}} \leq \infty \end{cases} \tag{16}$$

$$h_{\text{PT}} = \begin{cases} 1 & \text{if} \quad -\infty < h^{*}_{\text{PT}} \leq \tau_1 \\ 2 & \text{if} \quad \tau_1 < h^{*}_{\text{PT}} \leq \tau_2 \\ 3 & \text{if} \quad \tau_2 < h^{*}_{\text{PT}} \leq \tau_3 \\ 4 & \text{if} \quad \tau_3 < h^{*}_{\text{PT}} \leq \tau_4 \\ 5 & \text{if} \quad \tau_4 < h^{*}_{\text{PT}} \leq \infty \end{cases} \tag{17}$$

where τ_1, τ_2, τ_3 and τ_4 are threshold parameters.

Likelihood function The likelihood function for observation n is given by:

$$P_n = \int_{\varepsilon_{\text{PT}}} \int_{\varepsilon_{\text{Car}}} \left(\Lambda_1(y|\varepsilon_{\text{Car}}, \varepsilon_{\text{PT}}) P_2(h^0_{\text{Car}}|\varepsilon_{\text{Car}}) P_3(h_{\text{Car}}|\varepsilon_{\text{Car}}) P_4(h_{\text{PT}}|\varepsilon_{\text{PT}}) f_5(\varepsilon_{\text{Car}}, \varepsilon_{\text{PT}}) d\varepsilon_{\text{Car}} d\varepsilon_{\text{PT}} \right)$$

(18)

where

$$\Lambda_1(y|\varepsilon_{\text{Car}}, \varepsilon_{\text{PT}}) = \left(\frac{1}{1+e^{-\Delta U}}\right)^y \left(\frac{e^{-\Delta U}}{1+e^{-\Delta U}}\right)^{(1-y)} \tag{19}$$

$$\begin{aligned}
P_2(h^0_{\text{Car}}=1|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_1+\lambda_1\tilde{U}_{\text{Car}}}}\\
P_2(h^0_{\text{Car}}=2|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_2+\lambda_1\tilde{U}_{\text{Car}}}} - \tfrac{1}{1+e^{-\tau_1+\lambda_1\tilde{U}_{\text{Car}}}}\\
P_2(h^0_{\text{Car}}=3|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_3+\lambda_1\tilde{U}_{\text{Car}}}} - \tfrac{1}{1+e^{-\tau_2+\lambda_1\tilde{U}_{\text{Car}}}}\\
P_2(h^0_{\text{Car}}=4|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_4+\lambda_1\tilde{U}_{\text{Car}}}} - \tfrac{1}{1+e^{-\tau_3+\lambda_1\tilde{U}_{\text{Car}}}}\\
P_2(h^0_{\text{Car}}=5|\varepsilon_{\text{Car}}) &= 1 - \tfrac{1}{1+e^{-\tau_4+\lambda_1\tilde{U}_{\text{Car}}}}
\end{aligned} \tag{20}$$

$$\begin{aligned}
P_3(h_{\text{Car}}=1|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_1+\lambda_2\tilde{U}_{\text{Car}}}}\\
P_3(h_{\text{Car}}=2|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_2+\lambda_2\tilde{U}_{\text{Car}}}} - \tfrac{1}{1+e^{-\tau_1+\lambda_2\tilde{U}_{\text{Car}}}}\\
P_3(h_{\text{Car}}=3|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_3+\lambda_2\tilde{U}_{\text{Car}}}} - \tfrac{1}{1+e^{-\tau_2+\lambda_2\tilde{U}_{\text{Car}}}}\\
P_3(h_{\text{Car}}=4|\varepsilon_{\text{Car}}) &= \tfrac{1}{1+e^{-\tau_4+\lambda_2\tilde{U}_{\text{Car}}}} - \tfrac{1}{1+e^{-\tau_3+\lambda_2\tilde{U}_{\text{Car}}}}\\
P_3(h_{\text{Car}}=5|\varepsilon_{\text{Car}}) &= 1 - \tfrac{1}{1+e^{-\tau_4+\lambda_2\tilde{U}_{\text{Car}}}}
\end{aligned} \tag{21}$$

$$\begin{aligned}
P_4(h_{\text{PT}}=1|\varepsilon_{\text{PT}}) &= \tfrac{1}{1+e^{-\tau_1+\lambda_3\tilde{U}_{\text{PT}}}}\\
P_4(h_{\text{PT}}=2|\varepsilon_{\text{PT}}) &= \tfrac{1}{1+e^{-\tau_2+\lambda_3\tilde{U}_{\text{PT}}}} - \tfrac{1}{1+e^{-\tau_1+\lambda_3\tilde{U}_{\text{PT}}}}\\
P_4(h_{\text{PT}}=3|\varepsilon_{\text{PT}}) &= \tfrac{1}{1+e^{-\tau_3+\lambda_3\tilde{U}_{\text{PT}}}} - \tfrac{1}{1+e^{-\tau_2+\lambda_3\tilde{U}_{\text{PT}}}}\\
P_4(h_{\text{PT}}=4|\varepsilon_{\text{PT}}) &= \tfrac{1}{1+e^{-\tau_4+\lambda_3\tilde{U}_{\text{PT}}}} - \tfrac{1}{1+e^{-\tau_3+\lambda_3\tilde{U}_{\text{PT}}}}\\
P_4(h_{\text{PT}}=5|\varepsilon_{\text{PT}}) &= 1 - \tfrac{1}{1+e^{-\tau_4+\lambda_3\tilde{U}_{\text{PT}}}}
\end{aligned} \tag{22}$$

$$f_5(\varepsilon_{\text{Car}}, \varepsilon_{\text{PT}}) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left(\varepsilon^2_{\text{Car}} + \varepsilon^2_{\text{PT}} - 2\rho\varepsilon_{\text{Car}}\varepsilon_{\text{PT}}\right)\right) \tag{23}$$

This model can be specified with Python as follows:

```
# Import modules
from biogeme import *
from headers import *

dataFile = "maya.dat"

rho =  Beta( 'rho', 0, -1, 1, 0, 'correlation')
beta0 =  Beta( 'beta0', 0, -10000, 10000, 0, 'beta 0')
beta1 =  Beta( 'beta1', 0, -10000, 10000, 0, 'beta 1')
beta2 =  Beta( 'beta2', 0, -10000, 10000, 0, 'beta 2')
tau1 =  Beta( 'tau1', 0, -10000, 10000, 0, 'tau 1')
```

```
delta1 =  Beta( ’delta1’, 0, 0, 10000, 0, ’delta 1’)
delta2 =  Beta( ’delta2’, 0, 0, 10000, 0, ’delta 2’)
delta3 =  Beta( ’delta3’, 0, 0, 10000, 0, ’delta 3’)
lambda1 =  Beta( ’lambda1’, 0, -10000, 10000, 0, ’lambda 1’)
lambda2 =  Beta( ’lambda2’, 0, -10000, 10000, 0, ’lambda 2’)
lambda3 =  Beta( ’lambda3’, 0, -10000, 10000, 0, ’lambda 3’)

epsilonCar = bioNormal(’r1’)
epsilonPT = rho * bioNormal(’r1’) + (1-rho*rho)**0.5 * bioNormal(’r2’)

Ucar = beta0 + beta1 * FTTime + beta2 * FTCost / Inc + epsilonCar
Upt = beta1 * PTTime + beta2 * PTCost / Inc + epsilonPT

deltaU = Ucar - Upt

choice = (Mobil_Nov <> 1)

Pcar = 1 / (1 + exp(-deltaU))
PPT = exp(-deltaU)/(1+exp(-deltaU))

P = {0: Pcar, 1: PPT}
l = Elem(P,choice)

UtildeCar = beta0 + beta1 * FTTime + epsilonCar
UtildePT = beta1 * PTTime + epsilonPT

tau2 = tau1 + delta1
tau3 = tau2 + delta2
tau4 = tau3 + delta3

P2 = {
    1: 1/(1+exp(-tau1 + lambda1 * UtildeCar)),
    2: (1/(1+exp(-tau2 +lambda1 * UtildeCar)))
       -(1/(1+exp(-tau1 + lambda1 * UtildeCar))),
    3: (1/(1+exp(-tau3 +lambda1 * UtildeCar)))
       -(1/(1+exp(-tau2 + lambda1 * UtildeCar))),
    3: (1/(1+exp(-tau3 +lambda1 * UtildeCar)))
       -(1/(1+exp(-tau2 + lambda1 * UtildeCar))),
    4: (1/(1+exp(-tau4 +lambda1 * UtildeCar)))
       -(1/(1+exp(-tau3 + lambda1 * UtildeCar))),
    5: 1-(1/(1+exp(-tau4 + lambda1 * UtildeCar)))
    }
```

```
P3 = {
    1: 1/(1+exp(-tau1 + lambda2 * UtildeCar)),
    2: (1/(1+exp(-tau2 +lambda2 * UtildeCar)))
       -(1/(1+exp(-tau1 + lambda2 * UtildeCar))),
    3: (1/(1+exp(-tau3 +lambda2 * UtildeCar)))
       -(1/(1+exp(-tau2 + lambda2 * UtildeCar))),
    3: (1/(1+exp(-tau3 +lambda2 * UtildeCar)))
       -(1/(1+exp(-tau2 + lambda2 * UtildeCar))),
    4: (1/(1+exp(-tau4 +lambda2 * UtildeCar)))
       -(1/(1+exp(-tau3 + lambda2 * UtildeCar))),
    5: 1-(1/(1+exp(-tau4 + lambda2 * UtildeCar)))
    }

P4 = {
    1: 1/(1+exp(-tau1 + lambda3 * UtildePT)),
    2: (1/(1+exp(-tau2 +lambda3 * UtildePT)))
       -(1/(1+exp(-tau1 + lambda3 * UtildePT))),
    3: (1/(1+exp(-tau3 +lambda3 * UtildePT)))
       -(1/(1+exp(-tau2 + lambda3 * UtildePT))),
    3: (1/(1+exp(-tau3 +lambda3 * UtildePT)))
       -(1/(1+exp(-tau2 + lambda3 * UtildePT))),
    4: (1/(1+exp(-tau4 +lambda3 * UtildePT)))
       -(1/(1+exp(-tau3 + lambda3 * UtildePT))),
    5: 1-(1/(1+exp(-tau4 + lambda3 * UtildePT)))
    }

term = l * Elem(P2,PreCarSa) * Elem(P3,PstCarSa) * Elem(P4,PstPTSa)

drawIterator('drawIter')
rowIterator('obsIter', Datafile(dataFile))

prob = Sum(term,'drawIter')
loglike = Sum(log(prob),'obsIter')
BIOGEME_OBJECT.FORMULA = loglike
BIOGEME_OBJECT.DRAWS = 1000
```

3 Inside Biogeme

The new version of Biogeme is divided into two parts. The first part consists of C++ classes and methods, and deals with the estimation algorithm. The second part consists of several Python modules which provide modeling elements allowing the user to specify the model and the likelihood function. Some of these modules contain predefined likelihood functions and models that can be directly employed. This part is very modular and extensible so that the user can develop her own Python modules to add to the existing ones. Although no major computation is performed in Python, a consistency check of the model is performed before its actual loading.

The estimation is performed in 3 main steps as it is presented in figure 1. First, the model and the likelihood function are parsed from a specification file. Then, dedicated C++ code for the model is generated, that is C++ code for the likelihood function and its derivatives. The motivation of this step is mainly for efficiency purpose when running the optimization procedure. Finally, the estimation itself is performed by an optimizer.

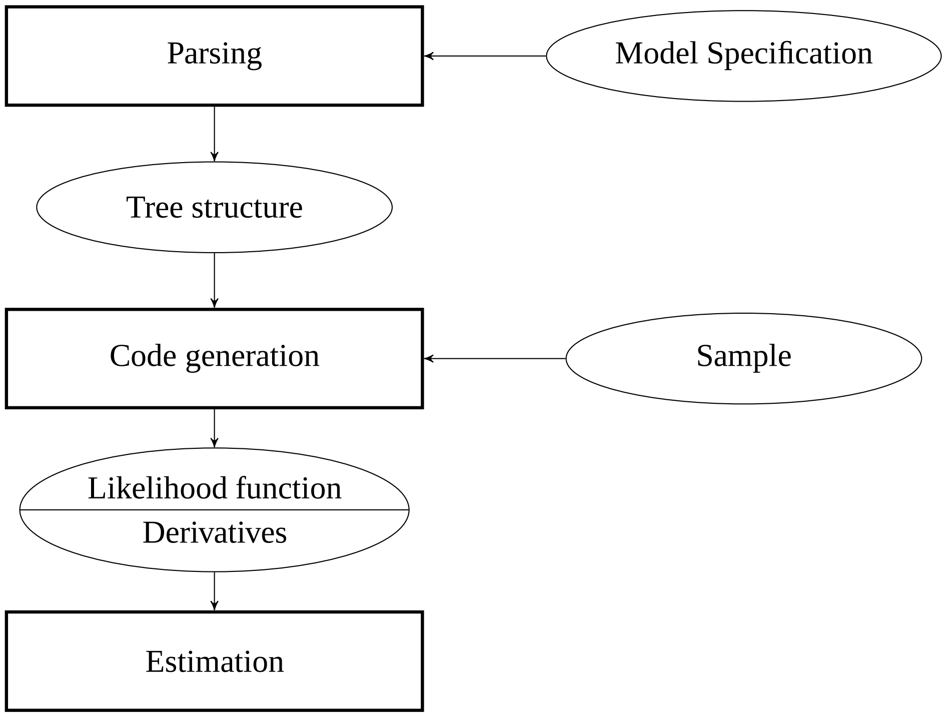

Figure 1: Steps of the estimation procedure in Biogeme

3.1 Parsing the model specification

The model specification file is read by the Biogeme parser which stores the likelihood function in a tree structure representing the mathematical formulation. Each node of the tree is associated with one of the elements described in section 2.1. The number of the children nodes is equal to the number of the operands involved in the corresponding operation or function. The leaves of the tree are constant numbers and literals (parameters or variables). Hence an expression can be evaluated by setting values to literals. For instance, the likelihood function for the logit model formulated in equation (3) is stored as illustrated in figure 2.

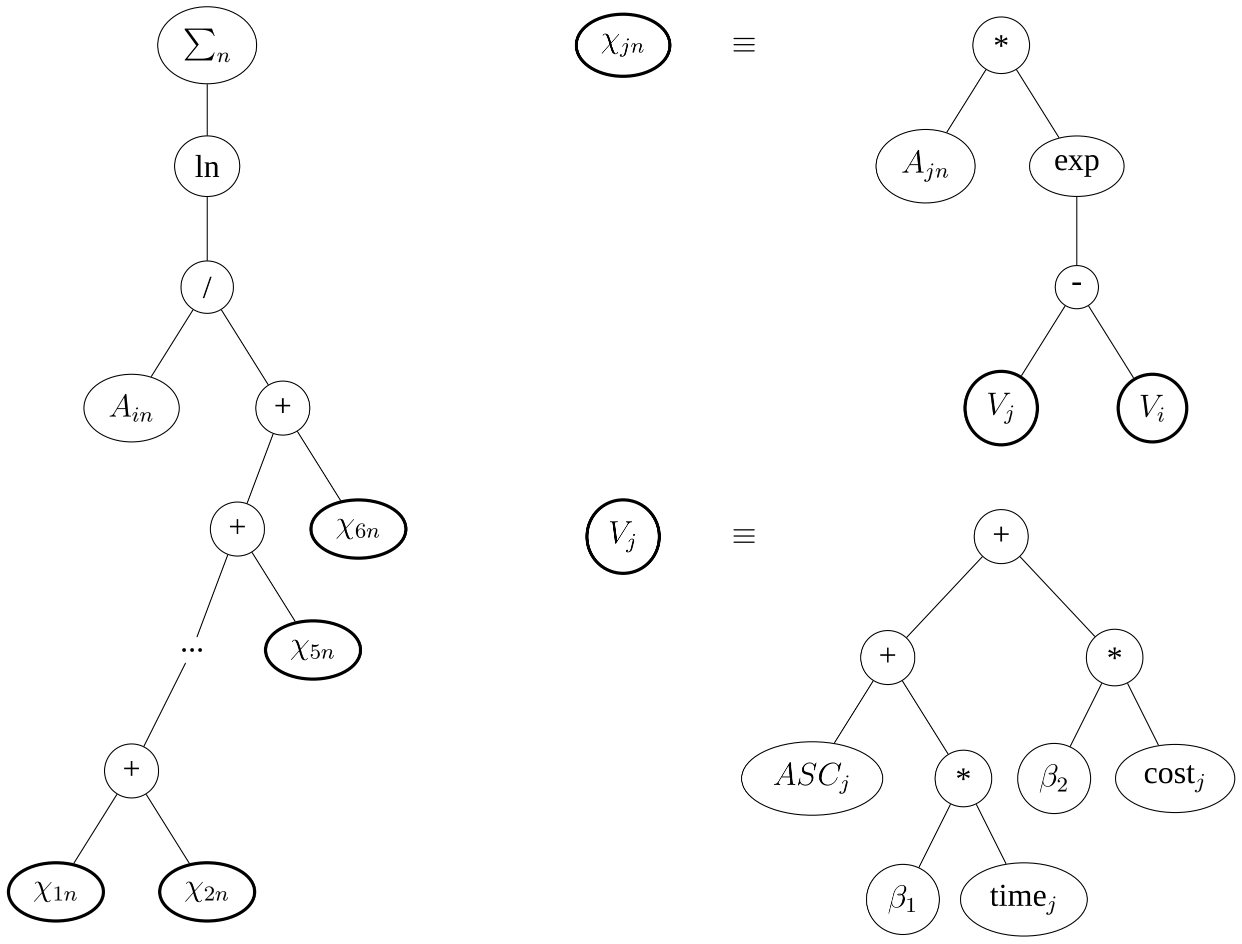

Figure 2: Tree representation in Biogeme of the likelihood function (equation (3)) associated with the logit model in section 2.2.1

Each node is also able to provide the derivative of its associated expression. This computation is executed recursively through the children nodes of the subtree using chain rules differentiation.

3.2 Code generation and Optimization

For the sake of efficiency, the C++ code for the likelihood function is produced from the model specification and from a data file. The C++ code for derivatives, needed by the optimization algorithms, is also automatically generated. This approach allows us to exploit multithreading mechanism in programming and to take advantage of multiprocessor machines.

Thanks to the object-oriented design of Biogeme, each node can easily generate its corresponding C++ code. Generating the code for derivatives is also straightforward since the derivative expression is provided by the current node. Note that the computation of higher-order derivatives is also possible.

Regarding the estimation part, the parameters to be estimated are indicated to the optimization package as well as the function to be optimized and its derivatives. Three optimization algorithms are implemented in Biogeme as in the previous version: CFSQP, SolvOpt, DONLP2. Bierlaire (2003) provides more details and the comparison of execution time of these three algorithms.

4 Status of the development

The first stable version of the new version of Biogeme is still under development. The specification and the estimation of the models are being validated. As of today, most of the examples distributed with Biogeme have been validated. The new version produces the exact same results.

The new design of Biogeme, allowing the user to define explicitly her model and the likelihood function, implies a loss of efficiency compared with the previous version. When the models are fully validated, we will work on the improvement of the efficiency, mainly by generating more efficient C++ code dedicated to the model. The handling of multithreading is also in progress. Consequently, minor change may occur in the syntax in the future.

The new design of the software opens interesting perspectives for future developments. In particular, we can imagine creating codes in other languages for the likelihood function, like Matlab or Gauss.

We expect this new version of Biogeme to allow us (and the research community when the software will be packaged for distribution) to investigate new models, including state of the art specifications, appropriate for the complex phenomena we plan to analyze.

Acknowledgements

We are very grateful to Maya Abou Zeid for making her model and data available to us. We thank also Biogeme users for their continuous remarks and suggestions for improving the software.

References

Abou Zeid, M. (2009) Measuring and modeling travel and activity well-being, Ph.D. Thesis, Massachusetts Institute of Technology.

Ben-Akiva, M., D. McFadden, K. Train, J. Walker, C. Bhat, M. Bierlaire, D. Bolduc, A. Boersch-Supan, D. Brownstone, D. Bunch, A. Daly, A. de Palma, D. Gopinath, A. Karlstrom and M. A. Munizaga (2002) Hybrid choice models: Progress and challenges, *Marketing Letters*, **13** (3) 163–175, August 2002.

Bierlaire, M. (2003) BIOGEME: a free package for the estimation of discrete choice models, paper presented at *Swiss Transport Research Conference*.

McFadden, D. (1978) Modeling the choice of residential location, in A. Karlqvist (ed.) *Spatial Interaction Theory and Residential Location*, 75–96, North-Holland, Amsterdam.

Train, K. (2003) *Discrete Choice Methods with Simulation*, Cambridge University Press. Http://emlab.berkeley.edu/books/choice.html.

Walker, J. L. (2001) Extended discrete choice models: integrated framework, flexible error structures, and latent variables, Ph.D. Thesis, Massachusetts Institute of Technology.