Calculating indicators with PandasBiogeme

Michel Bierlaire

December 23, 2018

Report TRANSP-OR 181223
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
transp-or.epfl.ch

Series on Biogeme

This document is an updated version of Bierlaire (2017), adapted for PandasBiogeme.

The package Biogeme (`biogeme.epfl.ch`) is designed to estimate the parameters of various models using maximum likelihood estimation. It is particularly designed for discrete choice models. But it can also be used to extract indicators from an estimated model. In this document, we describe how to calculate some indicators particularly relevant in the context of discrete choice models: market shares, revenues, elasticities, and willingness to pay. Clearly, the use of the software is not restricted to these indicators, neither to choice models. But these examples illustrate most of the capabilities.

We assume that the reader is already familiar with discrete choice models, and has successfully installed PandasBiogeme. Note that PythonBiogeme and PandasBiogeme have a very similar syntax. The difference is that PythonBiogeme is an independent software package written in C++, and using the Python language for model specification. PandasBiogeme is a genuine Python package written in Python and C++, that relies on the Pandas library for the management of the data. The syntax for model specification is almost identical, but there are slight differences. We refer the reader to Bierlaire (2018) for a detailed discussion of these differences. This document has been written using PandasBiogeme 3.1, but should remain valid for future versions.

1 The model

See 01nestedEstimation.py in Section A.1

We consider a case study involving a transportation mode choice model, using revealed preference data collected in Switzerland in 2009 and 2010 (see Atasoy et al., 2013). The model is a nested logit model with 3 alternatives: *public transportation*, *car* and *slow modes*. The utility functions are defined as:

```
V_PT = BETA_TIME_FULLTIME * TimePT_scaled * fulltime +
       BETA_TIME_OTHER * TimePT_scaled * notfulltime +
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR +
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime +
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime +
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM +
       BETA_DIST_MALE * distance_km_scaled * male +
       BETA_DIST_FEMALE * distance_km_scaled * female +
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender
```

where ASC_CAR, ASC_SM, BETA_TIME_FULLTIME, BETA_TIME_OTHER, BETA_DIST_MALE, BETA_DIST_FEMALE, BETA_DIST_UNREPORTED, BETA_COST, are parameters to be estimated, TimePT_scale, MarginalCostPT_scaled, TimeCar_scale, CostCarCHF_scale, distance_km_scale are attributes and fulltime, notfulltime, male, female, unreportedGender are socio-economic characteristics. The two alternatives "public transportation" and "slow modes" are grouped into a nest. The complete specification is available in the file 01nestedEstimation.py reported in Section A.1. We refer the reader to Bierlaire (2018) for an introduction to the syntax.

The parameters are estimated using PandasBiogeme. Their values are reported in Table 1.

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.261	0.100	2.61	0.01
2	ASC_SM	0.0591	0.217	0.273	0.785
3	BETA_COST	-0.716	0.138	-5.18	0.00
4	BETA_DIST_FEMALE	-0.831	0.193	-4.31	0.00
5	BETA_DIST_MALE	-0.686	0.161	-4.27	0.00
6	BETA_DIST_UNREPORTED	-0.703	0.196	-3.58	0.000344
7	BETA_TIME_FULLTIME	-1.60	0.333	-4.80	0.00
8	BETA_TIME_OTHER	-0.577	0.296	-1.95	0.0515
9	NEST_NOCAR	1.53	0.306	1.73[a]	0.08

Summary statistics

Number of observations = 1906

Number of excluded observations = 359

Number of estimated parameters = 9

$$\begin{aligned}
\mathcal{L}(\beta_0) &= -2093.955 \\
\mathcal{L}(\hat{\beta}) &= -1298.498 \\
-2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 1590.913 \\
\rho^2 &= 0.380 \\
\bar{\rho}^2 &= 0.376
\end{aligned}$$

[a] t-test against 1

Table 1: Nested logit model: estimated parameters

2 Market shares and revenues

See 02nestedSimulation.py in Section A.2

Once the model has been estimated, it must be used to derive useful indicators. PandasBiogeme provides a simulation feature for this purpose. We start by describing how to calculate market shares using sample enumeration. It is necessary to have a sample of individuals from the population. For each of them, the value of each of the variables involved in the model must be known. Note that it is possible to use the same sample that was used for estimation, but only if it contains revealed preferences data. Indeed, the calculation of indicators require real values for the variables, not values that have been designed and engineered for the sake of estimating parameters, like in stated preferences data. It is the procedure used in this document.

More formally, consider a choice model $P_n(i|x_n, \mathcal{C}_n)$ providing the probability that individual n chooses alternative i within the choice set $\mathcal{C}_n$, given the explanatory variables x_n. In order to calculate the market shares in the population of size N, a sample of N_s individuals is drawn. As it is rarely possible to draw from the population with equal sampling probability, it is assumed that stratified sampling has been used, and that each individual n in the sample is associated with a weight w_n correcting for sampling biases. The weights are normalized such that

$$N_s = \sum_{n=1}^{N_s} w_n. \tag{1}$$

An estimator of the market share of alternative i in the population is

$$W_i = \frac{1}{N_s} \sum_{n=1}^{N_s} w_n P_n(i|x_n, \mathcal{C}_n). \tag{2}$$

If the alternative i involves a price variable p_{in}, the expected revenue generated by i is

$$R_i = \frac{N}{N_s} \sum_{n=1}^{N_s} w_n p_{in} P_n(i|x_n, p_{in}, \mathcal{C}_n). \tag{3}$$

In practice, the size of the population is rarely known, and the above quantity is used only in the context of price optimization. In this case, the factor N/N_s can be omitted.

To calculate (2) and (3) with PandasBiogeme, a specification file must be prepared. We describe how to build this file. We name the resulting file 02nestedSimulation.py, reported in Section A.2:

1. Start with a copy of the model estimation file 01nestedEstimation.py.

2. Keep the part that builds the database, and defines the model specification. In this example, we keep the first 106 lines.

3. Define the choice probability for each alternative:

```
prob_pt = models.nested(V,av,nests,0)
prob_car = models.nested(V,av,nests,1)
prob_sm = models.nested(V,av,nests,2)
```

4. Define the quantities that must be simulated in a dictionary. In this case, we calculate, for each individual

   - the normalized weights, that verify (1),
   - the choice probability of each alternative,
   - the revenues generated by public transportation.

   This is specified using the following statement:

```
simulate = {'weight': normalizedWeight,
            'Prob. car': prob_car,
            'Prob. public transportation': prob_pt,
            'Prob. slow modes':prob_sm,
            'Revenue public transportation':prob_pt * MarginalCostPT}
```

   Each entry of this dictionary corresponds to a quantity that will be calculated for each observation in the sample. The key of the entry is a string, that will identify the column in the Pandas data structure that will be generated. The value must be a valid formula describing the calculation.

5. We provide both the database and the formulas to be simulated to Biogeme:

```
biogeme  = bio.BIOGEME(database,simulate)
biogeme.modelName = "02nestedSimulation"
```

6. Now, we need to retrieve the values of the parameters that were calculated at the estimation stage. First, we obtain the names of the parameters that we need for the simulation. Note that it may not be exactly the same list as for estimation.

```
betas = biogeme.freeBetaNames
```

   Then, we read the results of the estimation from the "pickle" file:

```
results = res.bioResults(pickleFile='01nestedEstimation.pickle')
```

Now, we can retrieve the estimated values:

```
betaValues = results.getBetaValues()
```

7. We now perform the simulation itself:

   ```
   simulatedValues = biogeme.simulate(betaValues)
   ```

   It generates a Pandas data frame. Each row corresponds to an observation in the Biogeme database, and each column corresponds to a quantity requested above.

8. We can also calculate confidence intervals on these quantities, using simulation. First, we draw 100 realizations of the maximum likelihood estimator, and extract the parameters that we need for simulation (identified by the list betas):

   ```
   b = results.getBetasForSensitivityAnalysis(betas, size=100)
   ```

   And we calculate 90% confidence intervals:

   ```
   left, right = biogeme.confidenceIntervals(b,0.9)
   ```

   The two Pandas data frames have the same structure as the simulated values, and contain the left and right bounds of the intervals, respectively.

9. We can now calculate the market shares, and the confidence intervals. First, we add a column to the data frames for the weighted probabilities involved in (2):

   ```
   simulatedValues['Weighted prob. car'] = \
       simulatedValues['weight'] * simulatedValues['Prob. car']
   left['Weighted prob. car'] = left['weight'] * left['Prob. car']
   right['Weighted prob. car'] = right['weight'] * right['Prob. car']
   ```

   The market shares as well as the confidence intervals, are simply the mean of these new columns:

   ```
   marketShare_car = simulatedValues['Weighted prob. car'].mean()
   marketShare_car_left = left['Weighted prob. car'].mean()
   marketShare_car_right = right['Weighted prob. car'].mean()
   ```

   The market shares of the other models are calculated similarly.

10. For the revenues, we use (3) with $\mathsf{N} = \mathsf{N}_s$, for the sake of the example. In this case, the sum of the new column is calculated instead of the mean.

```
revenues_pt = (simulatedValues['Revenue public transportation'] *
               simulatedValues['weight']).sum()
revenues_pt_left = (left['Revenue public transportation'] *
                    left['weight']).sum()
revenues_pt_right = (right['Revenue public transportation'] *
                     right['weight']).sum()
```

Note that, in the above code above, we did not include the line continuation character \. Indeed, Python automatically implies line continuation inside parentheses, brackets and braces.

The output of the Python script is as follows:

```
Running 02nestedSimulation.py...
Number of males:   943
Number of females: 871
Unreported gender: 92
Market share for car: 65.3% [60.4%,68.7%]
Market share for PT: 28.1% [23.8%,32.4%]
Market share for slow modes: 6.6% [4.7%,10.6%]
Revenues for PT: 3018.431 [2485.425,3771.998]
```

3 Elasticities

Consider now one of the variables involved in the model, for instance x_{ink}, the kth variable associated by individual n with alternative i. The objective is to anticipate the impact of a change of the value of this variable on the choice of individual n, and subsequently on the market share of alternative i.

3.1 Point elasticities

If the variable is continuous, we assume that the relative (infinitesimal) change of the variable is the same for every individual in the population, that is

$$\frac{\partial x_{ink}}{x_{ink}} = \frac{\partial x_{ipk}}{x_{ipk}} = \frac{\partial x_{ik}}{x_{ik}}, \tag{4}$$

where

$$x_{ik} = \frac{1}{N_s} \sum_{n=1}^{N_s} x_{ink}. \tag{5}$$

The *disaggregate direct point elasticity* of the model with respect to the variable x_{ink} is defined as

$$E^{P_n(i)}_{x_{ink}} = \frac{\partial P_n(i|x_n, \mathcal{C}_n)}{\partial x_{ink}} \frac{x_{ink}}{P_n(i|x_n, \mathcal{C}_n)}. \tag{6}$$

It is called

- disaggregate, because it refers to the choice model related to a specific individual,
- direct, because it measures the impact of a change of an attribute of alternative i on the choice probability of the same alternative,
- point, because we consider an infinitesimal change of the variable.

The *aggregate direct point elasticity* of the model with respect to the average value x_{ik} is defined as

$$E^{W_i}_{x_{ik}} = \frac{\partial W_i}{\partial x_{ik}} \frac{x_{ik}}{W_i}. \tag{7}$$

Using (2), we obtain

$$E^{W_i}_{x_{ik}} = \frac{1}{N_s} \sum_{n=1}^{N_s} w_n \frac{\partial P_n(i|x_n, \mathcal{C}_n)}{\partial x_{ik}} \frac{x_{ik}}{W_i}. \tag{8}$$

From (4), we obtain

$$E^{W_i}_{x_{ik}} = \frac{1}{N_s} \sum_{n=1}^{N_s} w_n \frac{\partial P_n(i|x_n, \mathcal{C}_n)}{\partial x_{ink}} \frac{x_{ink}}{W_i} = \frac{1}{N_s} \sum_{n=1}^{N_s} w_n E^{P_n(i)}_{x_{ink}} \frac{P_n(i|x_n, \mathcal{C}_n)}{W_i}, \tag{9}$$

where the second equation is derived from (6). Using (2) again, we obtain

$$E^{W_i}_{x_{ik}} = \sum_{n=1}^{N_s} E^{P_n(i)}_{x_{ink}} \frac{w_n P_n(i|x_n, \mathcal{C}_n)}{\sum_{n=1}^{N_s} w_n P_n(i|x_n, \mathcal{C}_n)}. \tag{10}$$

This equation shows that the calculation of aggregate elasticities involves a weighted sum of disaggregate elasticities. However, the weight is not w_n as for the market share, but a normalized version of $w_n P_n(i|x_n, \mathcal{C}_n)$.

The *disaggregate cross point elasticity* of the model with respect to the variable x_{jnk} is defined as

$$E^{P_n(i)}_{x_{jnk}} = \frac{\partial P_n(i|x_n, \mathcal{C}_n)}{\partial x_{jnk}} \frac{x_{jnk}}{P_n(i|x_n, \mathcal{C}_n)}. \tag{11}$$

It is called *cross* elasticity because it measures the sensitivity of the model for alternative i with respect to a modification of the attribute of another alternative.

3.2 Arc elasticities

A similar derivation can be done for arc elasticities. In this case, the relative change of the variable is not infinitesimal anymore. The idea is to analyze a before/after scenario. The variable x_{ink} in the before scenario becomes $x_{ink} + \Delta x_{ink}$ in the after scenario. As above, we assume that the relative change of the variable is the same for every individual in the population, that is

$$\frac{\Delta x_{ink}}{x_{ink}} = \frac{\Delta x_{ipk}}{x_{ipk}} = \frac{\Delta x_{ik}}{x_{ik}}, \tag{12}$$

where x_{ik} is defined by (5). The *disaggregate direct arc elasticity* of the model with respect to the variable x_{ink} is defined as

$$E^{P_n(i)}_{x_{ink}} = \frac{\Delta P_n(i|x_n, \mathcal{C}_n)}{\Delta x_{ink}} \frac{x_{ink}}{P_n(i|x_n, \mathcal{C}_n)}. \tag{13}$$

The *aggregate direct arc elasticity* of the model with respect to the average value x_{ik} is defined as

$$E^{W_i}_{x_{ik}} = \frac{\Delta W_i}{\Delta x_{ik}} \frac{x_{ik}}{W_i}. \tag{14}$$

The two quantities are also related by (10), following the exact same derivation as for the point elasticity.

3.3 Using PandasBiogeme for point elasticities

See 03 nestedElasticities .py in Section A.3

The calculation of (6) involves derivatives. For simple models such as logit, the analytical formula of these derivatives can easily be derived. However, their derivation for advanced models can be tedious. It is common to make mistakes in the derivation itself, and even more common to make mistakes in the implementation. Therefore, PandasBiogeme provides an operator that calculates the derivative of a formula. It is illustrated in the file 03 nestedElasticities .py, reported in Section A.3. We describe here the calculation of the elasticity of the demand for public transportation with respect to the travel time of public transportation. Other elasticities are calculated similarly. The calculation of the disaggregate elasticities for each individual by PandasBiogeme are performed using the following statement:

```
direct_elas_pt_time = \
  Derive(prob_pt,'TimePT') * TimePT / prob_pt
```

and adding the corresponding entry in the simulation dictionary:

```
simulate = {'weight': normalizedWeight,
            'Prob. car': prob_car,
            'Prob. public transportation': prob_pt,
            'Prob. slow modes':prob_sm,
            'direct_elas_pt_time':direct_elas_pt_time,
            'direct_elas_pt_cost':direct_elas_pt_cost,
            'direct_elas_car_time':direct_elas_car_time,
            'direct_elas_car_cost':direct_elas_car_cost,
            'direct_elas_sm_dist':direct_elas_sm_dist}
```

The above syntax should be self-explanatory. But there is an important aspect to take into account. In the context of the estimation of the parameters of the model, the variables are often scaled in order to improve the numerical properties of the likelihood function, using statements like

```
TimePT_scaled = DefineVariable('TimePT_scaled', TimePT / 200 )
```

or

```
TimePT_scaled = TimePT / 200
```

The DefineVariable operator is designed to preprocess the data file, and can be seen as a way to add another column in the data file, defining a new variable. However, if it is used, the relationship between the new variable and the original one is lost, for the sake of computational speed. Therefore, prob_pt depends on TimePT_scaled, but not on TimePT. Therefore, the result of Derive(prob_pt,'TimePT') is zero.

Consequently, when you need to calculate derivatives, you may want to replace statements like

```
TimePT_scaled = DefineVariable('TimePT_scaled', TimePT / 200 )
```

by

```
TimePT_scaled  = TimePT / 200
```

in order to maintain the analytical structure of the formula to be derived.

The aggregate point elasticities can be obtained by aggregating the disaggregate elasticities, using (10). This requires the calculation of the normalization factors

$$\sum_{n=1}^{N_s} w_n P_n(i|x_n, \mathcal{C}_n). \tag{15}$$

This is performed with the following code, that first creates new columns in the Pandas data frame, and then calculate their sum:

```
simulatedValues['Weighted prob. PT'] = \
  simulatedValues['weight'] * simulatedValues['Prob. public transportation']
denominator_pt = simulatedValues['Weighted prob. PT'].sum()
```

The calculation of the aggregate direct elasticity (10) is performed as follows:

```
direct_elas_term_pt_time = (simulatedValues['Weighted prob. PT']
  * simulatedValues['direct_elas_pt_time'] / denominator_pt).sum()
```

Note that, in this case, we did not explicitly create a new column before calculating the sum. Looking at the formula, we have

- the disaggregate elasticity: simulatedValues['direct_elas_pt_time'] $= E^{P_n(i)}_{x_{ink}}$,
- the numerator: simulatedValues['Weighted prob. PT'] $= w_n P_n(i|x_n, \mathcal{C}_n)$, calculated previously,
- the denominator denominator_pt $= \sum_{n=1}^{N_s} w_n P_n(i|x_n, \mathcal{C}_n)$, calculated previously.

The output of the Python script is as follows:

```
Running 03nestedElasticities.py...
Number of males:   943
Number of females: 871
Unreported gender: 92
Aggregate direct elasticity of car wrt time: -0.0441
Aggregate direct elasticity of car wrt cost: -0.0906
Aggregate direct elasticity of PT wrt time: -0.274
Aggregate direct elasticity of PT wrt cost: -0.32
Aggregate direct elasticity of SM wrt distance: -1.09
```

3.4 Using PandasBiogeme for cross elasticities

See 04nestedElasticities.py in Section A.4

The calculation of (11) is performed in a similar way as the direct elasticities (6), using the following statements:

```
cross_elas_pt_time = Derive(prob_pt,'TimeCar') * TimeCar / prob_pt
```

The output of the Python script is the following:

```
Running 04nestedElasticities.py...
Number of males:   943
Number of females: 871
Unreported gender: 92
Aggregate cross elasticity of car wrt time: 0.107
Aggregate cross elasticity of car wrt cost: 0.123
Aggregate cross elasticity of PT wrt car time: 0.0953
Aggregate cross elasticity of PT wrt car cost: 0.2
```

Note that these values are now positive. Indeed, when the travel time or travel cost of a competing mode increase, the market share increases.

3.5 Using PandasBiogeme for arc elasticities

See 05nestedElasticities.py in Section A.5

Arc elasticities require a before and after scenarios. In this case, we calculate the sensitivity of the market share of the slow modes alternative when there is a uniform increase of 1 kilometer.

The "before" scenario is represented by the same model as above. The after scenario is modeled using the following statements:

```
delta_dist = 1
distance_km_scaled_after = (distance_km + delta_dist) / 5
V_SM_after = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled_after * male + \
       BETA_DIST_FEMALE * distance_km_scaled_after * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled_after * unreportedGender
V_after = {0: V_PT,
           1: V_CAR,
           2: V_SM_after}
prob_sm_after = nested(V_after,av,nests,2)
```

Then, the arc elasticity is calculated as

```
elas_sm_dist = \
 (prob_sm_after - prob_sm) * distance_km / (prob_sm * delta_dist)
```

The output of the Python script is as follows:

```
Running 05nestedElasticities.py...
Number of males:   943
Number of females: 871
Unreported gender: 92
Aggregate direct elasticity of slow modes wrt distance: -1.01
```

4 Willingness to pay

See 06nestedWTP.py in Section A.6

If the model contains a cost or price variable (like in this example), it is possible to analyze the trade-off between any variable and money. This reflects the willingness of the decision maker to pay for a modification of another variable of the model. A typical example in transportation is the *value of time*, that is the amount of money a traveler is willing to pay in order to decrease her travel time.

Let c_{in} be the cost of alternative i for individual n. Let x_{ink} be the value of another variable of the model. Let $V_{in}(c_{in}, x_{ink})$ be the value of the utility function. Consider a scenario where the variable of interest takes the value

$x_{ink}+\delta^x_{ink}$. We denote by δ^c_{in} the additional cost that would achieve the same utility, that is

$$V_{in}(c_{in}+\delta^c_{in}, x_{ink}+\delta^x_{ink}) = V_{in}(c_{in}, x_{ink}). \tag{16}$$

The willingness to pay to increase the value of x_{ink} is defined as the additional cost per unit of x, that is

$$\delta^c_{in}/\delta^x_{ink}, \tag{17}$$

and is obtained by solving Equation (16). If x_{ink} and c_{in} appear linearly in the utility function, that is if

$$V_{in}(c_{in}, x_{ink}) = \beta_c c_{in} + \beta_x x_{ink} + \cdots, \tag{18}$$

and

$$V_{in}(c_{in}+\delta^c_{in}, x_{ink}+\delta^x_{ink}) = \beta_c(c_{in}+\delta^c_{in}) + \beta_x(x_{ink}+\delta^x_{ink}) + \cdots. \tag{19}$$

Therefore, (17) is

$$\delta^c_{in}/\delta^x_{ink} = -\beta_x/\beta_c. \tag{20}$$

If x_{ink} is a continuous variable, and if V_{in} is differentiable in x_{ink} and c_{in}, we can invoke Taylor's theorem in (16):

$$\begin{aligned} V_{in}(c_{in}, x_{ink}) &= V_{in}(c_{in}+\delta^c_{in}, x_{ink}+\delta^x_{ink}) \\ &\approx V_{in}(c_{in}, x_{ink}) + \delta^c_{in}\frac{\partial V_{in}}{\partial c_{in}}(c_{in}, x_{ink}) + \delta^x_{ink}\frac{\partial V_{in}}{\partial x_{ink}}(c_{in}, x_{ink}) \end{aligned} \tag{21}$$

Therefore, the willingness to pay is equal to

$$\frac{\delta^c_{in}}{\delta^x_{ink}} = -\frac{(\partial V_{in}/\partial x_{ink})(c_{in}, x_{ink})}{(\partial V_{in}/\partial c_{in})(c_{in}, x_{ink})}. \tag{22}$$

Note that if x_{ink} and c_{in} appear linearly in the utility function, (22) is the same as (20). If we consider now a scenario where the variable under interest takes the value $x_{ink}-\delta^x_{ink}$, the same derivation leads to the willingness to pay to *decrease* the value of x_{ink}:

$$\frac{\delta^c_{in}}{\delta^x_{ink}} = \frac{(\partial V_{in}/\partial x_{ink})(c_{in}, x_{ink})}{(\partial V_{in}/\partial c_{in})(c_{in}, x_{ink})}. \tag{23}$$

The calculation of the value of time corresponds to such a scenario:

$$\frac{\delta^c_{in}}{\delta^t_{in}} = \frac{(\partial V_{in}/\partial t_{in})(c_{in}, t_{in})}{(\partial V_{in}/\partial c_{in})(c_{in}, t_{in})} = \frac{\beta_t}{\beta_c}, \tag{24}$$

where the last equation assumes that V is linear in these variables. Note that, in this special case of linear utility functions, the value of time is constant across individuals, and is also independent of δ^t_{in}. This is not true in general.

The calculation of (23) involves the calculation of derivatives. It is done in PandasBiogeme using the following statements:

```
WTP_PT_TIME = Derive(V_PT,'TimePT') / Derive(V_PT,'MarginalCostPT')
WTP_CAR_TIME = Derive(V_CAR,'TimeCar') / Derive(V_CAR,'CostCarCHF')
```

The full specification file can be found in Section A.6. The output of the Python script is as follows:

```
Running 06nestedWTP.py...
Number of males:    943
Number of females: 871
Unreported gender: 92
Average WTP for car: 3.96 CI:[1.81,6.65]
Unique values:  ['2.42', '6.69']
WTP car for workers: 6.69 CI:[4.06,10.2]
WTP car for females: 3.17 CI:[1.16,5.62]
WTP car for males: 4.96 CI:[2.63,7.96]
```

The average value of time for car is 3.96 CHF/hour (confidence interval: [1.81,6.65]). This value is abnormally low, which is a sign of a potential poor specification of the model. Note also that, with this specification, the value of time is the same for car and public transportation, as the coefficients of the time and cost variables are generic.

Finally, it is important to look at the distribution of the willingness to pay in the population/sample. We have implemented a Python function that calculates the average willingness to pay for a subgroup of the population, defined by a filter.

```
def wtpForSubgroup(filter):
    size = filter.sum()
    sim = simulatedValues[filter]
    totalWeight = sim['weight'].sum()
    weight = sim['weight'] * size / totalWeight
    wtpcar = (60 * sim['WTP CAR time'] * weight ).mean()
    wtpcar_left = (60 * left[filter]['WTP CAR time'] * weight ).mean()
    wtpcar_right = (60 * right[filter]['WTP CAR time'] * weight ).mean()
    return wtpcar, wtpcar_left,wtpcar_right
```

We start by calculating the number of entries in the filter that are True.

```
size = filter.sum()
```

Then, we extract the simulated values corresponding to the filter:

```
sim = simulatedValues[filter]
```

We calculate the total weight of these observations:

```
totalWeight = sim['weight'].sum()
```

We renormalize the weights in order to verify (1):

```
weight = sim['weight'] * size / totalWeight
```

We are now ready to calculate the average quantities:

```
wtpcar = (60 * sim['WTP CAR time'] * weight ).mean()
wtpcar_left = (60 * left[filter]['WTP CAR time'] * weight ).mean()
wtpcar_right = (60 * right[filter]['WTP CAR time'] * weight ).mean()
```

They are returned as a Python tuple:

```
return wtpcar, wtpcar_left,wtpcar_right
```

For instance, in order to obtain the value for full time workers, we use the following code:

```
filter = database.data['OccupStat'] == 1
w,l,r = wtpForSubgroup(filter)
print(f"WTP car for workers: {w:.3g} CI:[{l:.3g},{r:.3g}]")
```

This exploits the functionalities of Pandas. We have two Pandas data frames involved here: database.data is the Biogeme data file, and simulatedValues is the output of the simulation. The variable filter is a vector of boolean variables of length 1906 (the total number of observations in the sample).

We can also plot the distribution of the willingness to pay in the population (see Figure 1), using the following code:

```
import matplotlib.pyplot as plt
plt.hist(60*simulatedValues['WTP CAR time'],
         weights = simulatedValues['weight'])
plt.xlabel("WTP (CHF/hour)")
plt.ylabel("Individuals")
plt.show()
```

In this case, they are only two values: 2.42 CHF/hour and 6.69 CHF/hour. Unique values can be extracted in Pandas using the following statement:

```
60 * simulatedValues['WTP CAR time'].unique()
```

where the constant 60 is designed to report the output in CHF/hours instead of CHF/min.

5 Conclusion

PandasBiogeme is a flexible tool that allows to extract useful indicators from complex models. In this document, we have presented how some indicators relevant for discrete choice models can be generated. As the output of the simulation is a Pandas data frame, a great deal of analysis can be performed using the functionalities of Python and Pandas.

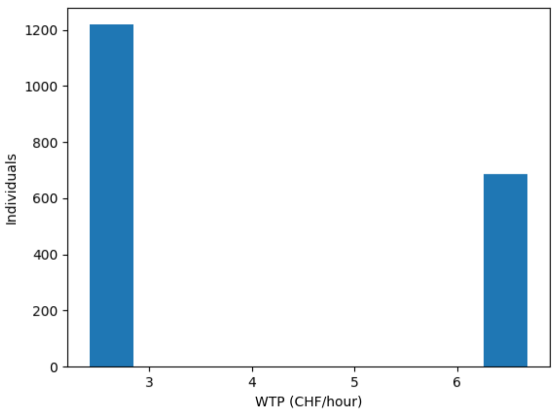

Figure 1: Distribution of the willingness to pay in the sample

A Complete specification files

We provide here the code of the specification files used in this document.

A.1 01nestedEstimation.py

```
import pandas as pd

import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models

pandas = pd.read_table("optima.dat")
database = db.Database("optima",pandas)

# The Pandas data structure is available as database.data.
# Use all the Pandas functions to investigate the database.
# For instance:
#print(database.data.describe())

from headers import *

exclude = (Choice == -1.0)
database.remove(exclude)


### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_PT = Beta('ASC_PT',0,None,None,1)
ASC_SM = Beta('ASC_SM',0,None,None,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,None,None,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,None,None,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,None,None,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,None,None,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,None,None,0)
BETA_COST = Beta('BETA_COST',0,None,None,0)


###Definition of variables:
# For numerical reasons, it is good practice to scale the data to
# that the values of the parameters are around 1.0.

# The following statements are designed to preprocess the data.
# It is like creating a new columns in the data file. This
# should be preferred to the statement like
# TimePT_scaled = Time_PT / 200.0
# which will cause the division to be reevaluated again and again,
# through the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = TimePT / 200
TimeCar_scaled = TimeCar / 200
MarginalCostPT_scaled = MarginalCostPT / 10
CostCarCHF_scaled = CostCarCHF / 10
distance_km_scaled = distance_km / 5
male = (Gender == 1)
female = (Gender == 2)
unreportedGender = (Gender == -1)

fulltime = (OccupStat == 1)
notfulltime = (OccupStat != 1)

### Definition of utility functions:
V_PT = ASC_PT + BETA_TIME_FULLTIME * TimePT_scaled * fulltime + \
       BETA_TIME_OTHER * TimePT_scaled * notfulltime + \
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR + \
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime + \
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime + \
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled * male + \
       BETA_DIST_FEMALE * distance_km_scaled * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender

# Associate utility functions with the numbering of alternatives
V = {0: V_PT,
     1: V_CAR,
```

```
    2: V_SM}

# Associate the availability conditions with the alternatives.
# In this example all alternatives are available for each individual.


av = {0: 1,
      1: 1,
      2: 1}

### DEFINITION OF THE NESTS:
# 1: nests parameter
# 2: list of alternatives

MU_NOCAR = Beta('MU_NOCAR',1.0,1.0,None,0)

CAR_NEST = 1.0 , [ 1]
NO_CAR_NEST = MU_NOCAR , [ 0, 2]
nests = CAR_NEST, NO_CAR_NEST

# The choice model is a nested logit, with availability conditions
logprob = models.lognested(V,av,nests,Choice)
biogeme  = bio.BIOGEME(database,logprob)
biogeme.modelName = "01nestedEstimation"
results = biogeme.estimate()
print("Estimated betas: {}".format(len(results.data.betaValues)))
print("Results=",results)
```

A.2 02nestedSimulation.py

```
import sys
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
import biogeme.results as res

print("Running 02nestedSimulation.py...")

pandas = pd.read_table("optima.dat")
database = db.Database("optima",pandas)

# The Pandas data structure is available as database.data. Use all the
# Pandas functions to investigate the database
#print(database.data.describe())

from headers import *

exclude = (Choice == -1.0)
database.remove(exclude)

### Normalize the weights
sumWeight = database.data['Weight'].sum()
normalizedWeight = Weight * 1906 / 0.814484

### Calculate the number of occurrences of a value in the database
numberOfMales = database.count("Gender",1)
print(f"Number of males:   {numberOfMales}")
numberOfFemales = database.count("Gender",2)
print(f"Number of females: {numberOfFemales}")
### For more complex conditions, using directly Pandas
unreportedGender = \
                     database.data[(database.data["Gender"] != 1)
                      & (database.data["Gender"] != 2)].count()["Gender"]
print(f"Unreported gender: {unreportedGender}")

### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_PT = Beta('ASC_PT',0,None,None,1)
ASC_SM = Beta('ASC_SM',0,None,None,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,None,None,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,None,None,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,None,None,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,None,None,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,None,None,0)
BETA_COST = Beta('BETA_COST',0,None,None,0)



###Definition of variables:
# For numerical reasons, it is good practice to scale the data to
```

```
# that the values of the parameters are around 1.0.

# The following statements are designed to preprocess the data.
# It is like creating a new columns in the data file. This
# should be preferred to the statement like
# TimePT_scaled = Time_PT / 200.0
# which will cause the division to be reevaluated again and again,
# through the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = TimePT / 200
TimeCar_scaled = TimeCar / 200
MarginalCostPT_scaled = MarginalCostPT / 10
CostCarCHF_scaled = CostCarCHF / 10
distance_km_scaled = distance_km / 5

male = (Gender == 1)
female = (Gender == 2)
unreportedGender = (Gender == -1)

fulltime = (OccupStat == 1)
notfulltime = (OccupStat != 1)

### Definition of utility functions:
V_PT = ASC_PT + BETA_TIME_FULLTIME * TimePT_scaled * fulltime + \
       BETA_TIME_OTHER * TimePT_scaled * notfulltime + \
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR + \
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime + \
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime + \
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled * male + \
       BETA_DIST_FEMALE * distance_km_scaled * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender

# Associate utility functions with the numbering of alternatives
V = {0: V_PT,
     1: V_CAR,
     2: V_SM}

# Associate the availability conditions with the alternatives.
# In this example all alternatives are available for each individual.


av = {0: 1,
      1: 1,
      2: 1}

### DEFINITION OF THE NESTS:
# 1: nests parameter
# 2: list of alternatives

MU_NOCAR = Beta('MU_NOCAR',1.0,1.0,None,0)

CAR_NEST = 1.0 , [ 1]
NO_CAR_NEST = MU_NOCAR , [ 0, 2]
nests = CAR_NEST, NO_CAR_NEST

# The choice model is a nested logit
prob_pt = models.nested(V,av,nests,0)
prob_car = models.nested(V,av,nests,1)
prob_sm = models.nested(V,av,nests,2)

simulate = {'weight': normalizedWeight,
            'Prob. car': prob_car,
            'Prob. public transportation': prob_pt,
            'Prob. slow modes':prob_sm,
            'Revenue public transportation':prob_pt * MarginalCostPT}

biogeme  = bio.BIOGEME(database,simulate)
biogeme.modelName = "02nestedSimulation"

""" Retrieve the names of the parameters """
betas = biogeme.freeBetaNames
""" Read the estimation results from the file """
results = res.bioResults(pickleFile='01nestedEstimation.pickle')
""" Extract the values that are necessary """
betaValues = results.getBetaValues()



"""
```

```
simulatedValues is a Panda data frame with the same number of rows as the
database, and as many columns as formulas to simulate.
"""
simulatedValues = biogeme.simulate(betaValues)

""" Calculate confidence intervals """
b = results.getBetasForSensitivityAnalysis(betas,size=100)
"""
Returns data frame containing, for each simulated value, the left and right
bounds of the confidence interval calculated by simulation.
"""
left,right = biogeme.confidenceIntervals(b,0.9)

""" We calculate now the market shares and their confidence intervals """

simulatedValues['Weighted prob. car'] = \
  simulatedValues['weight'] * simulatedValues['Prob. car']
left['Weighted prob. car'] = left['weight'] * left['Prob. car']
right['Weighted prob. car'] = right['weight'] * right['Prob. car']

marketShare_car = simulatedValues['Weighted prob. car'].mean()
marketShare_car_left = left['Weighted prob. car'].mean()
marketShare_car_right = right['Weighted prob. car'].mean()
print(f"Market share for car: {100*marketShare_car:.1f}% [{100*marketShare_car_left:.1f}%,{100*marketShare_car_ri

simulatedValues['Weighted prob. PT'] = simulatedValues['weight'] * simulatedValues['Prob. public transportation'
marketShare_pt = simulatedValues['Weighted prob. PT'].mean()
marketShare_pt_left = (left['Prob. public transportation'] * left['weight']).mean()
marketShare_pt_right = (right['Prob. public transportation'] * right['weight']).mean()
print(f"Market share for PT: {100*marketShare_pt:.1f}% [{100*marketShare_pt_left:.1f}%,{100*marketShare_pt_right

marketShare_sm = (simulatedValues['Prob. slow modes'] *
                  simulatedValues['weight']).mean()
marketShare_sm_left = (left['Prob. slow modes'] * left['weight']).mean()
marketShare_sm_right = (right['Prob. slow modes'] * right['weight']).mean()
print(f"Market share for slow modes: {100*marketShare_sm:.1f}% [{100*marketShare_sm_left:.1f}%,{100*marketShare_s

""" and, similarly, the revenues """

revenues_pt = (simulatedValues['Revenue public transportation'] *
               simulatedValues['weight']).sum()
revenues_pt_left = (left['Revenue public transportation'] *
                    left['weight']).sum()
revenues_pt_right = (right['Revenue public transportation'] *
                     right['weight']).sum()
print(f"Revenues for PT: {revenues_pt:.3f} [{revenues_pt_left:.3f},{revenues_pt_right:.3f}]")
```

A.3 03nestedElasticities.py

```
import sys
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
import biogeme.results as res

print("Running 03nestedElasticities.py...")

pandas = pd.read_table("optima.dat")
database = db.Database("optima",pandas)

# The Pandas data structure is available as database.data. Use all the
# Pandas functions to investigate the database
#print(database.data.describe())

from headers import *

exclude = (Choice == -1.0)
database.remove(exclude)

### Normalize the weights
sumWeight = database.data['Weight'].sum()
normalizedWeight = Weight * 1906 / 0.814484

### Calculate the number of occurrences of a value in the database
numberOfMales = database.count("Gender",1)
print(f"Number of males:   {numberOfMales}")
numberOfFemales = database.count("Gender",2)
print(f"Number of females: {numberOfFemales}")
### For more complex conditions, using directly Pandas
unreportedGender = \
```

```
                    database.data[(database.data["Gender"] != 1)
                     & (database.data["Gender"] != 2)].count()["Gender"]
print(f"Unreported gender: {unreportedGender}")

### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_PT = Beta('ASC_PT',0,None,None,1)
ASC_SM = Beta('ASC_SM',0,None,None,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,None,None,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,None,None,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,None,None,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,None,None,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,None,None,0)
BETA_COST = Beta('BETA_COST',0,None,None,0)



###Definition of variables:
# For numerical reasons, it is good practice to scale the data to
# that the values of the parameters are around 1.0.

# The following statements are designed to preprocess the data.
# It is like creating a new columns in the data file. This
# should be preferred to the statement like
# TimePT_scaled = Time_PT / 200.0
# which will cause the division to be reevaluated again and again,
# through the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = TimePT / 200
TimeCar_scaled = TimeCar / 200
MarginalCostPT_scaled = MarginalCostPT / 10
CostCarCHF_scaled = CostCarCHF / 10
distance_km_scaled = distance_km / 5

male = (Gender == 1)
female = (Gender == 2)
unreportedGender = (Gender == -1)

fulltime = (OccupStat == 1)
notfulltime = (OccupStat != 1)

### Definition of utility functions:
V_PT = ASC_PT + BETA_TIME_FULLTIME * TimePT_scaled * fulltime + \
       BETA_TIME_OTHER * TimePT_scaled * notfulltime + \
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR + \
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime + \
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime + \
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled * male + \
       BETA_DIST_FEMALE * distance_km_scaled * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender

# Associate utility functions with the numbering of alternatives
V = {0: V_PT,
     1: V_CAR,
     2: V_SM}

# Associate the availability conditions with the alternatives.
# In this example all alternatives are available for each individual.


av = {0: 1,
      1: 1,
      2: 1}

### DEFINITION OF THE NESTS:
# 1: nests parameter
# 2: list of alternatives

MU_NOCAR = Beta('MU_NOCAR',1.0,1.0,None,0)

CAR_NEST = 1.0 , [ 1]
NO_CAR_NEST = MU_NOCAR , [ 0, 2]
nests = CAR_NEST, NO_CAR_NEST

# The choice model is a nested logit
prob_pt = models.nested(V,av,nests,0)
prob_car = models.nested(V,av,nests,1)
prob_sm = models.nested(V,av,nests,2)

```

```python
direct_elas_pt_time = \
  Derive(prob_pt,'TimePT') * TimePT / prob_pt
direct_elas_pt_cost = \
  Derive(prob_pt,'MarginalCostPT') * MarginalCostPT / prob_pt
direct_elas_car_time = \
  Derive(prob_car,'TimeCar') * TimeCar / prob_car
direct_elas_car_cost = \
  Derive(prob_car,'CostCarCHF') * CostCarCHF / prob_car
direct_elas_sm_dist = \
  Derive(prob_sm,'distance_km') * distance_km / prob_sm

simulate = {'weight': normalizedWeight,
            'Prob. car': prob_car,
            'Prob. public transportation': prob_pt,
            'Prob. slow modes':prob_sm,
            'direct_elas_pt_time':direct_elas_pt_time,
            'direct_elas_pt_cost':direct_elas_pt_cost,
            'direct_elas_car_time':direct_elas_car_time,
            'direct_elas_car_cost':direct_elas_car_cost,
            'direct_elas_sm_dist':direct_elas_sm_dist}

biogeme  = bio.BIOGEME(database,simulate)
biogeme.modelName = "03nestedElasticties"

""" Retrieve the values of the parameters """
""" First, extract the names of parameters needed for the simulation """
betas = biogeme.freeBetaNames
""" Read the estimation results from the file """
results = res.bioResults(pickleFile='01nestedEstimation.pickle')
""" Extract the values that are necessary """
betaValues = results.getBetaValues(betas)


"""
simulatedValues is a Panda data frame with the same number of rows as the
database, and as many columns as formulas to simulate.
weighted_sinulatedValues has the same structure.
"""
simulatedValues = biogeme.simulate(betaValues)

""" We calculate the elasticities """

simulatedValues['Weighted prob. car'] = \
  simulatedValues['weight'] * simulatedValues['Prob. car']
simulatedValues['Weighted prob. PT'] = \
  simulatedValues['weight'] * simulatedValues['Prob. public transportation']
simulatedValues['Weighted prob. SM'] = \
  simulatedValues['weight'] * simulatedValues['Prob. slow modes']

denominator_car = simulatedValues['Weighted prob. car'].sum()
denominator_pt = simulatedValues['Weighted prob. PT'].sum()
denominator_sm = simulatedValues['Weighted prob. SM'].sum()

direct_elas_term_car_time = (simulatedValues['Weighted prob. car']
  * simulatedValues['direct_elas_car_time'] / denominator_car).sum()
print(f"Aggregate direct elasticity of car wrt time: {direct_elas_term_car_time:.3g}")

direct_elas_term_car_cost = (simulatedValues['Weighted prob. car']
  * simulatedValues['direct_elas_car_cost'] / denominator_car).sum()
print(f"Aggregate direct elasticity of car wrt cost: {direct_elas_term_car_cost:.3g}")

direct_elas_term_pt_time = (simulatedValues['Weighted prob. PT']
  * simulatedValues['direct_elas_pt_time'] / denominator_pt).sum()
print(f"Aggregate direct elasticity of PT wrt time: {direct_elas_term_pt_time:.3g}")

direct_elas_term_pt_cost = (simulatedValues['Weighted prob. PT']
  * simulatedValues['direct_elas_pt_cost'] / denominator_pt).sum()
print(f"Aggregate direct elasticity of PT wrt cost: {direct_elas_term_pt_cost:.3g}")


direct_elas_term_sm_dist = (simulatedValues['Weighted prob. SM']
  * simulatedValues['direct_elas_sm_dist'] / denominator_sm).sum()
print(f"Aggregate direct elasticity of SM wrt distance: {direct_elas_term_sm_dist:.3g}")
```

A.4 04 nestedElasticities .py

```python
import sys
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
```

```python
import biogeme.results as res

print("Running 04nestedElasticities.py...")

pandas = pd.read_table("optima.dat")
database = db.Database("optima",pandas)

# The Pandas data structure is available as database.data. Use all the
# Pandas functions to investigate the database
#print(database.data.describe())

from headers import *

exclude = (Choice == -1.0)
database.remove(exclude)

### Normalize the weights
sumWeight = database.data['Weight'].sum()
normalizedWeight = Weight * 1906 / 0.814484

### Calculate the number of occurrences of a value in the database
numberOfMales = database.count("Gender",1)
print(f"Number of males:   {numberOfMales}")
numberOfFemales = database.count("Gender",2)
print(f"Number of females: {numberOfFemales}")
### For more complex conditions, using directly Pandas
unreportedGender = \
                      database.data[(database.data["Gender"] != 1)
                       & (database.data["Gender"] != 2)].count()["Gender"]
print(f"Unreported gender: {unreportedGender}")


### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_PT = Beta('ASC_PT',0,None,None,1)
ASC_SM = Beta('ASC_SM',0,None,None,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,None,None,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,None,None,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,None,None,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,None,None,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,None,None,0)
BETA_COST = Beta('BETA_COST',0,None,None,0)



###Definition of variables:
# For numerical reasons, it is good practice to scale the data to
# that the values of the parameters are around 1.0.

# The following statements are designed to preprocess the data.
# It is like creating a new columns in the data file. This
# should be preferred to the statement like
# TimePT_scaled = Time_PT / 200.0
# which will cause the division to be reevaluated again and again,
# through the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = TimePT / 200
TimeCar_scaled = TimeCar / 200
MarginalCostPT_scaled = MarginalCostPT / 10
CostCarCHF_scaled = CostCarCHF / 10
distance_km_scaled = distance_km / 5

male = (Gender == 1)
female = (Gender == 2)
unreportedGender = (Gender == -1)

fulltime = (OccupStat == 1)
notfulltime = (OccupStat != 1)

### Definition of utility functions:
V_PT = ASC_PT + BETA_TIME_FULLTIME * TimePT_scaled * fulltime + \
       BETA_TIME_OTHER * TimePT_scaled * notfulltime + \
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR + \
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime + \
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime + \
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled * male + \
       BETA_DIST_FEMALE * distance_km_scaled * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender

```

```
# Associate utility functions with the numbering of alternatives
V = {0: V_PT,
     1: V_CAR,
     2: V_SM}

# Associate the availability conditions with the alternatives.
# In this example all alternatives are available for each individual.


av = {0: 1,
      1: 1,
      2: 1}

### DEFINITION OF THE NESTS:
# 1: nests parameter
# 2: list of alternatives

MU_NOCAR = Beta('MU_NOCAR',1.0,1.0,None,0)

CAR_NEST = 1.0 , [ 1]
NO_CAR_NEST = MU_NOCAR , [ 0, 2]
nests = CAR_NEST, NO_CAR_NEST



# The choice model is a nested logit
prob_pt = models.nested(V,av,nests,0)
prob_car = models.nested(V,av,nests,1)
prob_sm = models.nested(V,av,nests,2)

cross_elas_pt_time = Derive(prob_pt,'TimeCar') * TimeCar / prob_pt
cross_elas_pt_cost = Derive(prob_pt,'CostCarCHF') * CostCarCHF / prob_pt
cross_elas_car_time = Derive(prob_car,'TimePT') * TimePT / prob_car
cross_elas_car_cost = Derive(prob_car,'MarginalCostPT') * MarginalCostPT / prob_car

simulate = {'weight': normalizedWeight,
            'Prob. car': prob_car,
            'Prob. public transportation': prob_pt,
            'Prob. slow modes':prob_sm,
            'cross_elas_pt_time':cross_elas_pt_time,
            'cross_elas_pt_cost':cross_elas_pt_cost,
            'cross_elas_car_time':cross_elas_car_time,
            'cross_elas_car_cost':cross_elas_car_cost}

biogeme  = bio.BIOGEME(database,simulate)
biogeme.modelName = "02nestedSimulation_b"

""" Retrieve the values of the parameters """
""" First, extract the names of parameters needed for the simulation """
betas = biogeme.freeBetaNames
""" Read the estimation results from the file """
results = res.bioResults(pickleFile='01nestedEstimation.pickle')
""" Extract the values that are necessary """
betaValues = results.getBetaValues(betas)

"""
simulatedValues is a Panda data frame with the same number of rows as the
database, and as many columns as formulas to simulate.
weighted_sinulatedValues has the same structure.
"""
simulatedValues = biogeme.simulate(betaValues)

""" We calculate the elasticities """

simulatedValues['Weighted prob. car'] = simulatedValues['weight'] \
  * simulatedValues['Prob. car']
simulatedValues['Weighted prob. PT'] = simulatedValues['weight'] \
  * simulatedValues['Prob. public transportation']

denominator_car = simulatedValues['Weighted prob. car'].sum()
denominator_pt = simulatedValues['Weighted prob. PT'].sum()

cross_elas_term_car_time = (simulatedValues['Weighted prob. car']
  * simulatedValues['cross_elas_car_time'] / denominator_car).sum()
print(f"Aggregate cross elasticity of car wrt time: {cross_elas_term_car_time:.3g}")

cross_elas_term_car_cost = (simulatedValues['Weighted prob. car']
  * simulatedValues['cross_elas_car_cost'] / denominator_car).sum()
print(f"Aggregate cross elasticity of car wrt cost: {cross_elas_term_car_cost:.3g}")


cross_elas_term_pt_time = (simulatedValues['Weighted prob. PT']
  * simulatedValues['cross_elas_pt_time'] / denominator_pt).sum()
```

```python
print(f"Aggregate cross elasticity of PT wrt car time: {cross_elas_term_pt_time:.3g}")

cross_elas_term_pt_cost = (simulatedValues['Weighted prob. PT']
  * simulatedValues['cross_elas_pt_cost'] / denominator_pt).sum()
print(f"Aggregate cross elasticity of PT wrt car cost: {cross_elas_term_pt_cost:.3g}")
```

A.5 05 nestedElasticities .py

```python
import sys
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
import biogeme.results as res

print("Running 05nestedElasticities.py...")

pandas = pd.read_table("optima.dat")
database = db.Database("optima",pandas)

# The Pandas data structure is available as database.data. Use all the
# Pandas functions to investigate the database
#print(database.data.describe())

from headers import *

exclude = (Choice == -1.0)
database.remove(exclude)

### Normalize the weights
sumWeight = database.data['Weight'].sum()
normalizedWeight = Weight * 1906 / 0.814484

### Calculate the number of occurrences of a value in the database
numberOfMales = database.count("Gender",1)
print(f"Number of males:   {numberOfMales}")
numberOfFemales = database.count("Gender",2)
print(f"Number of females: {numberOfFemales}")
### For more complex conditions, using directly Pandas
unreportedGender = \
                    database.data[(database.data["Gender"] != 1)
                     & (database.data["Gender"] != 2)].count()["Gender"]
print(f"Unreported gender: {unreportedGender}")


### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_PT = Beta('ASC_PT',0,None,None,1)
ASC_SM = Beta('ASC_SM',0,None,None,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,None,None,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,None,None,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,None,None,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,None,None,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,None,None,0)
BETA_COST = Beta('BETA_COST',0,None,None,0)

###Definition of variables:
# For numerical reasons, it is good practice to scale the data to
# that the values of the parameters are around 1.0.

# The following statements are designed to preprocess the data.
# It is like creating a new columns in the data file. This
# should be preferred to the statement like
# TimePT_scaled = Time_PT / 200.0
# which will cause the division to be reevaluated again and again,
# through the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = TimePT / 200
TimeCar_scaled = TimeCar / 200
MarginalCostPT_scaled = MarginalCostPT / 10
CostCarCHF_scaled = CostCarCHF / 10
distance_km_scaled = distance_km / 5
delta_dist = 1.0
distance_km_scaled_after = (distance_km + delta_dist) / 5

male = (Gender == 1)
female = (Gender == 2)
unreportedGender = (Gender == -1)

fulltime = (OccupStat == 1)
```

```
notfulltime = (OccupStat != 1)

### Definition of utility functions:
V_PT = ASC_PT + BETA_TIME_FULLTIME * TimePT_scaled * fulltime + \
       BETA_TIME_OTHER * TimePT_scaled * notfulltime + \
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR + \
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime + \
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime + \
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled * male + \
       BETA_DIST_FEMALE * distance_km_scaled * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender

V_SM_after = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled_after * male + \
       BETA_DIST_FEMALE * distance_km_scaled_after * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled_after * unreportedGender

# Associate utility functions with the numbering of alternatives
V = {0: V_PT,
     1: V_CAR,
     2: V_SM}

V_after = {0: V_PT,
           1: V_CAR,
           2: V_SM_after}

# Associate the availability conditions with the alternatives.
# In this example all alternatives are available for each individual.


av = {0: 1,
      1: 1,
      2: 1}

### DEFINITION OF THE NESTS:
# 1: nests parameter
# 2: list of alternatives

MU_NOCAR = Beta('MU_NOCAR',1.0,1.0,None,0)

CAR_NEST = 1.0 , [ 1]
NO_CAR_NEST = MU_NOCAR , [ 0, 2]
nests = CAR_NEST, NO_CAR_NEST


# The choice model is a nested logit
prob_sm = models.nested(V,av,nests,2)
prob_sm_after = models.nested(V_after,av,nests,2)

direct_elas_sm_dist = \
  (prob_sm_after - prob_sm) * distance_km / (prob_sm * delta_dist)

simulate = {'weight': normalizedWeight,
            'Prob. slow modes':prob_sm,
            'direct_elas_sm_dist':direct_elas_sm_dist}

biogeme  = bio.BIOGEME(database,simulate)
biogeme.modelName = "05nestedElasticities"

""" Retrieve the values of the parameters """
""" First, extract the names of parameters needed for the simulation """
betas = biogeme.freeBetaNames
""" Read the estimation results from the file """
results = res.bioResults(pickleFile='01nestedEstimation.pickle')
""" Extract the values that are necessary """
betaValues = results.getBetaValues(betas)

"""
simulatedValues is a Panda data frame with the same number of rows as the
database, and as many columns as formulas to simulate.
weighted_sinulatedValues has the same structure.
"""
simulatedValues = biogeme.simulate(betaValues)

""" We calculate the elasticities """

simulatedValues['Weighted prob. slow modes'] = \
  simulatedValues['weight'] * simulatedValues['Prob. slow modes']

denominator_sm = simulatedValues['Weighted prob. slow modes'].sum()
```

```

direct_elas_sm_dist = (simulatedValues['Weighted prob. slow modes']
                        * simulatedValues['direct_elas_sm_dist'] /
                        denominator_sm).sum()
print(f"Aggregate direct elasticity of slow modes wrt distance: {direct_elas_sm_dist:.3g}")
```

A.6 06nestedWTP.py

```
import sys
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
import biogeme.results as res

import matplotlib.pyplot as plt

print("Running 06nestedWTP.py...")

pandas = pd.read_table("optima.dat")
database = db.Database("optima",pandas)

confidenceInterval = True

# The Pandas data structure is available as database.data. Use all the
# Pandas functions to investigate the database
#print(database.data.describe())

from headers import *

exclude = (Choice == -1.0)
database.remove(exclude)


### Normalize the weights
sumWeight = database.data['Weight'].sum()
normalizedWeight = Weight * 1906 / 0.814484

### Calculate the number of occurrences of a value in the database
numberOfMales = database.count("Gender",1)
print(f"Number of males:   {numberOfMales}")
numberOfFemales = database.count("Gender",2)
print(f"Number of females: {numberOfFemales}")
### For more complex conditions, using directly Pandas
unreportedGender = \
                   database.data[(database.data["Gender"] != 1)
                    & (database.data["Gender"] != 2)].count()["Gender"]
print(f"Unreported gender: {unreportedGender}")

### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_PT = Beta('ASC_PT',0,None,None,1)
ASC_SM = Beta('ASC_SM',0,None,None,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,None,None,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,None,None,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,None,None,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,None,None,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,None,None,0)
BETA_COST = Beta('BETA_COST',0,None,None,0)



###Definition of variables:
# For numerical reasons, it is good practice to scale the data to
# that the values of the parameters are around 1.0.

# The following statements are designed to preprocess the data.
# It is like creating a new columns in the data file. This
# should be preferred to the statement like
# TimePT_scaled = Time_PT / 200.0
# which will cause the division to be reevaluated again and again,
# through the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = TimePT / 200
TimeCar_scaled = TimeCar / 200
MarginalCostPT_scaled = MarginalCostPT / 10
CostCarCHF_scaled = CostCarCHF / 10
distance_km_scaled = distance_km / 5

male = (Gender == 1)
```

```python
female = (Gender == 2)
unreportedGender = (Gender == -1)

fulltime = (OccupStat == 1)
notfulltime = (OccupStat != 1)

### Definition of utility functions:
V_PT = ASC_PT + BETA_TIME_FULLTIME * TimePT_scaled * fulltime + \
       BETA_TIME_OTHER * TimePT_scaled * notfulltime + \
       BETA_COST * MarginalCostPT_scaled
V_CAR = ASC_CAR + \
        BETA_TIME_FULLTIME * TimeCar_scaled * fulltime + \
        BETA_TIME_OTHER * TimeCar_scaled * notfulltime + \
        BETA_COST * CostCarCHF_scaled
V_SM = ASC_SM + \
       BETA_DIST_MALE * distance_km_scaled * male + \
       BETA_DIST_FEMALE * distance_km_scaled * female + \
       BETA_DIST_UNREPORTED * distance_km_scaled * unreportedGender

# Associate utility functions with the numbering of alternatives
V = {0: V_PT,
     1: V_CAR,
     2: V_SM}

# Associate the availability conditions with the alternatives.
# In this example all alternatives are available for each individual.


av = {0: 1,
      1: 1,
      2: 1}

### DEFINITION OF THE NESTS:
# 1: nests parameter
# 2: list of alternatives

MU_NOCAR = Beta('MU_NOCAR',1.0,1.0,None,0)

CAR_NEST = 1.0 , [ 1]
NO_CAR_NEST = MU_NOCAR , [ 0, 2]
nests = CAR_NEST, NO_CAR_NEST

WTP_PT_TIME = Derive(V_PT,'TimePT') / Derive(V_PT,'MarginalCostPT')
WTP_CAR_TIME = Derive(V_CAR,'TimeCar') / Derive(V_CAR,'CostCarCHF')
#WTP_PT_TIME = WTP_PT_TIME.setBetaValues(betaValues)
#WTP_CAR_TIME = WTP_CAR_TIME.setBetaValues(betaValues)

simulate = {'weight': normalizedWeight,
            'WTP PT time': WTP_PT_TIME,
            'WTP CAR time': WTP_CAR_TIME}


biogeme = bio.BIOGEME(database,simulate)
biogeme.modelName = "06nestedWTP"

betas = biogeme.freeBetaNames
results = res.bioResults(pickleFile='01nestedEstimation.pickle')
betaValues = results.getBetaValues(betas)

"""
simulatedValues is a Panda data frame with the same number of rows as the
database, and as many columns as formulas to simulate.
"""

simulatedValues = biogeme.simulate(betaValues)

wtpcar = (60 * simulatedValues['WTP CAR time'] * simulatedValues['weight']).mean()
""" Calculate confidence intervals """
b = results.getBetasForSensitivityAnalysis(betas,size=1)
"""
    Returns data frame containing, for each simulated value, the left and right
    bounds of the confidence interval calculated by simulation.
    """
left,right = biogeme.confidenceIntervals(b,0.9)
wtpcar_left = (60 * left['WTP CAR time'] * left['weight']).mean()
wtpcar_right = (60 * right['WTP CAR time'] * right['weight']).mean()
print(f"Average WTP for car: {wtpcar:.3g} CI:[{wtpcar_left:.3g},{wtpcar_right:.3g}]")


"""
In this specific case, there are only two distinct values in the
population: for workers and non workers
"""
```

```
print("Unique values: ", [f"{i:.3g}" for i in 60 * simulatedValues['WTP CAR time'].unique()])

""" Check the value for groups of the population. Define a function
 that work for any filter to avoid repeating code """

def wtpForSubgroup(filter):
    size = filter.sum()
    sim = simulatedValues[filter]
    totalWeight = sim['weight'].sum()
    weight = sim['weight'] * size / totalWeight
    wtpcar = (60 * sim['WTP CAR time'] * weight ).mean()
    wtpcar_left = (60 * left[filter]['WTP CAR time'] * weight ).mean()
    wtpcar_right = (60 * right[filter]['WTP CAR time'] * weight ).mean()
    return wtpcar, wtpcar_left, wtpcar_right

"""
full time workers.
"""

filter = database.data['OccupStat'] == 1
w,l,r = wtpForSubgroup(filter)
print(f"WTP car for workers: {w:.3g} CI:[{l:.3g},{r:.3g}]")

"""
 females.
"""
filter = database.data['Gender'] == 2
w,l,r = wtpForSubgroup(filter)
print(f"WTP car for females: {w:.3g} CI:[{l:.3g},{r:.3g}]")

"""
 males.
"""
filter = database.data['Gender'] == 1
w,l,r = wtpForSubgroup(filter)
print(f"WTP car for males: {w:.3g} CI:[{l:.3g},{r:.3g}]")



"""
We draw the distribution of WTP in the population. In this case,
there are only two values
"""

plt.hist(60*simulatedValues['WTP CAR time'],
         weights = simulatedValues['weight'])
plt.xlabel("WTP (CHF/hour)")
plt.ylabel("Individuals")
plt.show()
```

References

Atasoy, B., Glerum, A. and Bierlaire, M. (2013). Attitudes towards mode choice in switzerland, *disP - The Planning Review* **49**(2): 101–117.

Bierlaire, M. (2017). Calculating indicators with pythonbiogeme, *Technical Report TRANSP-OR 170517*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne.

Bierlaire, M. (2018). PandasBiogeme: a short introduction, *Technical Report TRANSP-OR 181219*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne.