A short introduction to PandasBiogeme

Michel Bierlaire

June 5, 2020

Report TRANSP-OR 200605
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
`transp-or.epfl.ch`

Series on Biogeme

This is an updated version of Bierlaire (2018), adapted for Biogeme 3.2.6.

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. In this document, we present step by step how to specify a simple model, estimate its parameters and interpret the output of the package. We assume that the reader is already familiar with discrete choice models (Ben-Akiva and Lerman, 1985), 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. This is the standard mode of operations of more and more data scientists. The syntax for model specification is almost identical, but there are slight differences, that are highlighted at the end of the document. This document has been written using PandasBiogeme 3.2.6, but should remain valid for future versions.

1 Data

Biogeme assumes that a Pandas database is available, containing only numerical entries. Each column corresponds to a variable, each row to an observation.

If you are not familiar with Pandas, prepare a file that contains in its first line a list of labels corresponding to the available data, and that each subsequent line contains the exact same number of numerical data, each row corresponding to an observation. Delimiters can be tabs or spaces.

The data file used for this example is `swissmetro.dat`. It can be downloaded from the "Data" section of biogeme.epfl.ch.

2 Python

PandasBiogeme is a package of the Python programming language. Therefore, estimating a model amounts to writing a script in Python. Online tutorials and documentation about Python can easily be found. Although it is not necessary to master the Python language to specify models for Biogeme, it would definitely help to learn at least the basics. In this Section, we report some useful information when using the package Biogeme.

- Two versions of Python are commonly used: 2 and 3. Biogeme works only with Python version 3.

- Python is available on Linux, MacOSX and Windows. PandasBiogeme is platform independent.
- The syntax of Python is case sensitive. It means that `varname` and `Varname`, for instance, would represent two different entities.
- The indentation of the code is important in Python. It is advised to use a text editor that has a "Python mode" to help managing these indentations.
- A Python statement must be on a single line. Sometimes, for the sake of readability, it is convenient to split the statement on several lines. In that case, the character \ must be inserted at the end of a line to inform Python that the statement continues at the following line. There are several examples below, for instance in the specification of the utility functions.

3 The model

The model is a logit model with 3 alternatives: *train*, *Swissmetro* and *car*. The utility functions are defined as:

```
V1 = ASC_TRAIN + \
     B_TIME * TRAIN_TT_SCALED + \
     B_COST * TRAIN_COST_SCALED
V2 = ASC_SM + \
     B_TIME * SM_TT_SCALED + \
     B_COST * SM_COST_SCALED
V3 = ASC_CAR + \
     B_TIME * CAR_TT_SCALED + \
     B_COST * CAR_CO_SCALED
```

where

- `TRAIN_TT_SCALED`,
- `TRAIN_COST_SCALED`,
- `SM_TT_SCALED`,
- `SM_COST_SCALED`,
- `CAR_TT_SCALED`,
- `CAR_CO_SCALED`

are variables, and

- ASC_TRAIN,
- ASC_SM,
- ASC_CAR,
- B_TIME,
- B_COST

are parameters to be estimated. Note that it is not possible to identify all alternative specific constants ASC_TRAIN, ASC_SM, ASC_CAR from data. Consequently, ASC_SM is normalized to 0.

The availability of an alternative i is determined by the variable y_i, i=1, 2, 3, which is equal to 1 if the alternative is available, and 0 otherwise. The probability of choosing an available alternative i is given by the logit model:

$$P(i|\{1,2,3\};x,\beta) = \frac{y_i e^{V_i(x,\beta)}}{y_1 e^{V_1(x,\beta)} + y_2 e^{V_2(x,\beta)} + y_3 e^{V_3(x,\beta)}}. \tag{1}$$

Given a data set of N observations, the log likelihood of the sample is

$$\mathcal{L} = \sum_n \log P(i_n|\{1,2,3\};x_n,\beta) \tag{2}$$

where i_n is the alternative actually chosen by individual n, and x_n are the explanatory variables associated with individual n.

4 Model specification: PandasBiogeme

The model specification file must have an extension .py. The file 01logit.py is reported in Section A.1. We describe here its content.

The objective is to provide to PandasBiogeme the formula of the log likelihood function to maximize, using a syntax based on the Python programming language, and extended for the specific needs of Biogeme. The file can contain comments, designed to document the specification. Single-line comments are included using the characters #, consistently with the Python syntax. All characters after this command, up to the end of the current line, are ignored by Python. Multiple lines comments are created by adding a delimiter (""") at the beginning and the end of the comment. In our example, the file starts with comments describing the name of the file, its author and the date when it was created. A short description of its content is also provided.

```
"""File 01logit.py

:author: Michel Bierlaire, EPFL
:date: Thu Sep  6 15:14:39 2018

 Example of a logit model.
 Three alternatives: Train, Car and Swissmetro
 SP data
"""
```

These comments are completely ignored by Python. However, it is recommended to use many comments to describe the model specification, for future reference, or to help other persons to understand the specification.

The specification file must start by loading the Python libraries needed by PandasBiogeme. The following libraries must be loaded:

- `pandas`, the generic package for data management,
- `biogeme.database`, the Biogeme module for data management,
- `biogeme.biogeme`, the core Biogeme module,
- `biogeme.models`, the Biogeme module for choice models.

It is custom in Python to use shortcuts to simplify the syntax. Here, we use `pd`, `db`, `bio`, and `models`, respectively. Finally, we need to import the expressions to build the model specification. In this example, we use only the expression `Beta` that defines parameters to be estimated.

```
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
from biogeme.expressions import Beta
```

The next step consists in preparing the Pandas database. If you have a data file formatted for previous versions of Biogeme, this can easily be done using the following statements:

```
df = pd.read_csv('swissmetro.dat', '\t')
database = db.Database('swissmetro', df)
```

The first statement reads the data from the file, using tabs as delimiters. It stores it in a Pandas data structure. The second statement prepares the database for Biogeme. Clearly, if you prefer to create your Pandas database in another way, it is possible. In that case, you still have to use the second statement to transfer the Pandas database to Biogeme.

The name of the columns in the database characterize the variables for your model. In order to make them available as a Python variable, the following statement must be included:

```
globals().update(database.variables)
```

It is possible to tell PandasBiogeme to ignore some observations in the data file. A boolean expression must be defined, that is evaluated for each observation in the data file. Each observation such that this expression is "true" is discarded from the sample. In our example, the modeler has developed the model only for work trips, so that every observation such that the trip purpose is not 1 or 3 is removed.

Observations such that the dependent variable `CHOICE` is 0 are also removed. The convention is that "false" is represented by 0, and "true" by 1, so that the '*' can be interpreted as a "and", and the '+' as a "or". Note also that the result of the '+' can be 2, so that we test if the result is equal to 0 or not. The exclude condition in our example is therefore interpreted as: either (`PURPOSE` different from 1 and `PURPOSE` different from 3), or `CHOICE` equal to 0.

```
exclude = ((PURPOSE != 1) * (PURPOSE != 3) + (CHOICE == 0)) > 0
database.remove(exclude)
```

- We have conveniently used an intermediary Python variable `exclude` in this example. It is not necessary. The above statement is completely equivalent to

  ```
  database.remove((( PURPOSE != 1 ) * (  PURPOSE   !=  3  ) +\
                   ( CHOICE == 0 )) > 0)
  ```

- The same result can be obtained using Pandas directly, using the following syntax:

  ```
  remove = (((database.data.PURPOSE != 1) & \
             (database.data.PURPOSE != 3)) | \
             (database.data.CHOICE == 0))
  database.data.drop(database.data[remove].index,inplace=True)
  ```

  Pandas provides more powerful tools to manage the database. If you need to perform sophisticated data manipulations, it is advised to use Pandas instead of Biogeme for these purposes. Refer to the online Pandas documentation and the many tutorials available online.

The next statements use the function `Beta` to define the parameters to be estimated. For each parameter, the following information must be mentioned:

1. the name of the parameter,
2. the default value,
3. a lower bound (or `None`, if no bound is specified),
4. an upper bound, (or `None`, if no bound is specified),
5. a flag that indicates if the parameter must be estimated (0) or if it keeps its default value (1).

```
ASC_CAR = Beta('ASC_CAR',0,None,None,0)
ASC_TRAIN = Beta('ASC_TRAIN',0,None,None,0)
ASC_SM = Beta('ASC_SM',0,None,None,1)
B_TIME = Beta('B_TIME',0,None,None,0)
B_COST = Beta('B_COST',0,None,None,0)
```

- In Python, case sensitivity is enforced, so that `varname` and `Varname` would represent two different variables. In our example, the default value of each parameter is 0. If a previous estimation had been performed before, we could have used the previous estimates as default value.

- For the parameters that are estimated by PandasBiogeme, the default value is used as the starting value for the optimization algorithm. For the parameters that are not estimated, the default value is used throughout the estimation process. In our example, the parameter `ASC_SM` is not estimated (as specified by the `1` in the fifth argument on the corresponding line), and its value is fixed to `0`.

- A lower bound and an upper bound must be specified. If no bound is meaningful, use `None`.

- Nothing prevents to write

  ```
  car_cte = Beta('ASC_CAR',0,None, None,0)
  ```

  and to use `car_cte` later in the specification. We **strongly** advise against this practice, and suggest to use the exact same name for the Python variable on the left hand side, and for the PandasBiogeme variable, appearing as the first argument of the function, as illustrated in this example.

It is possible to define new variables in addition to the variables defined in the data files.

```
SM_COST = SM_CO * (GA == 0)
TRAIN_COST = TRAIN_CO * (GA == 0)
CAR_AV_SP = CAR_AV * (SP != 0)
TRAIN_AV_SP = TRAIN_AV * (SP != 0)
```

When boolean expressions are involved, the value True is represented by 1, and the value False is represented by 0. Therefore, a multiplication involving a boolean expression is equivalent to a "and" operator. The above code is interpreted in the following way:

- `CAR_AV_SP` is equal to `CAR_AV` if `SP` is different from 0, and is equal to 0 otherwise. `TRAIN_AV_SP` is defined similarly.
- `SM_COST` is equal to `SM_CO` if `GA` is equal to 0, that is, if the traveler does not have a yearly pass (called "general abonment"). If the traveler possesses a yearly pass, then `GA` is different from 0, and the variable `SM_COST` is zero. The variable `TRAIN_COST` is defined in the same way.

Variables can be also be rescaled. For numerical reasons, it is good practice to scale the data so that the values of the estimated parameters are around 1. A previous estimation with the unscaled data has generated parameters around -0.01 for both cost and time. Therefore, time and cost are divided by 100.

```
TRAIN_TT_SCALED = TRAIN_TT / 100
TRAIN_COST_SCALED = TRAIN_COST / 100
SM_TT_SCALED = SM_TT / 100
SM_COST_SCALED = SM_COST / 100
CAR_TT_SCALED = CAR_TT / 100
CAR_CO_SCALED = CAR_CO / 100
```

We now write the specification of the utility functions.

```
V1 = ASC_TRAIN + \
     B_TIME * TRAIN_TT_SCALED + \
     B_COST * TRAIN_COST_SCALED
V2 = ASC_SM + \
     B_TIME * SM_TT_SCALED + \
     B_COST * SM_COST_SCALED
V3 = ASC_CAR + \
     B_TIME * CAR_TT_SCALED + \
     B_COST * CAR_CO_SCALED
```

We need to associate each utility function with the number, the identifier, of the alternative, using the same numbering convention as in the data file. In this example, the convention is described in Table 1.

To do this, we use a Python dictionary:

Train	1
Swissmetro	2
Car	3

Table 1: Numbering of the alternatives

```
V = {1: V1,
     2: V2,
     3: V3}
```

We use also a dictionary to describe the availability conditions of each alternative:

```
av = {1: TRAIN_AV_SP,
      2: SM_AV,
      3: CAR_AV_SP}
```

We now define the choice model. The function `models.loglogit` provides the logarithm of the choice probability of the logit model. It takes three arguments:

1. the dictionary describing the utility functions,
2. the dictionary describing the availability conditions,
3. the alternative for which the probability must be calculated.

In this example, we obtain

```
logprob = models.loglogit(V, av, CHOICE)
```

We are now ready to create the `BIOGEME` object, using the following syntax:

```
biogeme = bio.BIOGEME(database, logprob)
```

The constructor accepts two mandatory arguments:

- the database object containing the data,
- the formula for the contribution to the log likelihood of each row in the database.

It is advised to give a name to the model using the following statement:

```
biogeme.modelName = '01logit'
```

The estimation of the model parameters is performed using the following statement.

```
results = biogeme.estimate()
```

5 Running PandasBiogeme

The script is executed like any python script. Typically, by typing

```
python 01logit.py
```

is a terminal, or by typing "shift-return" in a Jupyter notebook.

By default, running PandasBiogeme is silent, in the sense that it does not produce any output. Two files are generated:

- `01logit.html` reports the results of the estimation is HTML format, and can be opened in your favorite browser.
- `01logit.pickle` is a snapshot of the results of the estimation, and can be used in another Python script.

In order to avoid erasing previously generated results, the name of the files may vary from one run to the next. Therefore, it is important to verify the latest files created in the directory.

You can also print the name of the files that were actually created using the following Python statement:

```
print(f'HTML file:   {results.data.htmlFileName}')
print(f'Pickle file: {results.data.pickleFileName}')
```

6 PandasBiogeme: the report file

The report file generated by PandasBiogeme gathers information about the result of the estimation. First, some information about the version of Biogeme, and some links to relevant URLs is provided. Next, the name of the report file and the name of the database are reported.

The estimation report follows, including

- The number of parameters that have been estimated.
- The sample size, that is, the number of rows in the data file that have not been excluded.
- The number of excluded observations.
- `Init log likelihood` is the log likelihood $\mathcal{L}^i$ of the sample for the model defined with the default values of the parameters.
- `Final log likelihood` is the log likelihood $\mathcal{L}^*$ of the sample for the estimated model.
- `Likelihood ratio test for the init. model` is

$$-2(\mathcal{L}^i - \mathcal{L}^*) \tag{3}$$

  where $\mathcal{L}^i$ is the log likelihood of the init model as defined above, and $\mathcal{L}^*$ is the log likelihood of the sample for the estimated model.

- `Rho-square for the init. model` is

$$\rho^2 = 1 - \frac{\mathcal{L}^*}{\mathcal{L}^i}. \tag{4}$$

- `Rho-square-bar for the init. model` is

$$\rho^2 = 1 - \frac{\mathcal{L}^* - K}{\mathcal{L}^i}. \tag{5}$$

  where K is the number of estimated parameters.

- `Akaike Information Criterion` is:

$$2K - 2\mathcal{L}^*, \tag{6}$$

  where K is the number of estimated parameters.

- `Bayesian Information Criterion` is:

$$-2\mathcal{L}^* + \mathsf{K}\ln(\mathsf{N}), \tag{7}$$

  where K is the number of estimated parameters, and N is the sample size.

- `Final gradient norm` is the gradient of the log likelihood function computed for the estimated parameters.

- `Nbr of threads` is the number of processors used by Biogeme to calculate the log likelihood at each iteration.

- `Algorithm` is the optimization algorithm used to solve the maximum likelihood estimation problem.

- `Proportion analytical hessian` is the proportion of iterations where the analytical second derivatives matrix (called "hessian") has been calculated.

- `Relative projected gradient` is the norm of the projected gradient, scaled to account for the level of magnitude of the log likelihood. This quantity is used as stopping criterion for the algorithm.

- `Number of iterations` is the number of iterations performed by the optimization algorithms.

- `Number of function evaluations` reports the number of times that the log likelihood function has been calculated.

- `Number of gradient evaluations` reports the number of times that the gradient of the log likelihood function has been calculated.

- `Number of hessian evaluations` reports the number of times that the second derivatives matrix (or hessian) of the log likelihood function has been calculated.

- `Cause of termination` provides the reason why the optimization algorithm has stopped.

- `Optimization time` is the actual time used by the algorithm.

The following section reports the estimates of the parameters of the utility function, together with some statistics. For each parameter β_k, the following is reported:

- The name of the parameter.
- The estimated value β_k.
- The standard error σ_k of the estimate, calculated as the square root of the k^{th}diagonal entry of the Rao-Cramer bound (see Appendix B).
- The t statistics, calculated as $t_k = \beta_k/\sigma_k$.
- The p value, calculated as $2(1 - \Phi(t_k))$, where $\Phi(\cdot)$ is the cumulative distribution function of the univariate standard normal distribution.
- The robust standard error σ_k^R of the estimate, calculated as the square root of the k^{th}diagonal entry of the robust estimate of the variance covariance matrix. (see Appendix B).
- The robust t statistics, calculated as $t_k^R = \beta_k/\sigma_k^R$.
- The robust p value, calculated as $2(1 - \Phi(t_k^R))$, where $\Phi(\cdot)$ is the cumulative density function of the univariate normal distribution.

The last section reports, for each pair of parameters k and ℓ,

- the name of β_k,
- the name of β_ℓ,
- the entry $\Sigma_{k,\ell}$ of the Rao-Cramer bound (see Appendix B),
- the correlation between β_k and β_ℓ, calculated as

$$\frac{\Sigma_{k,\ell}}{\sqrt{\Sigma_{k,k}\Sigma_{\ell,\ell}}}, \tag{8}$$

- the t statistics, calculated as

$$t_{k,\ell} = \frac{\beta_k - \beta_\ell}{\sqrt{\Sigma_{k,k} + \Sigma_{\ell,\ell} - 2\Sigma_{k,\ell}}}, \tag{9}$$

- the p value, calculated as $2(1 - \Phi(t_{k,\ell}))$, where $\Phi(\cdot)$ is the cumulative density function of the univariate standard normal distribution,
- the entry $\Sigma_{k,\ell}^R$ of Σ^R, the robust estimate of the variance covariance matrix (see Appendix B),

- the robust correlation between β_k and β_ℓ, calculated as

$$\frac{\Sigma^R_{k,\ell}}{\sqrt{\Sigma^R_{k,k}\Sigma^R_{\ell,\ell}}}, \tag{10}$$

- the robust t statistics, calculated as

$$t^R_{k,\ell} = \frac{\beta_k - \beta_\ell}{\sqrt{\Sigma^R_{k,k} + \Sigma^R_{\ell,\ell} - 2\Sigma^R_{k,\ell}}}, \tag{11}$$

- the robust p value, calculated as $2(1-\Phi(t^R_{k,\ell}))$, where $\Phi(\cdot)$ is the cumulative density function of the univariate standard normal distribution,

The final lines report the value of the smallest and the largest eigenvalues, as well as the ratio between the two, called the "condition number". If smallest eigenvalue is close to zero, it is a sign of singularity, that may be due to a lack of variation in the data or an unidentified model.

7 The results as Python variables

The estimation function returns an object that contains the results of the estimation as well as the associated statistics. This object can be printed on screen:

```
print("Results=",results)
```

If `results` is the object returned by the estimation function, the results of the estimation can be accessed in `results.data`:

- `results.data.modelName`: the model name.
- `results.data.nparam`: the number K of estimated parameters.
- `results.data.betaValues`: a Numpy array containing the estimated values of the parameters, in an arbitrary order.
- `results.data.betaNames`: a list containing the name of the estimated parameters, in the same order as the values above.
- `results.data.initLogLike`: the value $\mathcal{L}^i$ is the initial log likelihood.
- `results.data.betas`: a list of objects corresponding to the parameters. Each of these objects contains the following entries, which should be self explanatory.

- beta.name,
- beta.value,
- beta.stdErr,
- beta.lb,
- beta.ub,
- beta.tTest,
- beta.pValue,
- beta.robust_stdErr,
- beta.robust_tTest,
- beta.robust_pValue,
- beta.bootstrap_stdErr,
- beta.bootstrap_tTest,
- beta.bootstrap_pValue.

- `results.data.logLike`: the value $\mathcal{L}^*$ of the log likelihood at the final value of the parameters.
- `results.data.g`: the gradient of the log likelihood at the final value of the parameters.
- `results.data.H`: the second derivatives matrix of the log likelihood at the final value of the parameters.
- `results.data.bhhh`: the BHHH matrix (16) at the final value of the parameters.
- `results.data.dataname`: the name of the database.
- `results.data.sampleSize`: the sample size N.
- `results.data.numberOfObservations`: the number of rows in the data file. If the data is not panel, it is the same as the sample size.
- `results.data.monteCarlo`: a boolean that is True if the model involves Monte-Carlo simulation for the calculation of integrals.
- `results.data.numberOfDraws`: number of draws used for Monte-Carlo simulation.
- `results.data.typesOfDraws`: type of draws used for Monte-Carlo simulation.

- `results.data.excludedData`: number of excluded data.
- `results.data.dataProcessingTime`: time needed to process the data before estimation.
- `results.data.drawsProcessingTime`: time needed to generate the draws for Monte-Carlo simulation.
- `results.data.optimizationTime`: time used by the optimization algorithm.
- `results.data.gradientNorm`: norm of the gradient of the log likelihood at the final value of the parameters.
- `results.data.optimizationMessages`: message returned by the optimization routine.
- `results.data.numberOfFunctionEval`: number of time the log likelihood function has been evaluated.
- `results.data.numberOfIterations`: number of iterations of the optimization algorithm.
- `results.data.numberOfThreads`: number of processors used.
- `results.data.htmlFileName`: name of the HTML file.
- `results.data.pickleFileName`: name of the Pickle file.
- `results.data.bootstrap`: a boolean that is True if the calculation of statistics using bootstrapping has been requested.
- `results.data.bootstrapTime`: the time needed for calculating the statistics with bootstrapping, if applicable.

In addition the robust variance-covariance matrix can be obtained using

```
results.data.getRobustVarCovar()
```

If you are just interested in the estimates of the parameters, they can be obtained as a `dict`:

```
betas = results.getBetaValues()
for k,v in betas.items():
    print(f"{k}=\t{v:.3g}")
```

The general statistics can also be obtained as a `dict`:

```
gs = results.getGeneralStatistics()
```

The results can also be obtained as a Pandas data frame:

```
pandasResults = results.getEstimatedParameters()
```

and

```
correlationResults = results.getCorrelationResults()
```

References

Ben-Akiva, M. E. and Lerman, S. R. (1985). *Discrete Choice Analysis: Theory and Application to Travel Demand*, MIT Press, Cambridge, Ma.

Berndt, E. K., Hall, B. H., Hall, R. E. and Hausman, J. A. (1974). Estimation and inference in nonlinear structural models, *Annals of Economic and Social Measurement* **3/4**: 653–665.

Bierlaire, M. (2018). PandasBiogeme: a short introduction, *Technical Report TRANSP-OR 181219*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne.
URL: *transp-or.epfl.ch/documents/technicalReports/Bier18.pdf*

Kauermann, G. and Carroll, R. (2001). A note on the efficiency of sandwich covariance matrix estimation, *Journal of the American Statistical Association* **96**(456).

White, H. (1982). Maximum likelihood estimation of misspecified models, *Econometrica* **50**: 1–25.

A Complete specification file

A.1 `01logit.py`

```python
"""File 01logit.py

:author: Michel Bierlaire, EPFL
:date: Thu Sep  6 15:14:39 2018

 Example of a logit model.
 Three alternatives: Train, Car and Swissmetro
 SP data
"""

import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
from biogeme.expressions import Beta

# Read the data
df = pd.read_csv('swissmetro.dat', '\t')
database = db.Database('swissmetro', df)

# The following statement allows you to use the names of the
# variable as Python variable.
globals().update(database.variables)

# Removing some observations
exclude = ((PURPOSE != 1) * (PURPOSE != 3) + (CHOICE == 0)) > 0
database.remove(exclude)

# Parameters to be estimated
ASC_CAR = Beta('ASC_CAR', 0, None, None, 0)
ASC_TRAIN = Beta('ASC_TRAIN', 0, None, None, 0)
ASC_SM = Beta('ASC_SM', 0, None, None, 1)
B_TIME = Beta('B_TIME', 0, None, None, 0)
B_COST = Beta('B_COST', 0, None, None, 0)


# Definition of new variables
SM_COST = SM_CO * (GA == 0)
TRAIN_COST = TRAIN_CO * (GA == 0)
CAR_AV_SP = CAR_AV * (SP != 0)
TRAIN_AV_SP = TRAIN_AV * (SP != 0)
TRAIN_TT_SCALED = TRAIN_TT / 100
TRAIN_COST_SCALED = TRAIN_COST / 100
SM_TT_SCALED = SM_TT / 100
```

```
SM_COST_SCALED = SM_COST / 100
CAR_TT_SCALED = CAR_TT / 100
CAR_CO_SCALED = CAR_CO / 100

# Definition of the utility functions
V1 = ASC_TRAIN + \
     B_TIME * TRAIN_TT_SCALED + \
     B_COST * TRAIN_COST_SCALED
V2 = ASC_SM + \
     B_TIME * SM_TT_SCALED + \
     B_COST * SM_COST_SCALED
V3 = ASC_CAR + \
     B_TIME * CAR_TT_SCALED + \
     B_COST * CAR_CO_SCALED

# Associate utility functions with the numbering of alternatives
V = {1: V1,
     2: V2,
     3: V3}

# Associate the availability conditions with the alternatives
av = {1: TRAIN_AV_SP,
      2: SM_AV,
      3: CAR_AV_SP}

# Definition of the model. This is the contribution of each
# observation to the log likelihood function.
logprob = models.loglogit(V, av, CHOICE)

# Create the Biogeme object
biogeme = bio.BIOGEME(database, logprob)
biogeme.modelName = '01logit'

# Estimate the parameters
results = biogeme.estimate()

# Get the results in a pandas table
pandasResults = results.getEstimatedParameters()
print(pandasResults)
```

B Estimation of the variance-covariance matrix

Under relatively general conditions, the asymptotic variance-covariance matrix of the maximum likelihood estimates of the vector of parameters $\theta \in \mathbb{R}^K$ is given by the Cramer-Rao bound

$$-\mathrm{E}\left[\nabla^2 \mathcal{L}(\theta)\right]^{-1} = \left\{-\mathrm{E}\left[\frac{\partial^2 \mathcal{L}(\theta)}{\partial\theta\partial\theta^T}\right]\right\}^{-1}. \qquad (12)$$

The term in square brackets is the matrix of the second derivatives of the log likelihood function with respect to the parameters evaluated at the true parameters. Thus the entry in the kth row and the ℓth column is

$$\frac{\partial^2 \mathcal{L}(\theta)}{\partial\theta_k \partial\theta_\ell}. \qquad (13)$$

Since we do not know the actual values of the parameters at which to evaluate the second derivatives, or the distribution of x_{in} and x_{jn} over which to take their expected value, we estimate the variance-covariance matrix by evaluating the second derivatives at the estimated parameters $\widehat{\theta}$ and the sample distribution of x_{in} and x_{jn} instead of their true distribution. Thus we use

$$\mathrm{E}\left[\frac{\partial^2 \mathcal{L}(\theta)}{\partial\theta_k \partial\theta_\ell}\right] \approx \sum_{n=1}^{N} \left[\frac{\partial^2 \left(y_{in} \ln P_n(i) + y_{jn} \ln P_n(j)\right)}{\partial\theta_k \partial\theta_\ell}\right]_{\theta=\widehat{\theta}}, \qquad (14)$$

as a consistent estimator of the matrix of second derivatives.

Denote this matrix as $\widehat{A}$. Note that, from the second order optimality conditions of the optimization problem, this matrix is negative semi-definite, which is the algebraic equivalent of the local concavity of the log likelihood function. If the maximum is unique, the matrix is negative definite, and the function is locally strictly concave.

An estimate of the Cramer-Rao bound (12) is given by

$$\widehat{\Sigma}_\theta^{\mathrm{CR}} = -\widehat{A}^{-1}. \qquad (15)$$

If the matrix $\widehat{A}$ is negative definite then $-\widehat{A}$ is invertible and the Cramer-Rao bound is positive definite.

Another consistent estimator of the (negative of the) second derivatives matrix can be obtained by the matrix of the cross-products of first derivatives as follows:

$$-\mathsf{E}\left[\frac{\partial^2 \mathcal{L}(\theta)}{\partial\theta\partial\theta^T}\right] \approx \sum_{n=1}^{n} \left(\frac{\partial \ell_n(\widehat{\theta})}{\partial\theta}\right)\left(\frac{\partial \ell_n(\widehat{\theta})}{\partial\theta}\right)^T = \widehat{B}, \qquad (16)$$

where

$$\left(\frac{\partial \ell_n(\widehat{\theta})}{\partial \theta}\right) = \frac{\partial}{\partial \theta}(\log \mathrm{P}(i_n|\mathcal{C}_n;\widehat{\theta})) \tag{17}$$

is the gradient vector of the likelihood of observation n. This approximation is employed by the BHHH algorithm, from the work by Berndt et al. (1974). Therefore, an estimate of the variance-covariance matrix is given by

$$\widehat{\Sigma}_{\theta}^{\mathrm{BHHH}} = \widehat{B}^{-1}, \tag{18}$$

although it is rarely used. Instead, $\widehat{B}$ is used to derive a third consistent estimator of the variance-covariance matrix of the parameters, defined as

$$\widehat{\Sigma}_{\theta}^{\mathrm{R}} = (-\widehat{A})^{-1}\ \widehat{B}\ (-\widehat{A})^{-1} = \widehat{\Sigma}_{\theta}^{\mathrm{CR}}\ (\widehat{\Sigma}_{\theta}^{\mathrm{BHHH}})^{-1}\ \widehat{\Sigma}_{\theta}^{\mathrm{CR}}. \tag{19}$$

It is called the *robust* estimator, or sometimes the *sandwich* estimator, due to the form of equation (19). Biogeme reports statistics based on both the Cramer-Rao estimate (15) and the robust estimate (19).

When the true likelihood function is maximized, these estimators are asymptotically equivalent, and the Cramer-Rao bound should be preferred (Kauermann and Carroll, 2001). When other consistent estimators are used, the robust estimator must be used (White, 1982). Consistent non-maximum likelihood estimators, known as pseudo maximum likelihood estimators, are often used when the true likelihood function is unknown or difficult to compute. In such cases, it is often possible to obtain consistent estimators by maximizing an objective function based on a simplified probability distribution.

C Differences with PythonBiogeme

The syntax of PandasBiogeme has been designed to be as close as possible to the syntax of PythonBiogeme. There are some differences though that we mention in this Section.

- There is no need anymore to specify an iterator.
- The `BIOGEME_OBJECT` and its variables (`ESTIMATE`, `PARAMETERS`, etc.) are obsolete.
- The exclusion of data was done as follows in PythonBiogeme:

```
exclude = (( PURPOSE != 1 ) * (  PURPOSE   !=  3  ) +\
           ( CHOICE == 0 )) > 0
BIOGEME_OBJECT.EXCLUDE = exclude
```

It is done as follows is PandasBiogeme:

```
exclude = (( PURPOSE != 1 ) * (   PURPOSE   !=   3   ) +\
           ( CHOICE == 0 )) > 0
database.remove(exclude)
```

- For the specification of the parameters using the `Beta` function, the PythonBiogeme syntax is still valid here. But it is slightly extended. In PythonBiogeme, it was mandatory to explicitly specify a lower and an upper bound. In PandasBiogeme, it is now possible to specify `None` if no bound is desired. Note that, in PythonBiogeme, the last argument of the `Beta` function allowed to give a text description of the parameter. This argument can still be provided (for compatibility reasons), but is ignored by PandasBiogeme.

- `DefineVariable`: the syntax is similar to PythonBiogeme, but not identical. The function `DefineVariable` requires a third argument, which is the name of the database. This allows to work with different databases in the same specification file.

- The name of the output files is defined by the statement

  ```
  biogeme.modelName = "01logit"
  ```

  In PythonBiogeme, it was defined by the name of the script. In PandasBiogeme, as it is technically possible to define several models in the same script, the name has to be explicitly mentioned.

- As discussed above, the estimation results are available in a Python object. This object is actually saved in a file with the extension `pickle`. This file can be read using the following statements:

  ```
  import biogeme.results as res
  results = res.bioResults(pickleFile=01logit.pickle')
  ```

  and the object `results` is recovered exactly how it was generated after the estimation.