Calculating indicators with PythonBiogeme

Michel Bierlaire

May 17, 2017

Report TRANSP-OR 170517
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
`transp-or.epfl.ch`

Series on Biogeme

The package Biogeme (`biogeme.epfl.ch`) is designed to estimate the parameters of various models using maximum likelihood estimation. 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.

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 (2016) for an introduction to the syntax.

The parameters are estimated using PythonBiogeme. Their values are reported in Table 1. A file named 01nestedEstimation_param.py is also generated. It contains the values of the estimated parameters written in PythonBiogeme syntax, as well as the code necessary to perform a sensitivity analysis. This code provides the variance-covariance matrix of the estimates.

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	0.0590	0.217	0.27	0.79
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	0.00
7	BETA_TIME_FULLTIME	-1.60	0.333	-4.80	0.00
8	BETA_TIME_OTHER	-0.577	0.296	-1.95	0.05
9	NEST_NOCAR	1.53	0.306	1.73[1]	0.08

Summary statistics

Number of observations = 1906

Number of excluded observations = 359

Number of estimated parameters = 9

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

[1] 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. PythonBiogeme 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 what 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 engineered to 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 PythonBiogeme, a specification file must be prepared. In our example, the file 02nestedSimulation.py, reported in Section A.2, has been produced as follows:

1. Start with a copy of the model estimation file 01nestedEstimation.py.
2. Replace all Beta statements by the equivalent statements including the estimated values in the file 01nestedEstimation_param.py.
3. Copy and paste the code for the sensitivity analysis, that is
   - the names of the parameters: the line starting with names=...
   - the values of the variance-covariance matrix: the line starting with values=...
   - the definition of the matrix itself:

     ```
     vc = bioMatrix(9,names,values)
     BIOGEME_OBJECT.VARCOVAR = vc
     ```

4. Remove the statement related to the estimation:

   ```
   BIOGEME_OBJECT.ESTIMATE = Sum(logprob,'obsIter')
   ```

5. Replace it by the statement for simulation:

   ```
   BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
   ```

   The simulate variable must be a dictionary describing what has to be calculated during the sample enumeration. In this case, we calculate, for each individual in the sample, the choice probability of each alternative. We also calculate the expected revenue generated by each individual for the public transportation companies, using the following statement:

   ```
   simulate = {'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. The key of the entry is a string, that will be used for the reporting. The value must be a valid formula describing the calculation. In our example, we have defined

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

   calculating the choice probability of each alternative as provided by the nested logit model.

In the output of the estimation (see the file 01nestedEstimation.html), the sum of all weights have been calculated using the statement

```
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = Sum(Weight,'obsIter')
```

The reported result is 0.814484. Therefore, in order to verify (1), we introduce the following statements:

```
theWeight = Weight * 1906 / 0.814484
BIOGEME_OBJECT.WEIGHT = theWeight
```

as there are 1906 entries in the data file.

The following statements are included for the calculation of elasticities and will be used later (see Section 3 for more details):

```
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities PT'] =
  Sum(theWeight * prob_pt ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities CAR'] =
  Sum(theWeight * prob_car ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities SM'] =
  Sum(theWeight * prob_sm ,'obsIter')
```

The simulation is performed using the statement

```
pythonbiogeme 02nestedSimulation optima.dat
```

It generates the file 02nestedSimulation.html, that contains the following sections:

- The preamble reports information about the version of PythonBiogeme, useful URLs and the names of the files involved in the run.

- Statistics: this section is the same as for the estimation, and reports the requested statistics:

```
                               Alt. 0 available:  1906
                                  Alt. 0 chosen:  536
                               Alt. 1 available:  1906
                                  Alt. 1 chosen:  1256
                               Alt. 2 available:  1906
                                  Alt. 2 chosen:  114
Cte loglikelihood (only for full choice sets):  -1524.92
                               Gender: females:  871
                                 Gender: males:  943
                             Gender: unreported:  92
          Normalization for elasticities CAR:  1244.77
           Normalization for elasticities PT:  535.086
           Normalization for elasticities SM:  126.147
                             Null loglikelihood:  -2093.96
                              Number of entries:  1906
                          Occupation: full time:  798
                                 Sum of weights:  0.814484
```

- The simulation report contains two parts: the aggregate values, and the detailed records. We start by describing the latter. It reports, for each row of the sample file, the weight w_n (last column) and, for each entry in the dictionary defined by BIOGEME_OBJECT.SIMULATE

  1. the calculated quantity,
  2. the 90% confidence interval for this quantity. It is calculated using simulation. As the estimates have been obtained from maximum likelihood, they are (asymptotically) normally distributed. Therefore, we draw from a multivariate normal distribution $N(\widehat{\beta}, \widehat{\Sigma})$, where $\widehat{\beta}$ is the vector of estimated parameters, and $\widehat{\Sigma}$ is the variance-covariance matrix defined by the BIOGEME OBJECT.VARCOVAR statement. The number of draws is controlled by the parameter NbrOfDrawsForSensitivityAnalysis. The requested quantity is calculated for each realization, and the 5% and the 95% quantiles of the obtained simulated values are reported to generate the 90% confidence interval. Note that the confidence interval is reported only if the statement

     ```
     BIOGEME_OBJECT.VARCOVAR = vc
     ```

     is present. If you do not need the confidence intervals, simply remove this statement from the .py file.

- Simulation report: aggregate values. For each calculated quantity, aggregate indicators are calculated. Denote by z_n the calculated quantity (in this case, the probability that individual n chooses the car alternative, for instance). Then, the following aggregate values are reported, together with the associated confidence interval (if requested):

  – Total:
  $$\sum_{n=1}^{N_s} z_n. \tag{4}$$
  – Weighted total:
  $$\sum_{n=1}^{N_s} w_n z_n. \tag{5}$$
  – Average:
  $$\frac{1}{N_s}\sum_{n=1}^{N_s} z_n. \tag{6}$$

– Weighted average:

$$\frac{1}{N_s}\sum_{n=1}^{N_s} w_n z_n. \tag{7}$$

– Non zeros:

$$\sum_{n=1}^{N_s} \delta(z_n \neq 0), \tag{8}$$

where

$$\delta(z_n \neq 0) = \begin{cases} 1 & \text{if } z_n \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

– Non zeros average:

$$\frac{\sum_{n=1}^{N_s} z_n}{\sum_{n=1}^{N_s} \delta(z_n \neq 0)}. \tag{10}$$

– Weighted non zeros average:

$$\frac{\sum_{n=1}^{N_s} w_n z_n}{\sum_{n=1}^{N_s} \delta(z_n \neq 0)}. \tag{11}$$

– Minimum:

$$\min_n z_n. \tag{12}$$

– Maximum:

$$\max_n z_n. \tag{13}$$

Therefore, the result of (2) is available in the row "Weighted average". In this example, the market shares are:

- car: 65.3078% (confidence interval: [60.5884%,69.0407%]),
- public transportation: 28.0738% (confidence interval: [23.603%,32.391%],
- slow modes: 6.61844% (confidence interval: 4.54637%,10.417%).

The result of (3) is obtained in the row "Weighted total". In this case, the expected revenue (generated by the individuals in the sample) is 3018.29 (confidence interval: [2442.87,3826.36]).

3 Elasticities

Consider now one of the variables involved in the model, for instance x_{ink}, the kth variable associated by individual n to 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

$$\frac{\partial x_{ink}}{x_{ink}} = \frac{\partial x_{ipk}}{x_{ipk}} = \frac{\partial x_{ik}}{x_{ik}}, \tag{14}$$

where

$$x_{ik} = \frac{1}{N_s} \sum_{n=1}^{N_s} x_{ink}. \tag{15}$$

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

$$E^{P_n(i)}_{x_{ink}} = \frac{\partial P_n(i|x_n, \mathcal{C}_n)}{\partial x_{ink}} \frac{x_{ink}}{P_n(i|x_n, \mathcal{C}_n)}. \tag{16}$$

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

$$E^{W_i}_{x_{ik}} = \frac{\partial W_i}{\partial x_{ik}} \frac{x_{ik}}{W_i}. \tag{17}$$

Using (2), we obtain

$$E^{W_i}_{x_{ik}} = \frac{1}{N_s} \sum_{n=1}^{N_s} w_n \frac{\partial P_n(i|x_n, \mathcal{C}_n)}{\partial x_{ik}} \frac{x_{ik}}{W_i}. \tag{18}$$

From (14), we obtain

$$E_{x_{ik}}^{W_i} = \frac{1}{N_s}\sum_{n=1}^{N_s} w_n \frac{\partial P_n(i|x_n,\mathcal{C}_n)}{\partial x_{ink}}\frac{x_{ink}}{W_i} = \frac{1}{N_s}\sum_{n=1}^{N_s} w_n E_{x_{ink}}^{P_n(i)}\frac{P_n(i|x_n,\mathcal{C}_n)}{W_i}, \tag{19}$$

where the second equation is derived from (16). Using (2) again, we obtain

$$E_{x_{ik}}^{W_i} = \sum_{n=1}^{N_s} E_{x_{ink}}^{P_n(i)} \frac{w_n P_n(i|x_n,\mathcal{C}_n)}{\sum_{n=1}^{N_s} w_n P_n(i|x_n,\mathcal{C}_n)}. \tag{20}$$

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

$$E_{x_{jnk}}^{P_n(i)} = \frac{\partial P_n(i|x_n,\mathcal{C}_n)}{\partial x_{jnk}} \frac{x_{jnk}}{P_n(i|x_n,\mathcal{C}_n)}. \tag{21}$$

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

$$\frac{\Delta x_{ink}}{x_{ink}} = \frac{\Delta x_{ipk}}{x_{ipk}} = \frac{\Delta x_{ik}}{x_{ik}}, \tag{22}$$

where x_{ik} is defined by (15). The *disaggregate direct arc elasticity* of the model with respect to the variable x_{ink} is defined as

$$E_{x_{ink}}^{P_n(i)} = \frac{\Delta P_n(i|x_n,\mathcal{C}_n)}{\Delta x_{ink}} \frac{x_{ink}}{P_n(i|x_n,\mathcal{C}_n)}. \tag{23}$$

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

$$E_{x_{ik}}^{W_i} = \frac{\Delta W_i}{\Delta x_{ik}} \frac{x_{ik}}{W_i}. \tag{24}$$

The two quantities are also related by (20), following the exact same derivation as for the point elasticity.

3.3 Using PythonBiogeme for point elasticities

See 03nestedElasticities.py in Section A.3

The calculation of (16) 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, PythonBiogeme provides an operator that calculates the derivative of a formula. It is illustrated in the file 03nestedElasticities.py, reported in Section A.3. The statements that trigger the calculation of the elasticities are:

```
elas_pt_time = Derive(prob_pt,'TimePT') * TimePT / prob_pt
elas_pt_cost = Derive(prob_pt,'MarginalCostPT') * MarginalCostPT / prob_pt
elas_car_time = Derive(prob_car,'TimeCar') * TimeCar / prob_car
elas_car_cost = Derive(prob_car,'CostCarCHF') * CostCarCHF / prob_car
elas_sm_dist = Derive(prob_sm,'distance_km') * distance_km / prob_sm
```

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 have been scaled in order to improve the numerical properties of the likelihood function, using statements like

```
TimePT_scaled = DefineVariable('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, the relationship between the new variable and the original one is lost. Therefore, PythonBiogeme is not able to properly calculate the derivatives. In this example, the variable of interest is TimePT, not TimePT_scaled. And their relationship must be explicitly known to correctly calculate the derivatives. Consequently, all statements such as

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

should be replaced by statements such as

```
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 (20). This requires the calculation of the normalization factors

$$\sum_{n=1}^{N_s} w_n P_n(i|x_n, \mathcal{C}_n). \tag{25}$$

This has been performed during the previous simulation using the statements

```
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities PT'] = \
                    Sum(theWeight * prob_pt ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities CAR'] = \
                    Sum(theWeight * prob_car ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities SM'] = \
                    Sum(theWeight * prob_sm ,'obsIter')
```

Therefore, we have now included the following statements:

```
normalization_pt  = 535.086
normalization_car = 1244.77
normalization_sm = 126.147
```

The quantities that must be calculated for each individual in order to derive the aggregate elasticities, correspond to the following entries in the dictionary:

```
'Agg. Elast. PT - Time': elas_pt_time * prob_pt / normalization_pt,
'Agg. Elast. PT - Cost': elas_pt_cost * prob_pt / normalization_pt,
'Agg. Elast. Car - Time': elas_car_time * prob_car / normalization_car,
'Agg. Elast. Car - Cost': elas_car_cost * prob_car / normalization_car,
'Agg. Elast. Slow modes - Distance': elas_sm_dist * prob_sm / normalization_sm
```

Note that the weights have not been included in the above formula, so that the values of the aggregate elasticities can be found in the row "Weighted total":

- Car — cost: -0.0906321,
- Car — travel time: -0.0440771,
- Public transportation — cost: -0.320246,
- Public transportation — travel time: -0.274315,
- Slow modes — distance: -1.09095.

Equivalently, we could have used statements like

```
'Agg. Elast. PT - Time': theWeight * elas_pt_time * prob_pt / normalization_pt,
```

and the aggregate value would have been found in the row "Total" instead of "Weighted total'. Note also that we have omitted to report the confidence intervals in this example, by commenting out the statement:

```
#BIOGEME_OBJECT.VARCOVAR = vc
```

The results are found in the file 03nestedElasticities.html.

3.4 Using PythonBiogeme for cross elasticities

See 04nestedElasticities.py in Section A.4

The calculation of (21) is performed in a similar way as the direct elasticities (16), using the following statements:

```
elas_car_cost = Derive(prob_car,'MarginalCostPT') * MarginalCostPT / prob_car
elas_car_time = Derive(prob_car,'TimePT') * TimePT / prob_car
elas_pt_cost = Derive(prob_pt,'CostCarCHF') * CostCarCHF / prob_pt
elas_pt_time = Derive(prob_pt,'TimeCar') * TimeCar / prob_pt
```

They calculate the following elasticities:

- choice of car with respect to the marginal cost of public transportation,
- choice of car with respect to travel time by public transportation,
- choice of public transportation with respect to cost of the car,
- choice of public transportation with respect to travel time by car.

The corresponding aggregate elasticities are calculated exactly like for the direct case, and their values can be found in the row "Weighted total".

- Agg. Elast. Car - Cost PT: 0.123008
- Agg. Elast. Car - Time PT: 0.106567
- Agg. Elast. PT - Cost car: 0.199984
- Agg. Elast. PT - Time car: 0.0953097

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

The results are found in the file 04nestedElasticities.html.

3.5 Using PythonBiogeme 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 aggregate elasticity is calculated as explained above. It is equal here to -1.00708, and the confidence interval is [-1.7212,-0.562574].

The results are found in the file 05nestedElasticities.html.

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

$$V_{in}(c_{in} + \delta^c_{in}, x_{ink} + \delta^x_{ink}) = V_{in}(c_{in}, x_{ink}). \tag{26}$$

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

$$\delta^c_{in}/\delta^x_{ink}, \tag{27}$$

and is obtained by solving Equation (26). If x_{ink} and c_{in} appear linearly in the utility function, that is if

$$V_{in}(c_{in}, x_{ink}) = \beta_c c_{in} + \beta_x x_{ink} + \cdots , \tag{28}$$

and

$$V_{in}(c_{in} + \delta^c_{in}, x_{ink} + \delta^x_{ink}) = \beta_c(c_{in} + \delta^c_{in}) + \beta_x(x_{ink} + \delta^x_{ink}) + \cdots . \quad (29)$$

Therefore, (27) is

$$\delta^c_{in}/\delta^x_{ink} = -\beta_x/\beta_c. \quad (30)$$

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 (26):

$$\begin{aligned} V_{in}(c_{in}, x_{ink}) &= V_{in}(c_{in} + \delta^c_{in}, x_{ink} + \delta^x_{ink}) \\ &\approx V_{in}(c_{in}, x_{ink}) + \delta^c_{in}\frac{\partial V_{in}}{\partial c_{in}}(c_{in}, x_{ink}) + \delta^x_{ink}\frac{\partial V_{in}}{\partial x_{ink}}(c_{in}, x_{ink}) \end{aligned} \quad (31)$$

Therefore, the willingness to pay is equal to

$$\frac{\delta^c_{in}}{\delta^x_{ink}} = -\frac{(\partial V_{in}/\partial x_{ink})(c_{in}, x_{ink})}{(\partial V_{in}/\partial c_{in})(c_{in}, x_{ink})}. \quad (32)$$

Note that if x_{ink} and c_{in} appear linearly in the utility function, (32) is the same as (30). 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}:

$$\frac{\delta^c_{in}}{\delta^x_{ink}} = \frac{(\partial V_{in}/\partial x_{ink})(c_{in}, x_{ink})}{(\partial V_{in}/\partial c_{in})(c_{in}, x_{ink})}. \quad (33)$$

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

$$\frac{\delta^c_{in}}{\delta^t_{in}} = \frac{(\partial V_{in}/\partial t_{in})(c_{in}, t_{in})}{(\partial V_{in}/\partial c_{in})(c_{in}, t_{in})} = \frac{\beta_t}{\beta_c}, \quad (34)$$

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 (33) involves the calculation of derivatives. It is done in Pythonbiogeme 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 aggregate values are found in the "Weighted average" row of the report file: 3.95822 CHF/hour (confidence interval: [1.98696,7.81565]). Note that 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. The detailed records of the report file allows to do so. It is easy to drag and drop the HTML report file into your favorite spreadsheet software in order to perform additional statistics. In this example, the value of time takes two values, depending on the employment status of the individual:

- Full time: 6.68992 (confidence interval: [4.15056, 11.1866])
- Not full time: 2.41847 (confidence interval: [0.829511, 5.91561])

The results are found in the file 06nestedWTP.html.

5 Conclusion

PythonBiogeme 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. The HTML format of the report allows to display the report in your favorite browser. It also allows to import the generated values in a spreadsheet for more manipulations.

A Complete specification files

A.1 01nestedEstimation.py

Available at biogeme.epfl.ch/examples/indicators/python/01nestedEstimation.py

```
## File 01nestedEstimation.py
## Simple nested logit model for the Optima case study
## Wed May 10 10:55:12 2017

from biogeme import *
from headers import *
from loglikelihood import *
from statistics import *
from nested import *

### Three alternatives:
# CAR: automobile
# PT: public transportation
# SM: slow mode (walking, biking)

### List of parameters to be estimated
ASC_CAR = Beta('ASC_CAR',0,-10000,10000,0)
ASC_SM = Beta('ASC_SM',0,-10000,10000,0)
BETA_TIME_FULLTIME = Beta('BETA_TIME_FULLTIME',0,-10000,10000,0)
BETA_TIME_OTHER = Beta('BETA_TIME_OTHER',0,-10000,10000,0)
BETA_DIST_MALE = Beta('BETA_DIST_MALE',0,-10000,10000,0)
BETA_DIST_FEMALE = Beta('BETA_DIST_FEMALE',0,-10000,10000,0)
BETA_DIST_UNREPORTED = Beta('BETA_DIST_UNREPORTED',0,-10000,10000,0)
BETA_COST = Beta('BETA_COST',0,-10000,10000,0)


###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,
# throuh the iterations. For models taking a long time to
# estimate, it may make a significant difference.

TimePT_scaled = DefineVariable('TimePT_scaled', TimePT / 200 )
TimeCar_scaled = DefineVariable('TimeCar_scaled', TimeCar / 200 )
MarginalCostPT_scaled = DefineVariable('MarginalCostPT_scaled',
                                       MarginalCostPT / 10 )
CostCarCHF_scaled = DefineVariable('CostCarCHF_scaled',
```

```
                                        CostCarCHF / 10 )
distance_km_scaled = DefineVariable('distance_km_scaled',
                                         distance_km / 5 )

male = DefineVariable('male',Gender == 1)
female = DefineVariable('female',Gender == 2)
unreportedGender = DefineVariable('unreportedGender',Gender == -1)

fulltime = DefineVariable('fulltime',OccupStat == 1)
notfulltime = DefineVariable('notfulltime',OccupStat != 1)

### Definition of utility functions:
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

# 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

NEST_NOCAR = Beta('NEST_NOCAR',1,1.0,10,0)

CAR = 1.0 , [ 1]
NO_CAR = NEST_NOCAR , [ 0, 2]
nests = CAR, NO_CAR

# All observations verifying the following expression will not be
# considered for estimation
BIOGEME_OBJECT.EXCLUDE = Choice == -1

```

```

# The choice model is a nested logit, with availability conditions
logprob = lognested(V,av,nests,Choice)

# Defines an itertor on the data
rowIterator('obsIter')

#Statistics
nullLoglikelihood(av,'obsIter')
choiceSet = [0,1,2]
cteLoglikelihood(choiceSet,Choice,'obsIter')
availabilityStatistics(av,'obsIter')

BIOGEME_OBJECT.STATISTICS['Gender: males'] = \
                    Sum(male,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: females'] = \
                    Sum(female,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: unreported'] = \
                    Sum(unreportedGender,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = \
                    Sum(fulltime,'obsIter')
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = \
                    Sum(Weight,'obsIter')

# Define the likelihood function for the estimation
BIOGEME_OBJECT.ESTIMATE = Sum(logprob,'obsIter')
BIOGEME_OBJECT.PARAMETERS['optimizationAlgorithm'] = "CFSQP"
```

A.2 02nestedSimulation.py

Available at biogeme.epfl.ch/examples/indicators/python/02nestedSimulation.py

```
## File 02nestedSimulation.py
## Simple nested logit model for the Optima case study
## Wed May 10 11:24:32 2017

from biogeme import *
from headers import *
from statistics import *
from nested import *

### Three alternatives:
# CAR: automobile
# PT: public transportation
# SM: slow mode (walking, biking)

### List of parameters and their estimated value.
ASC_CAR = Beta('ASC_CAR',0.261291,-10000,10000,0,'ASC_CAR' )
```

```
ASC_SM = Beta('ASC_SM',0.0590204,-10000,10000,0,'ASC_SM' )
BETA_TIME_FULLTIME = \
 Beta('BETA_TIME_FULLTIME',-1.59709,-10000,10000,0,'BETA_TIME_FULLTIME' )
BETA_TIME_OTHER = \
 Beta('BETA_TIME_OTHER',-0.577362,-10000,10000,0,'BETA_TIME_OTHER' )
BETA_DIST_MALE = \
 Beta('BETA_DIST_MALE',-0.686327,-10000,10000,0,'BETA_DIST_MALE' )
BETA_DIST_FEMALE = \
 Beta('BETA_DIST_FEMALE',-0.83121,-10000,10000,0,'BETA_DIST_FEMALE' )
BETA_DIST_UNREPORTED = \
 Beta('BETA_DIST_UNREPORTED',-0.702974,-10000,10000,0,'BETA_DIST_UNREPORTED' )
BETA_COST = \
 Beta('BETA_COST',-0.716192,-10000,10000,0,'BETA_COST' )


###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,
# throuh the iterations. For models taking a long time to estimate, it
# may make a significant difference.

TimePT_scaled = DefineVariable('TimePT_scaled', TimePT / 200 )
TimeCar_scaled = DefineVariable('TimeCar_scaled', TimeCar / 200 )
MarginalCostPT_scaled = DefineVariable('MarginalCostPT_scaled',
                                   MarginalCostPT / 10 )
CostCarCHF_scaled = DefineVariable('CostCarCHF_scaled',
                                CostCarCHF / 10 )
distance_km_scaled = DefineVariable('distance_km_scaled',
                                 distance_km / 5 )

male = DefineVariable('male',Gender == 1)
female = DefineVariable('female',Gender == 2)
unreportedGender  = DefineVariable('unreportedGender',Gender == -1)

fulltime = DefineVariable('fulltime',OccupStat == 1)
notfulltime = DefineVariable('notfulltime',OccupStat != 1)

### Definition of utility functions:
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


# 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

NEST_NOCAR = Beta('NEST_NOCAR',1.52853,1,10,0,'NEST_NOCAR' )


CAR = 1.0  , [  1]
NO_CAR = NEST_NOCAR  , [ 0,   2]
nests = CAR, NO_CAR

# All observations verifying the following expression will not be
# considered for estimation
exclude = (Choice   ==  -1)
BIOGEME_OBJECT.EXCLUDE =  exclude

##
## This has been copied-pasted from the file 01nestedEstimation_param.py
##
## Code for the sensitivity analysis generated after the estimation of the model
names = ['ASC_CAR','ASC_SM','BETA_COST','BETA_DIST_FEMALE','BETA_DIST_MALE','BETA_
values = [[0.0100225,-0.0023271,0.00151986,0.00285251,0.00621963,0.00247439,0.02359
vc = bioMatrix(9,names,values)
BIOGEME_OBJECT.VARCOVAR = vc



# The choice model is a nested logit
prob_pt = nested(V,av,nests,0)
prob_car = nested(V,av,nests,1)
```

```
prob_sm = nested(V,av,nests,2)

# Defines an itertor on the data
rowIterator('obsIter')

#Statistics
nullLoglikelihood(av,'obsIter')
choiceSet = [0,1,2]
cteLoglikelihood(choiceSet,Choice,'obsIter')
availabilityStatistics(av,'obsIter')

# Each weight is normalized so that the sum of weights is equal to the
# number of entries (1906).
# The normalization factor has been calculated during estimation
theWeight = Weight * 1906 / 0.814484


BIOGEME_OBJECT.STATISTICS['Gender: males'] = \
                        Sum(male,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: females'] = \
                        Sum(female,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: unreported'] = \
                        Sum(unreportedGender,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = \
                        Sum(fulltime,'obsIter')
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = \
                        Sum(Weight,'obsIter')
BIOGEME_OBJECT.STATISTICS['Number of entries'] = \
                        Sum(1-exclude,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities PT'] = \
                        Sum(theWeight * prob_pt ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities CAR'] = \
                        Sum(theWeight * prob_car ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities SM'] = \
                        Sum(theWeight * prob_sm ,'obsIter')

# Define the dictionary for the simulation.
simulate = {'Prob. car': prob_car,
            'Prob. public transportation': prob_pt,
            'Prob. slow modes':prob_sm,
            'Revenue public transportation':
                    prob_pt * MarginalCostPT}

BIOGEME_OBJECT.WEIGHT = theWeight
BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
```

A.3 03 nestedElasticities .py

Available at biogeme.epfl.ch/examples/indicators/python/03nestedElasticities.py

```
## File 03nestedElasticities.py
## Simple nested logit model for the Optima case study
## Calculation of direct point elasticities
## Wed May 10 12:20:59 2017

from biogeme import *
from headers import *
from statistics import *
from nested import *

### Three alternatives:
# CAR: automobile
# PT: public transportation
# SM: slow mode (walking, biking)

### List of parameters and their estimated value.
ASC_CAR = Beta('ASC_CAR',0.261291,-10000,10000,0,'ASC_CAR' )
ASC_SM = Beta('ASC_SM',0.0590204,-10000,10000,0,'ASC_SM' )
BETA_TIME_FULLTIME = \
 Beta('BETA_TIME_FULLTIME',-1.59709,-10000,10000,0,'BETA_TIME_FULLTIME' )
BETA_TIME_OTHER = \
 Beta('BETA_TIME_OTHER',-0.577362,-10000,10000,0,'BETA_TIME_OTHER' )
BETA_DIST_MALE = \
 Beta('BETA_DIST_MALE',-0.686327,-10000,10000,0,'BETA_DIST_MALE' )
BETA_DIST_FEMALE = \
 Beta('BETA_DIST_FEMALE',-0.83121,-10000,10000,0,'BETA_DIST_FEMALE' )
BETA_DIST_UNREPORTED = \
 Beta('BETA_DIST_UNREPORTED',-0.702974,-10000,10000,0,'BETA_DIST_UNREPORTED' )
BETA_COST = \
 Beta('BETA_COST',-0.716192,-10000,10000,0,'BETA_COST' )

###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.

### Warning: when calculation derivatives, the total formula must be
### known to Biogeme. In this case, the use of
### "DefineVariable" must be omitted, if the derivatives must be
### calculated with respect to the original variables (as is often the
### case)

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

#TimeCar_scaled  = DefineVariable('TimeCar_scaled', TimeCar   /
200 )
```

```
TimeCar_scaled  =  TimeCar   /  200

#MarginalCostPT_scaled  = DefineVariable('MarginalCostPT_scaled', MarginalCostPT
/  10 )
MarginalCostPT_scaled  =  MarginalCostPT   /  10

#CostCarCHF_scaled  = DefineVariable('CostCarCHF_scaled', CostCarCHF
/  10 )
CostCarCHF_scaled  = CostCarCHF   /  10

#distance_km_scaled  = DefineVariable('distance_km_scaled', distance_km
/  5 )
distance_km_scaled  = distance_km   /  5

male = DefineVariable('male',Gender == 1)
female = DefineVariable('female',Gender == 2)
unreportedGender  = DefineVariable('unreportedGender',Gender == -1)

fulltime = DefineVariable('fulltime',OccupStat == 1)
notfulltime = DefineVariable('notfulltime',OccupStat != 1)

### Definition of utility functions:

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

# 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
```

```
NEST_NOCAR = Beta('NEST_NOCAR',1.52853,1,10,0,'NEST_NOCAR' )


CAR = 1.0 , [ 1]
NO_CAR = NEST_NOCAR , [ 0, 2]
nests = CAR, NO_CAR

# All observations verifying the following expression will not be
# considered for estimation
exclude = (Choice == -1)
BIOGEME_OBJECT.EXCLUDE = exclude


##
## This has been copied-pasted from the file 01nestedEstimation_param.py
##
## Code for the sensitivity analysis generated after the estimation of the model
names = ['ASC_CAR','ASC_SM','BETA_COST','BETA_DIST_FEMALE','BETA_DIST_MALE','BETA_
values = [[0.0100225,-0.0023271,0.00151986,0.00285251,0.00621963,0.00247439,0.02359
vc = bioMatrix(9,names,values)
#BIOGEME_OBJECT.VARCOVAR = vc



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

elas_pt_time = Derive(prob_pt,'TimePT') * TimePT / prob_pt
elas_pt_cost = Derive(prob_pt,'MarginalCostPT') * MarginalCostPT / prob_pt
elas_car_time = Derive(prob_car,'TimeCar') * TimeCar / prob_car
elas_car_cost = Derive(prob_car,'CostCarCHF') * CostCarCHF / prob_car
elas_sm_dist = Derive(prob_sm,'distance_km') * distance_km / prob_sm

# Defines an itertor on the data
rowIterator('obsIter')
#Statistics
nullLoglikelihood(av,'obsIter')
choiceSet = [0,1,2]
cteLoglikelihood(choiceSet,Choice,'obsIter')
availabilityStatistics(av,'obsIter')

# Each weight is normalized so that the sum of weights is equal to the
# numer of entries (1906)
# The normalization factor has been calculated during estimation

theWeight = Weight * 1906 / 0.814484
```

```
normalization_pt  = 535.086
normalization_car = 1244.77
normalization_sm = 126.147

BIOGEME_OBJECT.STATISTICS['Gender: males'] = \
                    Sum(male,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: females'] = \
                    Sum(female,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: unreported'] = \
                    Sum(unreportedGender,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = \
                    Sum(fulltime,'obsIter')
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = \
                    Sum(Weight,'obsIter')
BIOGEME_OBJECT.STATISTICS['Number of entries'] = \
                    Sum(1-exclude,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities PT'] = \
                    Sum(theWeight * prob_pt ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities CAR'] = \
                    Sum(theWeight * prob_car ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities SM'] = \
                    Sum(theWeight * prob_sm ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = Sum(fulltime,'obsIter')

# Define the dictionary for the simulation.
simulate = {'Disag. Elast. PT - Time': elas_pt_time,
            'Disag. Elast. PT - Cost': elas_pt_cost,
            'Disag. Elast. Car - Time': elas_car_time,
            'Disag. Elast. Car - Cost': elas_car_cost,
            'Disag. Elast. Slow modes - Distance': elas_sm_dist,
            'Agg. Elast. PT - Time': \
               elas_pt_time * prob_pt / normalization_pt,
            'Agg. Elast. PT - Cost': \
               elas_pt_cost * prob_pt / normalization_pt,
            'Agg. Elast. Car - Time': \
               elas_car_time * prob_car / normalization_car,
            'Agg. Elast. Car - Cost': \
               elas_car_cost * prob_car / normalization_car,
            'Agg. Elast. Slow modes - Distance': \
               elas_sm_dist * prob_sm / normalization_sm
}

BIOGEME_OBJECT.WEIGHT = theWeight
BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
```

A.4 04nestedElasticities.py

Available at biogeme.epfl.ch/examples/indicators/python/04nestedElasticities.py

```python
## File 04nestedElasticities.py
## Simple nested logit model for the Optima case study
## Calculation of cross point elasticities
## Thu May 11 16:38:05 2017

from biogeme import *
from headers import *
from statistics import *
from nested import *

### Three alternatives:
# CAR: automobile
# PT: public transportation
# SM: slow mode (walking, biking)

### List of parameters and their estimated value.
ASC_CAR = Beta('ASC_CAR',0.261291,-10000,10000,0,'ASC_CAR' )
ASC_SM = Beta('ASC_SM',0.0590204,-10000,10000,0,'ASC_SM' )
BETA_TIME_FULLTIME = \
 Beta('BETA_TIME_FULLTIME',-1.59709,-10000,10000,0,'BETA_TIME_FULLTIME' )
BETA_TIME_OTHER = \
 Beta('BETA_TIME_OTHER',-0.577362,-10000,10000,0,'BETA_TIME_OTHER' )
BETA_DIST_MALE = \
 Beta('BETA_DIST_MALE',-0.686327,-10000,10000,0,'BETA_DIST_MALE' )
BETA_DIST_FEMALE = \
 Beta('BETA_DIST_FEMALE',-0.83121,-10000,10000,0,'BETA_DIST_FEMALE' )
BETA_DIST_UNREPORTED = \
 Beta('BETA_DIST_UNREPORTED',-0.702974,-10000,10000,0,'BETA_DIST_UNREPORTED' )
BETA_COST = \
 Beta('BETA_COST',-0.716192,-10000,10000,0,'BETA_COST' )


###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.

### Warning: when calculation derivatives, the total formula must be
### known to Biogeme. In this case, the use of
### "DefineVariable" must be omitted, if the derivatives must be
### calculated with respect to the original variables (as is often the
### case)

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

#TimeCar_scaled  = DefineVariable('TimeCar_scaled', TimeCar   /  200 )
TimeCar_scaled  =  TimeCar   /  200

```

```
#MarginalCostPT_scaled  = DefineVariable('MarginalCostPT_scaled', MarginalCostPT / 10 )
MarginalCostPT_scaled  =  MarginalCostPT  /  10

#CostCarCHF_scaled  = DefineVariable('CostCarCHF_scaled', CostCarCHF / 10 )
CostCarCHF_scaled  = CostCarCHF  /  10

#distance_km_scaled  = DefineVariable('distance_km_scaled', distance_km / 5 )
distance_km_scaled  = distance_km  /  5

male = DefineVariable('male',Gender == 1)
female = DefineVariable('female',Gender == 2)
unreportedGender  = DefineVariable('unreportedGender',Gender == -1)

fulltime = DefineVariable('fulltime',OccupStat == 1)
notfulltime = DefineVariable('notfulltime',OccupStat != 1)

### Definition of utility functions:

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

# 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

NEST_NOCAR = Beta('NEST_NOCAR',1.52853,1,10,0,'NEST_NOCAR' )
```

```
CAR = 1.0 , [ 1]
NO_CAR = NEST_NOCAR , [ 0, 2]
nests = CAR, NO_CAR

# All observations verifying the following expression will not be
# considered for estimation
exclude = (Choice == -1)
BIOGEME_OBJECT.EXCLUDE = exclude


##
## This has been copied-pasted from the file 01nestedEstimation_param.py
##
## Code for the sensitivity analysis generated after the estimation of the model
names = ['ASC_CAR','ASC_SM','BETA_COST','BETA_DIST_FEMALE','BETA_DIST_MALE','BETA_
values = [[0.0100225,-0.0023271,0.00151986,0.00285251,0.00621963,0.00247439,0.02359
vc = bioMatrix(9,names,values)
#BIOGEME_OBJECT.VARCOVAR = vc



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

elas_car_cost = Derive(prob_car,'MarginalCostPT') * MarginalCostPT / prob_car
elas_car_time = Derive(prob_car,'TimePT') * TimePT / prob_car
elas_pt_cost = Derive(prob_pt,'CostCarCHF') * CostCarCHF / prob_pt
elas_pt_time = Derive(prob_pt,'TimeCar') * TimeCar / prob_pt

# Defines an itertor on the data
rowIterator('obsIter')
#Statistics
nullLoglikelihood(av,'obsIter')
choiceSet = [0,1,2]
cteLoglikelihood(choiceSet,Choice,'obsIter')
availabilityStatistics(av,'obsIter')

theWeight = Weight * 1906 / 0.814484
normalization_pt = 535.086
normalization_car = 1244.77
normalization_sm = 126.147

BIOGEME_OBJECT.STATISTICS['Gender: males'] = \
                         Sum(male,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: females'] = \
```

```
                        Sum(female,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: unreported'] = \
                        Sum(unreportedGender,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = \
                        Sum(fulltime,'obsIter')
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = \
                        Sum(Weight,'obsIter')
BIOGEME_OBJECT.STATISTICS['Number of entries'] = \
                        Sum(1-exclude,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities PT'] = \
                        Sum(theWeight * prob_pt ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities CAR'] = \
                        Sum(theWeight * prob_car ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities SM'] = \
                        Sum(theWeight * prob_sm ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = Sum(fulltime,'obsIter')

# Define the dictionary for the simulation.
simulate = {'Disag. Elast. PT - Time car': elas_pt_time,
            'Disag. Elast. PT - Cost car': elas_pt_cost,
            'Disag. Elast. Car - Time PT': elas_car_time,
            'Disag. Elast. Car - Cost PT': elas_car_cost,
            'Agg. Elast. Car - Cost PT': \
                elas_car_cost * prob_car / normalization_car,
            'Agg. Elast. Car - Time PT': \
                elas_car_time * prob_car / normalization_car,
            'Agg. Elast. PT - Cost car': \
                elas_pt_cost * prob_pt / normalization_pt,
            'Agg. Elast. PT - Time car': \
                elas_pt_time * prob_pt / normalization_pt}

# Each weight is normalized so that the sum of weights is equal to the numer of en
BIOGEME_OBJECT.WEIGHT = theWeight
BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
```

A.5 05nestedElasticities.py

Available at biogeme.epfl.ch/examples/indicators/python/05nestedElasticities.py

```
## File 05nestedElasticities.py
## Simple nested logit model for the Optima case study
## Calculation of direct arc elasticities
## Thu May 11 16:38:05 2017

from biogeme import *
from headers import *
from statistics import *
from nested import *
```

```
### Three alternatives:
# CAR: automobile
# PT: public transportation
# SM: slow mode (walking, biking)

### List of parameters and their estimated value.
ASC_CAR = Beta('ASC_CAR',0.261291,-10000,10000,0,'ASC_CAR' )
ASC_SM = Beta('ASC_SM',0.0590204,-10000,10000,0,'ASC_SM' )
BETA_TIME_FULLTIME = \
 Beta('BETA_TIME_FULLTIME',-1.59709,-10000,10000,0,'BETA_TIME_FULLTIME' )
BETA_TIME_OTHER = \
 Beta('BETA_TIME_OTHER',-0.577362,-10000,10000,0,'BETA_TIME_OTHER' )
BETA_DIST_MALE = \
 Beta('BETA_DIST_MALE',-0.686327,-10000,10000,0,'BETA_DIST_MALE' )
BETA_DIST_FEMALE = \
 Beta('BETA_DIST_FEMALE',-0.83121,-10000,10000,0,'BETA_DIST_FEMALE' )
BETA_DIST_UNREPORTED = \
 Beta('BETA_DIST_UNREPORTED',-0.702974,-10000,10000,0,'BETA_DIST_UNREPORTED' )
BETA_COST = \
 Beta('BETA_COST',-0.716192,-10000,10000,0,'BETA_COST' )

###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.

### Warning: when calculation derivatives, the total formula must be
### known to Biogeme. In this case, the use of
### "DefineVariable" must be omitted, if the derivatives must be
### calculated with respect to the original variables (as is often the
### case)

 delta_dist = 1

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

 #TimeCar_scaled  = DefineVariable('TimeCar_scaled', TimeCar   /  200 )
 TimeCar_scaled  =  TimeCar   /  200

 #MarginalCostPT_scaled  = DefineVariable('MarginalCostPT_scaled', MarginalCostPT   /  10 )
 MarginalCostPT_scaled  =  MarginalCostPT   /  10

 #CostCarCHF_scaled  = DefineVariable('CostCarCHF_scaled', CostCarCHF   /  10 )
 CostCarCHF_scaled  = CostCarCHF   /  10
```

```
#distance_km_scaled  = DefineVariable('distance_km_scaled', distance_km
/ 5 )
distance_km_scaled  = distance_km   /  5
distance_km_scaled_after  = (distance_km + delta_dist)   /  5

male = DefineVariable('male',Gender == 1)
female = DefineVariable('female',Gender == 2)
unreportedGender  = DefineVariable('unreportedGender',Gender == -1)

fulltime = DefineVariable('fulltime',OccupStat == 1)
notfulltime = DefineVariable('notfulltime',OccupStat != 1)

### Definition of utility functions:

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

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: one,
      1: one,
      2: one}

### DEFINITION OF THE NESTS:
# 1: nests parameter
```

```
# 2: list of alternatives

NEST_NOCAR = Beta('NEST_NOCAR',1.52853,1,10,0,'NEST_NOCAR' )


CAR = 1.0 , [ 1]
NO_CAR = NEST_NOCAR , [ 0, 2]
nests = CAR, NO_CAR

# All observations verifying the following expression will not be
# considered for estimation
exclude = (Choice == -1)
BIOGEME_OBJECT.EXCLUDE = exclude


##
## This has been copied-pasted from the file 01nestedEstimation_param.py
##
## Code for the sensitivity analysis generated after the estimation of the model
names = ['ASC_CAR','ASC_SM','BETA_COST','BETA_DIST_FEMALE','BETA_DIST_MALE','BETA_
values = [[0.0100225,-0.0023271,0.00151986,0.00285251,0.00621963,0.00247439,0.02359
vc = bioMatrix(9,names,values)
BIOGEME_OBJECT.VARCOVAR = vc

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

prob_pt_after = nested(V_after,av,nests,0)
prob_car_after = nested(V_after,av,nests,1)
prob_sm_after = nested(V_after,av,nests,2)

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

# Defines an iterator on the data
rowIterator('obsIter')
#Statistics
nullLoglikelihood(av,'obsIter')
choiceSet = [0,1,2]
cteLoglikelihood(choiceSet,Choice,'obsIter')
availabilityStatistics(av,'obsIter')

theWeight = Weight * 1906 / 0.814484
normalization_pt = 535.086
normalization_car = 1244.77
normalization_sm = 126.147

BIOGEME_OBJECT.STATISTICS['Gender: males'] = \
```

```
                        Sum(male,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: females'] = \
                        Sum(female,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: unreported'] = \
                        Sum(unreportedGender,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = \
                        Sum(fulltime,'obsIter')
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = \
                        Sum(Weight,'obsIter')
BIOGEME_OBJECT.STATISTICS['Number of entries'] = \
                        Sum(1-exclude,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities PT'] = \
                        Sum(theWeight * prob_pt ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities CAR'] = \
                        Sum(theWeight * prob_car ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Normalization for elasticities SM'] = \
                        Sum(theWeight * prob_sm ,'obsIter')
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = Sum(fulltime,'obsIter')


# Define the dictionary for the simulation.
simulate = {'Disag. Elast. SM - Distance': elas_sm_dist,
            'Agg. Elast. SM - Distance': elas_sm_dist * prob_sm / normalization_sm

# Each weight is normalized so that the sum of weights is equal to the numer of en
BIOGEME_OBJECT.WEIGHT = theWeight
BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
```

A.6 06nestedWTP.py

Available at biogeme.epfl.ch/examples/indicators/python/06nestedWTP.py

```
## File 06nestedWTP.py
## Simple nested logit model for the Optima case study
## Thu May 11 17:23:04 2017

from biogeme import *
from headers import *
from statistics import *
from nested import *

### Three alternatives:
# CAR: automobile
# PT: public transportation
# SM: slow mode (walking, biking)

### List of parameters and their estimated value.
ASC_CAR = Beta('ASC_CAR',0.261291,-10000,10000,0,'ASC_CAR' )
```

```
ASC_SM = Beta('ASC_SM',0.0590204,-10000,10000,0,'ASC_SM' )
BETA_TIME_FULLTIME = \
 Beta('BETA_TIME_FULLTIME',-1.59709,-10000,10000,0,'BETA_TIME_FULLTIME' )
BETA_TIME_OTHER = \
 Beta('BETA_TIME_OTHER',-0.577362,-10000,10000,0,'BETA_TIME_OTHER' )
BETA_DIST_MALE = \
 Beta('BETA_DIST_MALE',-0.686327,-10000,10000,0,'BETA_DIST_MALE' )
BETA_DIST_FEMALE = \
 Beta('BETA_DIST_FEMALE',-0.83121,-10000,10000,0,'BETA_DIST_FEMALE' )
BETA_DIST_UNREPORTED = \
 Beta('BETA_DIST_UNREPORTED',-0.702974,-10000,10000,0,'BETA_DIST_UNREPORTED' )
BETA_COST = \
 Beta('BETA_COST',-0.716192,-10000,10000,0,'BETA_COST' )

###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.

### Warning: when calculation derivatives, the total formula must be
### known to Biogeme. In this case, the use of
### "DefineVariable" must be omitted, if the derivatives must be
### calculated with respect to the original variables (as is often the
### case)

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

#TimeCar_scaled  = DefineVariable('TimeCar_scaled', TimeCar   /  200 )
TimeCar_scaled  =  TimeCar   /  200

#MarginalCostPT_scaled  = DefineVariable('MarginalCostPT_scaled', MarginalCostPT   /  10 )
MarginalCostPT_scaled  =  MarginalCostPT   /  10

#CostCarCHF_scaled  = DefineVariable('CostCarCHF_scaled', CostCarCHF   /  10 )
CostCarCHF_scaled  = CostCarCHF   /  10

#distance_km_scaled  = DefineVariable('distance_km_scaled', distance_km   /  5 )
distance_km_scaled  = distance_km   /  5


male = DefineVariable('male',Gender == 1)
female = DefineVariable('female',Gender == 2)
unreportedGender  = DefineVariable('unreportedGender',Gender == -1)

fulltime = DefineVariable('fulltime',OccupStat == 1)
```

```
notfulltime = DefineVariable('notfulltime',OccupStat != 1)

### Definition of utility functions:
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

# It is advised to use the Derive operator, in order to take care
# automatically of the scaled variables.

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

# All observations verifying the following expression will not be
# considered for estimation
exclude = (Choice   ==  -1)
BIOGEME_OBJECT.EXCLUDE =  exclude


##
## This has been copied-pasted from the file 01nestedEstimation_param.py
##
## Code for the sensitivity analysis generated after the estimation of the model
names = ['ASC_CAR','ASC_SM','BETA_COST','BETA_DIST_FEMALE','BETA_DIST_MALE','BETA_
values = [[0.0100225,-0.0023271,0.00151986,0.00285251,0.00621963,0.00247439,0.02359
vc = bioMatrix(9,names,values)
BIOGEME_OBJECT.VARCOVAR = vc


# Defines an itertor on the data
rowIterator('obsIter')

theWeight = Weight * 1906 / 0.814484


BIOGEME_OBJECT.STATISTICS['Gender: males'] = \
                       Sum(male,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: females'] = \
                       Sum(female,'obsIter')
BIOGEME_OBJECT.STATISTICS['Gender: unreported'] = \
                       Sum(unreportedGender,'obsIter')
```

```
BIOGEME_OBJECT.STATISTICS['Occupation: full time'] = \
                        Sum(fulltime,'obsIter')
BIOGEME_OBJECT.STATISTICS['Sum of weights'] = \
                        Sum(Weight,'obsIter')
BIOGEME_OBJECT.STATISTICS['Number of entries'] = \
                        Sum(1-exclude,'obsIter')

simulate = {'PT: Time':TimePT,
              'PT: Value of time (CHF/min)': WTP_PT_TIME,
              'PT: Value of time (CHF/h)': 60 * WTP_PT_TIME,
              'Car: Time':TimeCar,
              'Car: Value of time (CHF/min)': WTP_CAR_TIME,
              'Car: Value of time (CHF/h)': 60 * WTP_CAR_TIME,
              'Male':male,
              'Full time':fulltime}

# Each weight is normalized so that the sum of weights is equal to the
# number of entries (1906).
BIOGEME_OBJECT.WEIGHT = theWeight
BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
```

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. (2016). Pythonbiogeme: a short introduction, *Technical Report TRANSP-OR 160706*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne.