Assisted specification with Biogeme 3.2.12

Michel Bierlaire Nicola Ortelli

August 16, 2023

Report TRANSP-OR 230816
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
`transp-or.epfl.ch`

Series on Biogeme

This document is an updated version of Bierlaire and Ortelli (2022), adapted to version 3.2.12 of Biogeme.

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. It is a Python package written in Python and C++, that relies on the Pandas library for the management of the data.

This document describes how to obtain assistance from Biogeme for the model specification. In particular, it shows how to apply the algorithm described by Ortelli et al. (2021). In a nutshell, an optimization algorithm is used to generate models based on a minimal number of inputs provided by the analyst. These inputs are used to build a space of possible specifications that may contain any form of variable interaction, nonlinear transformation, segmentation of the population and potential choice models; the space is then explored by an algorithm that sequentially introduces small modifications to an initial set of promising specifications.

We assume that the reader is already familiar with discrete choice models and Biogeme. This document has been written using Biogeme 3.2.12.

We use the Swissmetro example throughout the document. The Python scripts are available on GitHub in the biogeme repository, in the directory examples/assisted. They are also reported in the Appendix.

1 Catalogs

The philosophy of the assisted specification is that the analyst may have several specifications in mind, but does not know a priori which one is the most appropriate. Biogeme can then accept as input a "catalog" of different specifications, and estimate all specifications in the catalog, and provide a comparative report of the estimation results. It provides a great flexibility to the analyst who can replace any expression of the model by such a catalog, as illustrated with the examples in this document.

In some cases, the number of possible specifications is so high that an exhaustive enumeration is not feasible. In that case, the algorithm proposed by Ortelli et al. (2021) is applied in order to investigate a subset of potentially promising specifications.

The code used to generate the examples presented in this Section is available in Appendix 3.

Each catalog is associated with a unique name, and a list of different valid expressions, each of them also associated with a name. For instance, suppose that we want to define a catalog that contains both a logit and a nested logit models.

We first define each of the models, like in a regular Biogeme script:

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

and

```
logprob_nested = models.lognested(V, av, nests, CHOICE)
```

The catalog can then be defined using the following syntax, that is self-explanatory:

```
model_catalog = Catalog.from_dict(
    catalog_name='model_catalog',
    dict_of_expressions={
        'logit': logprob_logit,
        'nested': logprob_nested
    },
)
```

Note that the `Catalog` class must first be imported using the following syntax:

```
from biogeme.catalog import Catalog
```

A **catalog** is a regular Biogeme expression, that can be used in another expression. At each given point in time, exactly one of the expressions of the catalog is active, and used for the evaluation of the expression. For instance, if we print the catalog above, it corresponds to the logit specification by default:

```
print(model_catalog)
[model_catalog: logit]...
```

where the ellipsis is the actual expression of the logit model (which is too long to report in this document). In order to modify the configuration of a catalog, Biogeme uses a **controller**, that is accessible using the `controlled_by` attribute of the catalog. For instance, in order to activate the nested logit specification, we need to write

```
model_catalog.controlled_by.set_name('nested')
```

Now, if we print the catalog again, we obtain

```
print(model_catalog)
[model_catalog: nested]...
```

where the ellipsis is the actual expression of the nested logit model.

In general, there is no need to explicitly access to the controller, as Biogeme provides high level access to the catalog. The simplest one is an iterator:

```
for specification in model_catalog:
    print(specification)
```

provides the following output:

```
[model_catalog: nested]...
[model_catalog: logit]...
```

For the sake of this document, instead of listing the expressions themselves (which can be long and complicated), we report the configuration identifiers of the controller, that identifies all possible specifications associated with a catalog. This is usually not needed by regular users. The function used to do that is described in Appendix 2. For the `model_catalog`, it gives the following output:

```
model_catalog:logit
model_catalog:nested
```

Also, the Biogeme object has a function called `estimate_catalog`, that iterates on all specifications in a catalog (if possible), and estimate the corresponding models. If there are too many specifications to be enumerated, it launches the assisted specification algorithm if not. This function is illustrated in Section 3.

1.1 Synchronized catalogs

A catalog can be used for alternative nonlinear specifications of a variable. Here, we use the example of the train travel time, in the Swissmetro example. Again, we first define each specification separately:

1. the linear specification:

```
linear_train_tt = TRAIN_TT
```

2. the Box-Cox transform:

```
ell_travel_time = Beta('lambda_travel_time', 1, -10, 10, 0)
boxcox_train_tt = boxcox(TRAIN_TT_SCALED, ell_travel_time)
```

3. the squared variable:

```
squared_train_tt = TRAIN_TT * TRAIN_TT
```

Note that the `boxcox` function must first be imported as follows:

```
from biogeme.models import boxcox
```

The catalog can be defined, using the same syntax as above:

```
train_tt_catalog = Catalog.from_dict(
    catalog_name='train_tt_catalog',
    dict_of_expressions={
        'linear': linear_train_tt,
        'boxcox': boxcox_train_tt,
        'squared': squared_train_tt,
    },
)
```

The catalog can be used as a regular expression in the definition of the utility function, for instance:

```
V_TRAIN = ASC_TRAIN + B_TIME * train_tt_catalog + ...
```

Note that, because `V_TRAIN` contains a catalog, it is possible to iterate through its specifications as well:

```
for specification in V_TRAIN:
    print(specification)
```

generates the following output:

```
(ASC_TRAIN(init=0) + (B_TIME(init=0) * [train_tt_catalog:
   boxcox]...
(ASC_TRAIN(init=0) + (B_TIME(init=0) * [train_tt_catalog:
   linear]TRAIN_TT))
(ASC_TRAIN(init=0) + (B_TIME(init=0) * [train_tt_catalog:
   squared](TRAIN_TT * TRAIN_TT)))
```

where the ellipsis is replaced by the complete specification of the Box-Cox model.

Now, we would like to specify a similar catalog for the car travel time, in the same model. We apply the exact same syntax as above:

```
CAR_TT = Variable('CAR_TT')
linear_car_tt = CAR_TT
boxcox_car_tt = boxcox(CAR_TT, ell_travel_time)
squared_car_tt = CAR_TT * CAR_TT
car_tt_catalog = Catalog.from_dict(
    catalog_name='car_tt_catalog',
    dict_of_expressions={
        'linear': linear_car_tt,
        'boxcox': boxcox_car_tt,
        'squared': squared_car_tt,
    },
)
```

In order to illustrate how those catalogs are combined, we build a dummy expression that calculates their sum:

```
dummy_expression = train_tt_catalog + car_tt_catalog
```

If we print all possible configurations, we obtain nine combinations (the order in which they appear is irrelevant):

```
car_tt_catalog:linear;train_tt_catalog:linear
car_tt_catalog:linear;train_tt_catalog:boxcox
car_tt_catalog:linear;train_tt_catalog:squared
car_tt_catalog:boxcox;train_tt_catalog:squared
car_tt_catalog:boxcox;train_tt_catalog:boxcox
car_tt_catalog:boxcox;train_tt_catalog:linear
car_tt_catalog:squared;train_tt_catalog:linear
car_tt_catalog:squared;train_tt_catalog:boxcox
car_tt_catalog:squared;train_tt_catalog:squared
```

Indeed, the combination of three configurations for one variable and three configurations for the other one gives nine specifications. However, this is not always the desired effect. It is actually often desirable that the same nonlinear transform is applied to both variables. In that case, we need to synchronize the two catalogs. It means that they must be controlled by the same controller. This is achieved by constructing the second catalog as follows:

```
  car_tt_catalog = Catalog.from_dict(
    catalog_name='car_tt_catalog',
    dict_of_expressions={
        'linear': linear_car_tt,
        'boxcox': boxcox_car_tt,
        'squared': squared_car_tt,
    },
    controlled_by=train_tt_catalog.controlled_by
)
```

The `controlled_by` argument allows to explicitly specify a controller for the catalog. In this case, we provide the controller of the `train_tt_catalog`. Note that it is required that synchronized catalogs have exactly the same set of labels to identify their entries. If we now report the specifications of the dummy expression defined above, we obtain only three specifications, where both variables are associated with the same transformation:

```
train_tt_catalog:linear
train_tt_catalog:squared
train_tt_catalog:boxcox
```

Note that only the controller of the train travel time catalog is involved, as it is used also for the car travel time.

1.2 Alternative-specific coefficient

In discrete choice models, it is typical to test a specification where the coefficient of a variable is generic, that is, the same for all alternatives, or alternative-specific. For example, we are considering a catalog containing specifications where the cost coefficient and the time coefficient should be both generic, or both alternative-specific. In order to build such a catalog, we need the function `generic_alt_specific_catalogs` that can be imported as follows:

```
from biogeme.catalog import generic_alt_specific_catalogs
```

The following syntax is used:

```
(B_TIME_catalog_dict, B_COST_catalog_dict) =
    generic_alt_specific_catalogs(
     generic_name='coefficients',
     beta_parameters=[B_TIME, B_COST],
     alternatives=('TRAIN', 'CAR')
)
```

The function takes three[a] arguments:

1. a generic name that identifies the catalogs,
2. a list of parameters, defined with `Beta`,
3. a tuple containing the names identifying the alternatives.

It returns a tuple of dictionaries where the keys are the name of the alternatives, and the values are the corresponding catalogs. They are used as follows:

[a]As discussed later, it actually takes five arguments, but two of them have default values.

```
V_TRAIN = (
    B_TIME_catalog_dict['TRAIN'] * TRAIN_TT +
    B_COST_catalog_dict['TRAIN'] * TRAIN_COST
)
V_CAR = (
    B_TIME_catalog_dict['CAR'] * CAR_TT +
    B_COST_catalog_dict['CAR'] * CAR_COST
)
```

In order to illustrate the catalogs, we build again a dummy expression:

```
dummy_expression = V_TRAIN + V_CAR
```

There are two possible configurations for this expression, one where both coefficients are alternative-specific, and one where both are generic.

```
coefficients_gen_altspec:generic
coefficients_gen_altspec:altspec
```

If it is not desirable to have both coefficients synchronized, two different calls to the function must be performed:

```
(B_TIME_catalog_dict, ) = generic_alt_specific_catalogs(
    generic_name='time_coefficient',
    beta_parameters=[B_TIME],
    alternatives=('TRAIN', 'CAR')
)

(B_COST_catalog_dict, ) = generic_alt_specific_catalogs(
    generic_name='cost_coefficient',
    beta_parameters=[B_COST],
    alternatives=('TRAIN', 'CAR')
)
```

Note that the function returns a tuple. And if the tuple contains only one entry (as in this example), a comma must be explicitly mentioned in order to obtain this single entry. An equivalent syntax would be

```
B_TIME_catalog_dict_tuple = generic_alt_specific_catalogs(
    generic_name='time_coefficient',
    beta_parameters=[B_TIME],
    alternatives=('TRAIN', 'CAR')
)
B_TIME_catalog_dict = B_TIME_catalog_dict_tuple[0]
```

As the two specifications are now independent, iterating on the dummy expression provides four specifications:

```
cost_coefficient_gen_altspec:generic;time_coefficient_gen_altspec:generic
cost_coefficient_gen_altspec:generic;time_coefficient_gen_altspec:altspec
cost_coefficient_gen_altspec:altspec;time_coefficient_gen_altspec:generic
cost_coefficient_gen_altspec:altspec;time_coefficient_gen_altspec:altspec
```

1.3 Segmentations

In order to capture potential taste heterogeneity, specifications where a coefficient takes different values for different segments of the population can be investigated. The population is segmented using discrete socio-economic characteristics. If such a discrete variable takes L values, they correspond to L segments in the population. But several such variables can be combined to define a segmentation. If K socio-economic characteristics are considered, each of them with L_k discrete values, a total of $\prod_{k=1}^{K} L_k$ segments can potentially be defined, and a different coefficient associated with each of them. However, the number of segments defined in this way grows exponentially with K. It is statistically impossible to estimate a different coefficient for each segment when K is high. Therefore, we consider a simplified segmentation method that proceeds as follows:

- Define a reference coefficient β_{ref}.
- For each socio-economic characteristic x_k, select one value that corresponds to the reference. Without loss of generality, assume that it is the first one.
- Introduce a parameter β_k^ℓ, for each other value $\ell = 2, \ldots, L_k$.
- The value of the coefficient as a function of the socio-economic characteristics is defined as

$$\beta(x_1, \ldots, x_K) = \beta_{\text{ref}} + \sum_{k=1}^{K} \sum_{\ell=2}^{L_k} \beta_k^\ell \, \mathbb{1}[x_k = \ell],$$

  where $\mathbb{1}[x_k = \ell]$ is 1 if the condition within the brackets is true, and 0 otherwise.

The number of parameters is therefore $1 - K + \sum_{k=1}^{K} L_k$, which grows linearly with K.

Let's take an example with $K = 2$, where the first socio-economic characteristic segments the population between individuals who are commuters from those who are not, and the second segments the population into individuals without luggage, those carrying one piece of luggage, and those carrying more than one piece of luggage. Therefore, $L_1 = 2$ and $L_2 = 3$. This segmentation is associated with $1 - 2 + 2 + 3 = 4$ coefficients:

- β_{ref},
- $\beta_1^{\text{commuters}}$,

- $\beta_2^{\text{one\_luggage}}$,
- $\beta_2^{\text{several\_luggages}}$,

where the values "non commuters" and "no luggage" are used as reference for each variable, respectively. Now, note that the number of segments is $2 \cdot 3 = 6$. The value of the coefficient associated with each of them can be reconstructed from the above coefficients as follows:

Commuter	Luggages	Coefficient
yes	0	$\beta_{\text{ref}} + \beta_1^{\text{commuters}}$
yes	1	$\beta_{\text{ref}} + \beta_1^{\text{commuters}} + \beta_2^{\text{one\_luggage}}$
yes	> 1	$\beta_{\text{ref}} + \beta_1^{\text{commuters}} + \beta_2^{\text{several\_luggages}}$
no	0	β_{ref}
no	1	$\beta_{\text{ref}} + \beta_2^{\text{one\_luggage}}$
no	> 1	$\beta_{\text{ref}} + \beta_2^{\text{several\_luggages}}$

This simplified procedure makes the implicit assumption that the combined effects of two socio-economic characteristics is the sum of two specific effects. This is the price to pay to deal with the curse of dimensionality.

In this context, we would like to construct a catalog that contains the following specifications:

- no segmentation, that is, the same coefficient for the whole population,
- a segmentation with the first variable only, that is $1 - 1 + 2 = 2$ coefficients: β_{ref} and $\beta_1^{\text{commuters}}$,
- a segmentation with the second variable only, that is $1 - 1 + 3 = 3$ coefficients: β_{ref}, $\beta_2^{\text{one\_luggage}}$ and $\beta_2^{\text{several\_luggages}}$,
- a segmentation with both variables, involving 4 coefficients as described above.

And we would like to apply these segmentations to two alternative-specific constants, that must be segmented in the same way. To do that with Biogeme, we first need to define the segmentations, using the following syntax:

```
segmentation_purpose = database.generate_segmentation(
    variable='COMMUTERS',
    mapping={
        0: 'non_commuters',
        1: 'commuters'
    },
    reference='non_commuters'
```

```
)
segmentation_luggage = database.generate_segmentation(
    variable='LUGGAGE',
    mapping={
        0: 'no_lugg',
        1: 'one_lugg',
        3: 'several_lugg'
    },
    reference='no_lugg'
)
```

where the function `generate_segmentation` takes the following two arguments:

- the name of the discrete socio-economic characteristic in the database,
- a dictionary mapping the values of the variables in the database, and a name identifying what they mean,
- the name of the reference level.

Note that the name of the reference level can be omitted. One of the levels will then be arbitrarily chosen as the reference. We can now create the catalogs themselves:

```
ASC_TRAIN_catalog, ASC_CAR_catalog = segmentation_catalogs(
    generic_name='ASC',
    beta_parameters=[ASC_TRAIN, ASC_CAR],
    potential_segmentations=(
        segmentation_purpose,
        segmentation_luggage,
    ),
    maximum_number=2,
)
```

where the function `segmentation_catalogs` can be imported using the following statement

```
from biogeme.catalog import segmentation_catalogs
```

It takes four arguments:

1. a generic name that applies to all specifications,
2. a list of parameters to be segmented,
3. a list of potential segmentations,
4. the maximum number of segmentations that can be activated at the same time.

If we report the configurations of the dummy expression defined as the sum of the two catalogs, we obtain the following four configurations:

```
ASC:no_seg
ASC:LUGGAGE
ASC:COMMUTERS
ASC:COMMUTERS-LUGGAGE
```

If we call the same function with the parameter `maximum_number` set to 1, we obtain

```
ASC:no_seg
ASC:LUGGAGE
ASC:COMMUTERS
```

as the interaction with both variables is not allowed anymore.

1.4 Alternative-specific and segmented coefficients

It is also possible to segment alternative-specific coefficients, and generate catalogs that provide specifications with or without segmentation, and with generic or alternative-specific coefficients. This is done using the following syntax:

```
(B_TIME_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_TIME',
    beta_parameters=[B_TIME],
    alternatives=['TRAIN', 'CAR'],
    potential_segmentations=(
        segmentation_purpose,
        segmentation_luggage,
    ),
    maximum_number=1,
)
```

where the function `generic_alt_specific_catalogs` is the same as in Section 1.2, and can be imported as follows:

```
from biogeme.catalog import generic_alt_specific_catalogs
```

The function takes five arguments:

1. a generic name that identifies the catalogs,
2. a list of parameters, defined with `Beta`,
3. a tuple containing the names identifying the alternatives,
4. a list of potential segmentations (set to `None` by default),

5. the maximum number of segmentations that can be activated at the same time (set to 5 by default).

This function creates a dictionary with two catalogs `B_TIME_catalog['TRAIN']` and `B_TIME_catalog['CAR']`, synchronized, and therefore controlled by the same controller. There are six possible configurations:

```
B_TIME:no_seg;B_TIME_gen_altspec:generic
B_TIME:no_seg;B_TIME_gen_altspec:altspec
B_TIME:LUGGAGE;B_TIME_gen_altspec:generic
B_TIME:LUGGAGE;B_TIME_gen_altspec:altspec
B_TIME:COMMUTERS;B_TIME_gen_altspec:generic
B_TIME:COMMUTERS;B_TIME_gen_altspec:altspec
```

If we allow to segment the population with two socio-economic characteristics instead of just one, we obtain a total of eight configurations, as the double segmentation can be considered with generic or alternative-specific coefficients:

```
B_TIME:no_seg;B_TIME_gen_altspec:generic
B_TIME:no_seg;B_TIME_gen_altspec:altspec
B_TIME:LUGGAGE;B_TIME_gen_altspec:generic
B_TIME:LUGGAGE;B_TIME_gen_altspec:altspec
B_TIME:COMMUTERS;B_TIME_gen_altspec:generic
B_TIME:COMMUTERS;B_TIME_gen_altspec:altspec
B_TIME:COMMUTERS-LUGGAGE;B_TIME_gen_altspec:generic
B_TIME:COMMUTERS-LUGGAGE;B_TIME_gen_altspec:altspec
```

2 Comparing models

The use of catalogs generates a great deal of potential specifications. And we would like to focus of the best ones. One possibility would be to focus on one criterion, such as the Akaike Information Criterion (AIC), and decide that the best model is the one with the lowest AIC. While it is a valid idea, the outcome of the estimation will be exactly one model. And if, for some reasons, that model happens not to be acceptable, no other model will be proposed to the analyst. Instead, we would like to combine several indicators to identify good models. In particular, we would like to keep models that fit the data well (that is, associated with a high log likelihood), and models that are parsimonious (that is, with a low number of parameters). If we consider those two indicators simultaneously, we need to use the concept of dominance and Pareto optimality (formally defined in Appendix 1). Consider a model M_1 with K_1 parameters and final log likelihood $\mathcal{L}_1$, and M_2 with K_2 parameters and final log likelihood $\mathcal{L}_2$. We say that M_1 dominates M_2 if it is no worse than M_2 in any objective, and strictly better in at least one objective, that is:

$$\mathcal{L}_1 \geq \mathcal{L}_2 \text{ and } K_1 < K_2,$$

or

$$\mathcal{L}_1 > \mathcal{L}_2 \text{ and } K_1 \leq K_2.$$

In this context, we will keep only models that are not dominated. Such models are said to be Pareto optimal.

3 Estimating parameters using catalogs

We illustrate the concept of catalogs by estimating several specifications. We build on the examples from Section 1 on page 2.

3.1 Various choice models

We consider first a catalog that includes a logit and two nested logit models, each with a different nest definition. The catalog is constructed as described above:

```
model_catalog = Catalog.from_dict(
    catalog_name='model_catalog',
    dict_of_expressions={
        'logit': logprob_logit,
        'nested existing': logprob_nested_existing,
        'nested public': logprob_nested_public,
    },
)
```

and is provided to the Biogeme object:

```
the_biogeme = bio.BIOGEME(database, model_catalog)
```

The various specifications can be estimated using the `estimate_catalog` function:

```
dict_of_results = the_biogeme.estimate_catalog()
```

The complete code is available in Appendix 6. The output of estimation is a dictionary, where each key is the name of a model, and each value is an object containing the estimation results. In this document, we process this dictionary using the code presented in Appendix 5. The output of the script contains two parts. The first part contains the complete set of results (see Figure 1). Each column is associated with a model name, each name being associated with a specification below:

```
Model_000000    model_catalog:nested public
Model_000001    model_catalog:nested existing
Model_000002    model_catalog:logit
```

```
A total of 3 models have been estimated
== Estimation results ==
                                  Model_000000     Model_000001     Model_000002
Number of estimated parameters               5                5                4
Sample size                               6768             6768             6768
Final log likelihood              -5331.252007     -5236.900014     -5331.252007
Akaike Information Criterion      10672.504014     10483.800028     10670.504014
Bayesian Information Criterion    10706.603818     10517.899832     10697.783857
ASC_CAR (t-test)               -0.155  (-2.03)  -0.167  (-3.07)  -0.155  (-2.66)
ASC_TRAIN (t-test)             -0.701  (-5.22)  -0.512  (-6.47)  -0.701  (-8.49)
B_COST (t-test)                 -1.08  (-14.4)  -0.857  (-14.3)   -1.08  (-15.9)
B_TIME (t-test)                 -1.28  (-10.5)  -0.899  (-8.39)   -1.28  (-12.3)
MU_public (t-test)                  1   (8.78)
MU_existing (t-test)                              2.05   (12.5)
Model_000000    model_catalog:nested public
Model_000001    model_catalog:nested existing
Model_000002    model_catalog:logit
```

Figure 1: Different choice models: complete estimation report

It can be seen that the models `model_catalog:nested public` and `model_catalog:logit` achieve the same final log likelihood. The nest parameter of the nested logit model is actually 1. Therefore, model `model_catalog:nested public` is dominated by model `model_catalog:logit`, and should be rejected. This is how the second part of the output is generated, keeping only non dominated models, as reported in Figure 2 on the next page. Note that the logit model is better in terms of parsimony, and the nested logit model is better in terms of fit.

```
                                    Model_000000     Model_000001
Number of estimated parameters                 5                4
Sample size                                 6768             6768
Final log likelihood                -5236.900014     -5331.252007
Akaike Information Criterion        10483.800028     10670.504014
Bayesian Information Criterion      10517.899832     10697.783857
ASC_CAR (t-test)                 -0.167  (-3.07)  -0.155  (-2.66)
ASC_TRAIN (t-test)               -0.512  (-6.47)  -0.701  (-8.49)
B_COST (t-test)                  -0.857  (-14.3)   -1.08  (-15.9)
B_TIME (t-test)                  -0.899  (-8.39)   -1.28  (-12.3)
MU_existing (t-test)                2.05  (12.5)
Model_000000    model_catalog:nested existing
Model_000001    model_catalog:logit
```

Figure 2: Different choice models: Pareto optimal models

3.2 Nonlinear specifications

We consider a catalog that includes various specifications for the travel time variables:

- a linear specification,
- a Box-Cox transform,
- a power series of degree 3.

If x_t is the travel time variable, the catalog contains the following specifications:

$$x_t, \ \frac{x_t^\lambda - 1}{\lambda}, \text{ and } x_t + \beta_{\text{square}} x_t^2 + \beta_{\text{cube}} x_t^3.$$

It can be seen that some of these specifications involve additional parameters, some not. We use synchronized catalogs, so that the travel time variable is involved in the same way in all alternatives. The full specification is available in Appendix 7. The results associated with each of the three specifications are reported in Figure 3 on the following page. It is interesting to note that none of these model is dominated by another one.

```
A total of 3 models have been estimated
== Estimation results ==
                                   Model_000000         Model_000001      Model_000002
Number of estimated parameters                4                    5                 6
Sample size                                6768                 6768              6768
Final log likelihood                -5331.252007         -5292.095411      -5236.262942
Akaike Information Criterion        10670.504014         10594.190822      10484.525883
Bayesian Information Criterion      10697.783857         10628.290626      10525.445649
ASC_CAR (t-test)                  -0.155  (-2.66)  -0.00462  (-0.0963)   0.0434  (0.965)
ASC_TRAIN (t-test)                -0.701  (-8.49)      -0.485  (-7.53)    -0.409  (-6.8)
B_COST (t-test)                    -1.08  (-15.9)       -1.08  (-15.9)      -1.11  (-16)
B_TIME (t-test)                    -1.28  (-12.3)       -1.67  (-21.9)    -2.32  (-22.6)
lambda_travel_time (t-test)                              0.51  (6.6)
cube_tt_coef (t-test)                                                   0.000193  (7.38)
square_tt_coef (t-test)                                                   -0.105  (-21.2)
Model_000000    train_tt_catalog:linear
Model_000001    train_tt_catalog:boxcox
Model_000002    train_tt_catalog:power
```

Figure 3: Nonlinear specifications: complete estimation report

3.3 Alternative-specific coefficients

We consider a catalog that considers both generic and alternative-specific specifications for both the cost coefficient and the travel time coefficient. The full specification is available in Appendix 8. The results associated with each of the four specifications are reported in Figure 4 on the next page. Note that the model where the cost coefficient is generic and the time coefficient is alternative-specific is dominated by the model where the cost coefficient is alternative-specific and the time coefficient is generic. Indeed, both models involve 6 parameters, that the latter has a better fit.

```
A total of 4 models have been estimated
== Estimation results ==
                                       Model_000000          Model_000001       Model_000002       Model_000003
Number of estimated parameters                    6                     8                  6                  4
Sample size                                    6768                  6768               6768               6768
Final log likelihood                   -5312.894223          -5075.704346       -5083.499937       -5331.252007
Akaike Information Criterion           10637.788446          10167.408692       10178.999875       10670.504014
Bayesian Information Criterion         10678.708211          10221.968379        10219.91964       10697.783857
ASC_CAR (t-test)                   -0.271   (-2.29)     -0.367   (-3.32)   -0.427   (-5.55)   -0.155   (-2.66)
ASC_TRAIN (t-test)                 -0.202   (-1.82)   -0.0754   (-0.712)    0.189    (2.06)   -0.701   (-8.49)
B_COST (t-test)                       -1.07   (-16)                                            -1.08   (-15.9)
B_TIME_CAR (t-test)                 -1.12   (-10.3)      -1.29   (-7.92)
B_TIME_SM (t-test)                  -1.17   (-6.42)      -1.11   (-6.25)
B_TIME_TRAIN (t-test)               -1.57   (-14.4)     -0.889   (-7.51)
B_COST_CAR (t-test)                                     -0.786   (-5.27)   -0.939   (-8.1)
B_COST_SM (t-test)                                       -1.12   (-14.2)    -1.09   (-15.5)
B_COST_TRAIN (t-test)                                     -3.08   (-16)     -2.93   (-17.4)
B_TIME (t-test)                                                             -1.12   (-9.3)     -1.28   (-12.3)
Model_000000     B_COST_gen_altspec:generic;B_TIME_gen_altspec:altspec
Model_000001     B_COST_gen_altspec:altspec;B_TIME_gen_altspec:altspec
Model_000002     B_COST_gen_altspec:altspec;B_TIME_gen_altspec:generic
Model_000003     B_COST_gen_altspec:generic;B_TIME_gen_altspec:generic
```

Figure 4: Alternative-specific coefficients: complete estimation report

3.4 Segmentations

We consider a catalog that considers potential segmentations of the parameters. The alternative-specific constants are potentially interacted with the variables GA (identifying if the traveler owns a yearly subscription, with 2 levels) and LUGGAGES (identifying if the traveler is carrying luggages, with 3 levels), or both. The travel time coefficient is potentially interacted with the variables FIRST (identifying if the traveler is traveling first class, with 2 levels) or PURPOSE (identifying if the traveler is a commuter or not, with 2 levels). Maximum one such interaction is allowed.

Therefore, we have 4 specifications for the constants:

- not segmented,
- segmented by GA (yearly subscription to public transport),
- segmented by luggage,
- segmented both by GA and luggage,

and 3 specifications for the time coefficients:

- not segmented,
- segmented with first class,
- segmented with trip purpose,

so that we obtain a total of 12 specifications.

The full specification is available in Appendix 9. Among the 12 estimated models, 5 are Pareto optimal. The estimation results are reported in Figure 5 on the following page.

	Model_000000		Model_000001		Model_000002		Model_000003		Model_000004	
Number of estimated parameters	6		7		11		5		4	
Sample size	6768		6768		6768		6768		6768	
Final log likelihood	−5050.677696		−4976.118641		−4952.546476		−5234.708233		−5331.252007	
Akaike Information Criterion	10113.355391		9966.237282		9927.092951		10479.416466		10670.504014	
Bayesian Information Criterion	10154.275157		10013.977009		10002.112521		10513.51627		10697.783857	
ASC_CAR (t−test)	−0.249	(−3.97)	−0.281	(−4.53)	−0.298	(−4.12)	−0.187	(−3.23)	−0.155	(−2.66)
ASC_CAR_GA (t−test)	−0.301	(−1.56)	−0.231	(−1.19)	−0.206	(−1.05)				
ASC_TRAIN (t−test)	−1.28	(−14)	−1.37	(−14.7)	−1.79	(−15.4)	−0.814	(−9.45)	−0.701	(−8.49)
ASC_TRAIN_GA (t−test)	1.97	(22.3)	1.91	(21.5)	1.75	(19.1)				
B_COST (t−test)	−1.1	(−14.8)	−1.26	(−15.3)	−1.25	(−15.3)	−1.23	(−16.6)	−1.08	(−15.9)
B_TIME (t−test)	−1.18	(−11.3)	−0.621	(−4.46)	−0.622	(−4.42)	−0.647	(−4.69)	−1.28	(−12.3)
B_TIME_1st_class (t−test)			−0.914	(−8.6)	−0.891	(−8.26)	−1.02	(−9.87)		
ASC_CAR_one_lugg (t−test)					0.0324	(0.486)				
ASC_CAR_several_lugg (t−test)					−0.437	(−1.82)				
ASC_TRAIN_one_lugg (t−test)					0.635	(6.4)				
ASC_TRAIN_several_lugg (t−test)					0.431	(2)				

Model_000000	ASC:GA;B_TIME:no_seg
Model_000001	ASC:GA;B_TIME:FIRST
Model_000002	ASC:GA_LUGGAGE;B_TIME:FIRST
Model_000003	ASC:no_seg;B_TIME:FIRST
Model_000004	ASC:no_seg;B_TIME:no_seg

Figure 5: Segmentation: Pareto optimal models

3.5 Segmentations and alternative-specific coefficients

We consider a catalog that considers potential segmentations of the parameters as well as alternative-specific coefficients. We consider 4 specifications for the constants:

- not segmented,
- segmented by GA (yearly subscription to public transport),
- segmented by luggage,
- segmented both by GA and luggage.

We consider 6 specifications for the time coefficients:

- generic and not segmented,
- generic and segmented with first class,
- generic and segmented with trip purpose,
- alternative-specific and not segmented,
- alternative-specific and segmented with first class,
- alternative-specific and segmented with trip purpose.

Finally, We consider 2 specifications for the cost coefficients:

- generic,
- alternative-specific.

In total, we obtain 48 specifications. The full specification is available in Appendix 10. Among the 48 estimated models, 8 are Pareto optimal. The estimation results are reported in Figure 6 on the next page and Figure 7 on page 26.

	Model_000000	Model_000001	Model_000002	Model_000003
Number of estimated parameters	7	17	6	9
Sample size	6768	6768	6768	6768
Final log likelihood	−4976.118641	−4865.971435	−5050.677696	−4945.30006
Akaike Information Criterion	9966.237282	9765.94287	10113.355391	9908.60012
Bayesian Information Criterion	10013.977009	9881.882206	10154.275157	9969.979768
ASC_CAR (t−test)	−0.281 (−4.53)	−0.446 (−3.68)	−0.249 (−3.97)	−0.662 (−7.79)
ASC_CAR_GA (t−test)	−0.231 (−1.19)	−0.145 (−0.739)	−0.301 (−1.56)	−0.0761 (−0.389)
ASC_TRAIN (t−test)	−1.37 (−14.7)	−1.07 (−6.72)	−1.28 (−14)	−0.938 (−6.76)
ASC_TRAIN_GA (t−test)	1.91 (21.5)	1.26 (8.67)	1.97 (22.3)	1.52 (11.1)
B_COST (t−test)	−1.26 (−15.3)		−1.1 (−14.8)	
B_TIME (t−test)	−0.621 (−4.46)		−1.18 (−11.3)	−0.69 (−4.56)
B_TIME_1st_class (t−test)	−0.914 (−8.6)			−0.925 (−8.62)
ASC_CAR_one_lugg (t−test)		0.0264 (0.394)		
ASC_CAR_several_lugg (t−test)		−0.299 (−1.23)		
ASC_TRAIN_one_lugg (t−test)		0.674 (6.7)		
ASC_TRAIN_several_lugg (t−test)		0.495 (2.3)		
B_COST_CAR (t−test)		−0.836 (−5.28)		−0.848 (−7.25)
B_COST_SM (t−test)		−1.15 (−14)		−1.3 (−16.1)
B_COST_TRAIN (t−test)		−2.03 (−9.61)		−1.83 (−10.3)
B_TIME_CAR (t−test)		−1.55 (−11.3)		
B_TIME_CAR_commuters (t−test)		0.682 (3.48)		
B_TIME_SM (t−test)		−1.73 (−15.3)		
B_TIME_SM_commuters (t−test)		1.6 (8.06)		
B_TIME_TRAIN (t−test)		−1.34 (−12.7)		
B_TIME_TRAIN_commuters (t−test)		0.116 (0.848)		

Model_000000	ASC:GA; B_COST_gen_altspec: generic; B_TIME:FIRST; B_TIME_gen_altspec: generic
Model_000001	ASC:GA—LUGGAGE; B_COST_gen_altspec: altspec; B_TIME:COMMUTERS; B_TIME_gen_altspec: altspec
Model_000002	ASC:GA; B_COST_gen_altspec: generic; B_TIME: no_seg; B_TIME_gen_altspec: generic
Model_000003	ASC:GA; B_COST_gen_altspec: altspec; B_TIME:FIRST; B_TIME_gen_altspec: generic

Figure 6: Segmentation and alternative-specific coefficients: Pareto optimal models (part 1)

	Model_000004	Model_000005	Model_000006	Model_000007
Number of estimated parameters	4	5	11	13
Sample size	6768	6768	6768	6768
Final log likelihood	−5331.252007	−5234.708233	−4928.268572	−4890.815071
Akaike Information Criterion	10670.504014	10479.416466	9878.537145	9807.630143
Bayesian Information Criterion	10697.783857	10513.51627	9953.556715	9896.289635
ASC_CAR (t−test)	−0.155 (−2.66)	−0.187 (−3.23)	−0.383 (−2.95)	−0.434 (−3.72)
ASC_CAR_GA (t−test)	−0.		−0.217 (−1.14)	−0.173 (−0.891)
ASC_TRAIN (t−test)	−1701 (−8.49)	−0.814 (−9.45)	−0.965 (−7.29)	−0.593 (−4.28)
ASC_TRAIN_GA (t−test)			2.05 (21.8)	1.38 (9.3)
B_COST (t−test)	−1.08 (−15.9)	−1.23 (−16.6)	−1.13 (−15)	
B_TIME (t−test)	−0..28 (−12.3)	−0.647 (−4.69)		
B_TIME_1st_class (t−test)	−0	−1.02 (−9.87)		
ASC_CAR_one_lugg (t−test)				
ASC_CAR_several_lugg (t−test)				
ASC_TRAIN_one_lugg (t−test)				
ASC_TRAIN_several_lugg (t−test)				
B_COST_CAR (t−test)				−0.845 (−5.37)
B_COST_SM (t−test)				−1.15 (−14.1)
B_COST_TRAIN (t−test)				−2.09 (−9.76)
B_TIME_CAR (t−test)			−1.4 (−16.8)	−1.55 (−11.4)
B_TIME_CAR_commuters (t−test)			0.699 (3.61)	0.692 (3.54)
B_TIME_SM (t−test)			−1.8 (−16.6)	−1.74 (−15.4)
B_TIME_SM_commuters (t−test)			1.66 (8.62)	1.62 (8.15)
B_TIME_TRAIN (t−test)			−1.61 (−17.3)	−1.35 (−12.8)
B_TIME_TRAIN_commuters (t−test)			0.178 (1.3)	0.13 (0.956)

Model_000004	ASC: no_seg; B_COST_gen_altspec: generic; B_TIME: no_seg; B_TIME_gen_altspec: generic
Model_000005	ASC: no_seg; B_COST_gen_altspec: generic; B_TIME: FIRST; B_TIME_gen_altspec: generic
Model_000006	ASC:GA; B_COST_gen_altspec: generic; B_TIME:COMMUTERS; B_TIME_gen_altspec: altspec
Model_000007	ASC:GA; B_COST_gen_altspec: altspec; B_TIME:COMMUTERS; B_TIME_gen_altspec: altspec

Figure 7: Segmentation and alternative-specific coefficients: Pareto optimal models (part 2)

3.6 Combining several specifications

We consider now a combination of the various specifications considered so far:

- 3 models:
  - logit,
  - nested logit with two nests: public and private transportation,
  - nested logit with two nests existing and future modes,
- 3 functional forms for the travel time variables:
  - linear specification,
  - Box-Cox transform,
  - power series,
- 2 specifications for the cost coefficients:
  - generic,
  - alternative-specific,
- 2 specification for the travel time coefficients:
  - generic,
  - alternative-specific,
- 4 segmentations for the constants:
  - not segmented,
  - segmented by GA (yearly subscription to public transport),
  - segmented by luggage,
  - segmented both by GA and luggage,
- 3 segmentations for the time coefficients:
  - not segmented,
  - segmented with first class,
  - segmented with trip purpose.

This leads to a total of 432 specifications. The script with the specification is available in Appendix 11. If it is attempted to estimate all specifications of this catalog, the following exception will be raised:

```
There are too many [432] different specifications for the log
    likelihood function. This is above the maximum number: 100.
    Simplify the specification, change the value of the
    parameter maximum_number_catalog_expressions, or consider
    using the AssistedSpecification object in the
    "biogeme.assisted" module.
```

4 Assisted specification

When the systematic estimation of all possible specifications is infeasible, it is possible to rely on the assisted specification algorithm, inspired by the work of Ortelli et al. (2021).

This is done by first creating the object, using the following syntax:

```
assisted_specification = AssistedSpecification(
    biogeme_object=the_biogeme,
    multi_objectives=loglikelihood_dimension,
    pareto_file_name=PARETO_FILE_NAME,
)
```

where the class `AssistedSpecification` must be imported as follows:

```
from biogeme.assisted import AssistedSpecification
```

Its constructor takes three arguments:

1. the biogeme object,
2. a function providing all the indicators used to exclude dominated models,
3. the name of a file that will collect all the models that have been estimated,
4. a function verifying the validity of the results (optional).

The biogeme object is constructed as before, from the database and the catalog:

```
the_biogeme = bio.BIOGEME(database, model_catalog)
```

The function must take the estimation results as argument, and return a list of indicators. The convention is that, the lower the value of the indicator, the better the model. Here is an example of such a function:

```
def loglikelihood_dimension(results):
    """Function returning the negative log likelihood and the
       number
```

```
    of parameters, designed for multi-objective optimization

    :param results: estimation results
    :type results: biogeme.results.bioResults
    """
    return [-results.data.logLike, results.data.nparam]
```

The two indicators in this case are

- the opposite of the final log likelihood (opposite, because of the above mentioned convention),
- the number of estimated parameters.

Another example involving three indicators is as follows:

```
def AIC_BIC_dimension(results):
    """Function returning the AIC, BIC and the number
    of parameters, designed for multi-objective optimization

    :param results: estimation results
    :type results: biogeme.results.bioResults
    """
    return [results.data.akaike, results.data.bayesian,
       results.data.nparam]
```

The three indicators are the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC) and the number of estimated parameters. Those two examples can actually be directly imported from biogeme:

```
from biogeme.multiobjectives import loglikelihood_dimension,
    AIC_BIC_dimension
```

The "pareto file" is the memory of the process. It stores all models that have been estimated by the algorithm, together with the relevant indicators. It is organized in three sections:

1. The `[Pareto]` section contains all models that are not dominated.
2. The `[Considered]` section contains all models that have been estimated.
3. The `[Removed]` section contains all models that have been Pareto optimal at some point during the algorithm, but that have been rejected by a dominating model.
4. The `[Invalid]` section contains all models that have been identified as invalid.

If the file exists when the algorithm is started, its content is used to initialize the algorithm. This allows to interrupt the algorithm and to relaunch it without losing what has been found so far.

Like the function calculating the indicators, the function verifying the validity of the results takes also estimation results as argument, as returns a tuple with two values:

1. a boolean that is True if the results are valid, and False otherwise,
2. a string explaining why the results are invalid, or None if they are valid.

Here is an example of such a function, where the results are reported invalid if any coefficient of time or cost is non negative:

```
def validity(results):
    """Function verifying that the estimation results are valid.

    The results are not valid if any of the time or cost
        coefficient is non negative.
    """
    for beta in results.data.betas:
        if 'TIME' in beta.name and beta.value >= 0:
            return False, f'{beta.name} = {beta.value}'
        if 'COST' in beta.name and beta.value >= 0:
            return False, f'{beta.name} = {beta.value}'
    return True, None
```

The algorithm is executed using the following statement:

```
non_dominated_models = assisted_specification.run()
```

Similarly to the `estimate_catalog` function, it returns a dictionary with all Pareto optimal models. The code is reported in Appendix 12.

Before looking at the results in the next section, we note that the concept of "valid" models can be dealt with in several ways. In particular, the sign of a coefficient can be constrained using the bounds appearing in the definition of the Beta expression. For instance, if the time and cost coefficients are constrained to be non positive, all models will be "valid" by design, and the above function will always return "True". This may be a good alternative if there is a high rate of rejected invalid models, that may decrease the capability of the algorithm to explore the space of possible specifications.

5 Using the Pareto file

As mentioned above, the Pareto file contains the description of all models that have been estimated by the algorithm, as well as the requested indicators. In this Section, we describe some post-processing methods that allow to exploit it.

5.1 Selecting one model

Each model in the file is characterized by an ID. For instance:

```
SPEC_ID = (
    'ASC:GA-LUGGAGE;'
    'B_COST_gen_altspec:generic;'
    'B_TIME:FIRST;'
    'B_TIME_gen_altspec:generic;'
    'model_catalog:logit;'
    'train_tt_catalog:power'
)
```

corresponds to a model where

- the constants are segmented both by GA and luggages,
- the cost coefficient is generic,
- the time coefficient is segmented by first class,
- the time coefficient is generic,
- the model is logit,
- the travel time variable is transformed using a power series.

The Biogeme object corresponding to this specification can be obtained using the following constructor:

```
the_biogeme = bio.BIOGEME.from_configuration(
    config_id=SPEC_ID,
    expression=model_catalog,
    database=database,
)
```

It can be used, either for re-estimation, or for applications.

5.2 Post processing

The post processing object accepts as input the Biogeme object as well as the Pareto file:

```
post_processing = ParetoPostProcessing(
    biogeme_object=the_biogeme,
        pareto_file_name=PARETO_FILE_NAME
)
```

where the class itself is imported as follows:

```
from biogeme.assisted import ParetoPostProcessing
```

The main purpose of this object is to re-estimate all models that are Pareto optimal. This can be done using the statement:

```
post_processing.reestimate(recycle=True)
```

The option "recycle=True" does not re-estimate a model if the pickle file is already present. Instead, it reads the results from this file. This may be useful when you interrupt the process. The next time you run it, it does not need to re-estimate the models that have already been processed. If you set it to False, the models are re-estimated, irrespectively of the presence of the pickle file. Note that no output file is overwritten. If an HTML file or a pickle file for a model already exist, a version number is inserted in the name of the file. For instance, if `my_model.html` already exists, the results will be saved in the file `my_model~00.html`.

Finally, it is possible to obtain an illustration of the amount of models that have been estimated by the algorithm and saved in the Pareto file. This can be done using the following statements:

```
_ = post_processing.plot(
    label_x='Negative log likelihood',
    label_y='Nbr of parameters',
)
    plt.show()
```

It generates a figure with two axes, corresponding to two objectives. Each model is represented by a point with coordinates calculated using the corresponding objectives. The shape of the point represents the status of the model:

- A circle represents a Pareto optimal model.
- A cross represents a model that has been Pareto optimal at some point during the course of the algorithm, and later dominated by another model.
- A star represents a model that has been deemed invalid.
- A small dot represents all other models that have been considered.

An example of this illustration is available in Figure 8 on the next page.

Note that, when more than two objectives have been used by the algorithm, the first two are used by default for the plot. But other objectives can be selected using the parameters `objective_x` and `objective_y`. This can

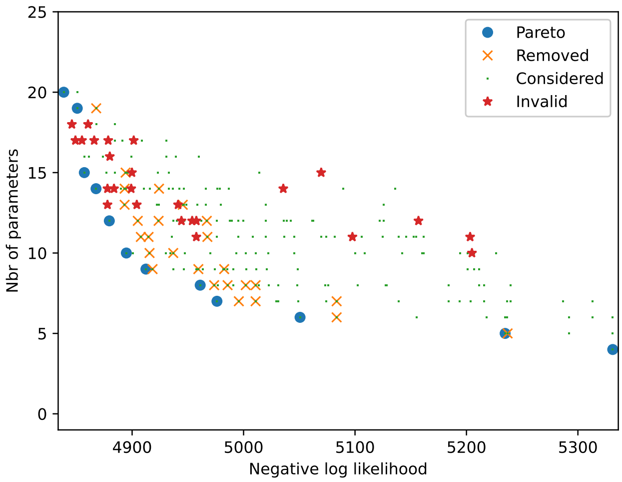

Figure 8: Models in the Pareto file

also be used to swap the position of the axes, as illustrated by the following statement, that generates the picture in Figure 9 on the following page:

```
_ = post_processing.plot(
    label_x='Nbr of parameters',
    label_y='Negative log likelihood',
    objective_x=1,
    objective_y=0,
)
```

6 Conclusion

This report describes several functionalities of Biogeme that happened to be useful to the authors in the context of model development. It is important to emphasize that they are not designed to replace the analyst and the modeler. Instead, they are designed to assist her, in order to facilitate the investigation of many possible specifications.

These features are experimental, and are likely to be improved in the future.

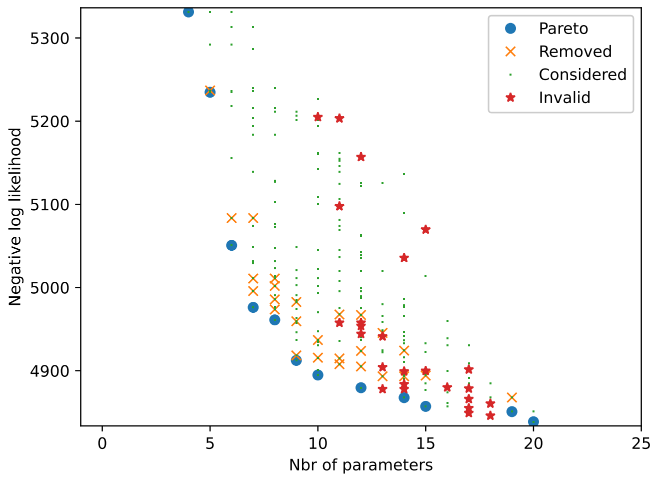

Figure 9: Models in the Pareto file (swapped axes)

References

Bierlaire, M. and Ortelli, N. (2022). Assisted specification with biogeme, *Technical Report TRANSP-OR 220707*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Ortelli, N., Hillel, T., Pereira, F., de Lapparent, M. and Bierlaire, M. (2021). Assisted specification of discrete choice models, *Journal of Choice Modelling* **39**(100285).

Appendix

1 Dominance and Pareto optimality

We consider a vector $x \in \mathbb{R}^n$, which is associated with P indicators: $f_1(x)$, ..., $f_P(x)$. Each of these indicators is such that lower values are better than higher values. As there are multiple indicators, it is not necessarily straightforward to decide which between two vectors x and y is better, as one can be better for some indicators, and the other one for other indicators. In order to formalize this, we introduce the concept of dominance.

Consider two vectors $x, y \in \mathbb{R}^n$. We say that x is **dominating** y, and use the notation $x \prec y$, if

1. x is no worse in any objective

$$\forall i \in \{1, \dots, P\}, f_i(x) \leq f_i(y),$$

2. x is strictly better in at least one objective

$$\exists i \in \{1, \dots, P\}, f_i(x) < f_i(y).$$

The dominance relation has the following properties:

- Not reflexive: $x \not\prec x$.
- Not symmetric: $x \prec y \not\Rightarrow y \prec x$.
- Instead: $x \prec y \Rightarrow y \not\prec x$.
- Transitive: $x \prec y$ and $y \prec z \Rightarrow x \prec z$.
- Not complete: $\exists x, y$: $x \not\prec y$ and $y \not\prec x$.

Consider now a set $\mathcal{F} \subseteq \mathbb{R}^n$. The vector $x^* \in \mathcal{F}$ is **Pareto optimal** if it is not dominated by any solution in $\mathcal{F}$:

$$\nexists x \in \mathcal{F} \text{ such that } x \prec x^*.$$

Intuitively, x^* is Pareto optimal if no objective can be improved without degrading at least one of the others.

As the relation is not complete, there may be more than one Pareto optimal solution in a set. The Pareto optimal set is defined as

$$P^* = \{x^* \in \mathcal{F} | \nexists x \in \mathcal{F} : x \prec x^*\}.$$

2 Function printing the configurations of an expression

```
def print_all_configurations(expression: Expression) -> None:
    """Prints all configurations that an expression can take
    """
    expression.set_central_controller()
    total =
       expression.central_controller.number_of_configurations()
    print(f'Total: {total} configurations')
    for config_id in
       expression.central_controller.all_configurations_ids:
        print(config_id)
```

3 Illustrations of the catalogs

This is the code used to generate the examples in Section 1.

```
"""File simple_example.py

:author: Michel Bierlaire, EPFL
:date: Sun Aug  6 18:13:18 2023

Example of a catalog

"""
import sys
import numpy as np
import biogeme.biogeme as bio
from biogeme import models
from biogeme.expressions import Beta, Variable, Expression
from biogeme.models import boxcox
from biogeme.catalog import Catalog,
   generic_alt_specific_catalogs, segmentation_catalogs
from results_analysis import report
from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
```

```python
    CAR_CO_SCALED,
)

def print_all_configurations(expression: Expression) -> None:
    """Prints all configurations that an expression can take
    """
    expression.set_central_controller()
    total = expression.central_controller.number_of_configurations()
    print(f'Total: {total} configurations')
    for config_id in expression.central_controller.all_configurations_ids:
        print(config_id)

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


# Definition of the utility functions
V1 = ASC_TRAIN + B_TIME * TRAIN_TT_SCALED + B_COST * TRAIN_COST_SCALED
V2 = 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_logit = models.loglogit(V, av, CHOICE)

MU = Beta('MU', 1, 1, 10, 0)
existing = MU, [1, 3]
future = 1.0, [2]
nests = existing, future
logprob_nested = models.lognested(V, av, nests, CHOICE)

model_catalog = Catalog.from_dict(
    catalog_name='model_catalog',
    dict_of_expressions={
        'logit': logprob_logit,
        'nested': logprob_nested,
    },
```

```
)

print('*** Current status of the catalog ***')
print(model_catalog)
print('*** Use the controller to select a different
   configuration ***')
model_catalog.controlled_by.set_name('nested')
print('*** Current status of the catalog ***')
print(model_catalog)

print('*** Iterator ***')
for specification in model_catalog:
    print(specification)

print_all_configurations(model_catalog)

print('*** Nonlinear specifications *** ')
TRAIN_TT = Variable('TRAIN_TT')
TRAIN_COST = Variable('TRAIN_COST')
ell_travel_time = Beta('lambda_travel_time', 1, -10, 10, 0)
linear_train_tt = TRAIN_TT
boxcox_train_tt = boxcox(TRAIN_TT, ell_travel_time)
squared_train_tt = TRAIN_TT * TRAIN_TT
train_tt_catalog = Catalog.from_dict(
    catalog_name='train_tt_catalog',
    dict_of_expressions={
        'linear': linear_train_tt,
        'boxcox': boxcox_train_tt,
        'squared': squared_train_tt,
    },
)

ASC_TRAIN = Beta('ASC_TRAIN', 0, None, None, 0)
B_TIME = Beta('B_TIME', 0, None, 0, 0)
V_TRAIN = ASC_TRAIN + B_TIME * train_tt_catalog

print_all_configurations(V_TRAIN)

print('** Unsynchronized catalogs **')
CAR_TT = Variable('CAR_TT')
CAR_COST = Variable('CAR_COST')
linear_car_tt = CAR_TT
boxcox_car_tt = boxcox(CAR_TT, ell_travel_time)
squared_car_tt = CAR_TT * CAR_TT
car_tt_catalog = Catalog.from_dict(
    catalog_name='car_tt_catalog',
    dict_of_expressions={
        'linear': linear_car_tt,
        'boxcox': boxcox_car_tt,
```

```
        'squared': squared_car_tt,
    },
)

dummy_expression = train_tt_catalog + car_tt_catalog

print_all_configurations(dummy_expression)

print('** Synchronized catalogs **')
CAR_TT = Variable('CAR_TT')
CAR_COST = Variable('CAR_COST')
linear_car_tt = CAR_TT
boxcox_car_tt = boxcox(CAR_TT, ell_travel_time)
squared_car_tt = CAR_TT * CAR_TT
car_tt_catalog = Catalog.from_dict(
    catalog_name='car_tt_catalog',
    dict_of_expressions={
        'linear': linear_car_tt,
        'boxcox': boxcox_car_tt,
        'squared': squared_car_tt,
    },
    controlled_by=train_tt_catalog.controlled_by
)

dummy_expression = train_tt_catalog + car_tt_catalog

print_all_configurations(dummy_expression)


print('*** Alternative specific ***')

(B_TIME_catalog_dict, B_COST_catalog_dict) = generic_alt_specific_catalogs(
    generic_name='coefficients',
    beta_parameters=[B_TIME, B_COST],
    alternatives=('TRAIN', 'CAR')
)

V_TRAIN = (
    B_TIME_catalog_dict['TRAIN'] * TRAIN_TT +
    B_COST_catalog_dict['TRAIN'] * TRAIN_COST
)
V_CAR = (
    B_TIME_catalog_dict['CAR'] * CAR_TT +
    B_COST_catalog_dict['CAR'] * CAR_COST
)

dummy_expression = V_TRAIN + V_CAR

```

```
print_all_configurations(dummy_expression)

print('*** Alternative specific - not synchronized ***')

(B_TIME_catalog_dict, ) = generic_alt_specific_catalogs(
    generic_name='time_coefficient',
    beta_parameters=[B_TIME],
    alternatives=('TRAIN', 'CAR')
)

(B_COST_catalog_dict, ) = generic_alt_specific_catalogs(
    generic_name='cost_coefficient',
    beta_parameters=[B_COST],
    alternatives=('TRAIN', 'CAR')
)

V_TRAIN = (
    B_TIME_catalog_dict['TRAIN'] * TRAIN_TT +
    B_COST_catalog_dict['TRAIN'] * TRAIN_COST
)
V_CAR = (
    B_TIME_catalog_dict['CAR'] * CAR_TT +
    B_COST_catalog_dict['CAR'] * CAR_COST
)

dummy_expression = V_TRAIN + V_CAR

print_all_configurations(dummy_expression)

print('*** Segmentation ***')

# We consider two trip purposes: 'commuters' and anything else.
    We
# need to define a binary variable first
database.data['COMMUTERS'] = np.where(database.data['PURPOSE']
    == 1, 1, 0)
segmentation_purpose = database.generate_segmentation(
    variable='COMMUTERS',
    mapping={
        0: 'non_commuters',
        1: 'commuters'
    },
    reference='non_commuters'

)
segmentation_luggage = database.generate_segmentation(
    variable='LUGGAGE',
    mapping={
        0: 'no_lugg',
```

```
        1: 'one_lugg',
        3: 'several_lugg'
    },
    reference='no_lugg'
)


ASC_TRAIN_catalog, ASC_CAR_catalog = segmentation_catalogs(
    generic_name='ASC',
    beta_parameters=[ASC_TRAIN, ASC_CAR],
    potential_segmentations=(
        segmentation_purpose,
        segmentation_luggage,
    ),
    maximum_number=2,
)



dummy_expression = ASC_TRAIN_catalog + ASC_CAR_catalog

print_all_configurations(dummy_expression)

ASC_TRAIN_catalog, ASC_CAR_catalog = segmentation_catalogs(
    generic_name='ASC',
    beta_parameters=[ASC_TRAIN, ASC_CAR],
    potential_segmentations=(
        segmentation_purpose,
        segmentation_luggage,
    ),
    maximum_number=1,
)



dummy_expression = ASC_TRAIN_catalog + ASC_CAR_catalog

print_all_configurations(dummy_expression)

print('** Segmentation and alternative specific **')

(B_TIME_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_TIME',
    beta_parameters=[B_TIME],
    alternatives=['TRAIN', 'CAR'],
    potential_segmentations=(
        segmentation_purpose,
        segmentation_luggage,
    ),
```

```
    maximum_number=1,
)

print_all_configurations(B_TIME_catalog_dict['TRAIN'])

(B_TIME_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_TIME',
    beta_parameters=[B_TIME],
    alternatives=['TRAIN', 'CAR'],
    potential_segmentations=(
        segmentation_purpose,
        segmentation_luggage,
    ),
    maximum_number=2,
)

print_all_configurations(B_TIME_catalog_dict['TRAIN'])
```

4 Data

```
"""File swissmetro_data.py

:author: Michel Bierlaire, EPFL
:date: Mon Mar  6 15:17:03 2023

Data preparation for Swissmetro, and definition of the variables
"""

import pandas as pd
import biogeme.database as db
from biogeme.expressions import Variable

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

GROUP = Variable('GROUP')
SURVEY = Variable('SURVEY')
SP = Variable('SP')
ID = Variable('ID')
PURPOSE = Variable('PURPOSE')
FIRST = Variable('FIRST')
TICKET = Variable('TICKET')
WHO = Variable('WHO')
LUGGAGE = Variable('LUGGAGE')
AGE = Variable('AGE')
MALE = Variable('MALE')
```

```
INCOME = Variable('INCOME')
GA = Variable('GA')
ORIGIN = Variable('ORIGIN')
DEST = Variable('DEST')
TRAIN_AV = Variable('TRAIN_AV')
CAR_AV = Variable('CAR_AV')
SM_AV = Variable('SM_AV')
TRAIN_TT = Variable('TRAIN_TT')
TRAIN_CO = Variable('TRAIN_CO')
TRAIN_HE = Variable('TRAIN_HE')
SM_TT = Variable('SM_TT')
SM_CO = Variable('SM_CO')
SM_HE = Variable('SM_HE')
SM_SEATS = Variable('SM_SEATS')
CAR_TT = Variable('CAR_TT')
CAR_CO = Variable('CAR_CO')
CHOICE = Variable('CHOICE')

# Removing some observations can be done directly using pandas.
# remove = (((database.data.PURPOSE != 1) &
#            (database.data.PURPOSE != 3)) |
#           (database.data.CHOICE == 0))
# database.data.drop(database.data[remove].index,inplace=True)
# Here we use the "biogeme" way:
exclude = ((PURPOSE != 1) * (PURPOSE != 3) + (CHOICE == 0)) > 0
database.remove(exclude)


# Definition of new variables
SM_COST = database.DefineVariable('SM_COST', SM_CO * (GA == 0))
TRAIN_COST = database.DefineVariable('TRAIN_COST', TRAIN_CO * (GA == 0))
CAR_AV_SP = database.DefineVariable('CAR_AV_SP', CAR_AV * (SP != 0))
TRAIN_AV_SP = database.DefineVariable('TRAIN_AV_SP', TRAIN_AV * (SP != 0))
TRAIN_TT_SCALED = database.DefineVariable('TRAIN_TT_SCALED', TRAIN_TT / 100)
TRAIN_COST_SCALED = database.DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100)
SM_TT_SCALED = database.DefineVariable('SM_TT_SCALED', SM_TT / 100)
SM_COST_SCALED = database.DefineVariable('SM_COST_SCALED', SM_COST / 100)
CAR_TT_SCALED = database.DefineVariable('CAR_TT_SCALED', CAR_TT / 100)
CAR_CO_SCALED = database.DefineVariable('CAR_CO_SCALED', CAR_CO / 100)
```

5 Reporting

```
"""File results_analysis

:author: Michel Bierlaire, EPFL
:date: Thu Jul 13 16:32:45 2023

Reports the results of the catalog estimation
"""


from biogeme.results import compile_estimation_results, pareto_optimal


def report(dict_of_results):
    """Reports the results of the estimared catalogs"""
    print(f'A total of {len(dict_of_results)} models have been estimated')
    print('== Estimation results ==')

    compiled_results, specs = compile_estimation_results(
        dict_of_results, use_short_names=True
    )
    print(compiled_results)
    for short_name, spec in specs.items():
        print(f'{short_name}\t{spec}')

    pareto_results = pareto_optimal(dict_of_results)
    compiled_pareto_results, pareto_specs = compile_estimation_results(
        pareto_results, use_short_names=True
    )
    print(compiled_pareto_results)
    for short_name, spec in pareto_specs.items():
        print(f'{short_name}\t{spec}')
```

6 Estimation of a catalog with two models

```
"""File b01model.py

:author: Michel Bierlaire, EPFL
:date: Fri Jul 14 09:47:21 2023

Investigate several choice models:
- logit
- nested logit with two nests: public and private transportation
- nested logit with two nests existing and future modes
```

```python
for a total of 3 specifications.
"""
import biogeme.biogeme as bio
from biogeme import models
from biogeme.expressions import Beta
from biogeme.catalog import Catalog
from results_analysis import report
from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
    CAR_CO_SCALED,
)

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


# Definition of the utility functions
V1 = ASC_TRAIN + B_TIME * TRAIN_TT_SCALED + B_COST * TRAIN_COST_SCALED
V2 = 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_logit = models.loglogit(V, av, CHOICE)

MU_existing = Beta('MU_existing', 1, 1, 10, 0)
existing = MU_existing, [1, 3]
future = 1.0, [2]
nests_existing = existing, future
logprob_nested_existing = models.lognested(V, av,
```

```
    nests_existing, CHOICE)

MU_public = Beta('MU_public', 1, 1, 10, 0)
public = MU_public, [1, 2]
private = 1.0, [3]
nests_public = public, private
logprob_nested_public = models.lognested(V, av, nests_public,
    CHOICE)

model_catalog = Catalog.from_dict(
    catalog_name='model_catalog',
    dict_of_expressions={
        'logit': logprob_logit,
        'nested existing': logprob_nested_existing,
        'nested public': logprob_nested_public,
    },
)
# Create the Biogeme object
the_biogeme = bio.BIOGEME(database, model_catalog)
the_biogeme.modelName = 'b01model'
the_biogeme.generate_html = False
the_biogeme.generate_pickle = False

# Estimate the parameters
dict_of_results = the_biogeme.estimate_catalog()

report(dict_of_results)
```

7 Estimation of a catalog with nonlinear specifications

```
"""File b02nonlinear.py

:author: Michel Bierlaire, EPFL
:date: Thu Jul 13 21:31:54 2023

Investigate of nonlinear specifications for the travel time
    variables:
- linear specification,
- Box-Cox transform,
- power series,
for a total of 3 specifications.
"""
import biogeme.biogeme as bio
from biogeme import models
from biogeme.expressions import Beta
from biogeme.models import boxcox
from biogeme.catalog import Catalog
```

```python
from results_analysis import report
from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
    CAR_CO_SCALED,
)


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

# Non linear specifications for the travel time

# Parameter of the Box-Cox transform
ell_travel_time = Beta('lambda_travel_time', 1, -10, 10, 0)

# Coefficients of the power series
square_tt_coef = Beta('square_tt_coef', 0, None, None, 0)
cube_tt_coef = Beta('cube_tt_coef', 0, None, None, 0)


def power_series(the_variable):
    """Generate the expression of a polynomial of degree 3

    :param the_variable: variable of the polynomial
    :type the_variable: biogeme.expressions.Expression
    """
    return (
        the_variable
        + square_tt_coef * the_variable**2
        + cube_tt_coef * the_variable * the_variable**3
    )


linear_train_tt = TRAIN_TT_SCALED
boxcox_train_tt = boxcox(TRAIN_TT_SCALED, ell_travel_time)
power_train_tt = power_series(TRAIN_TT_SCALED)
train_tt_catalog = Catalog.from_dict(
```

```
    catalog_name='train_tt_catalog',
    dict_of_expressions={
        'linear': linear_train_tt,
        'boxcox': boxcox_train_tt,
        'power': power_train_tt,
    },
)

linear_sm_tt = SM_TT_SCALED
boxcox_sm_tt = boxcox(SM_TT_SCALED, ell_travel_time)
power_sm_tt = power_series(SM_TT_SCALED)
sm_tt_catalog = Catalog.from_dict(
    catalog_name='sm_tt_catalog',
    dict_of_expressions={
        'linear': linear_sm_tt,
        'boxcox': boxcox_sm_tt,
        'power': power_sm_tt,
    },
    controlled_by=train_tt_catalog.controlled_by,
)

linear_car_tt = CAR_TT_SCALED
boxcox_car_tt = boxcox(CAR_TT_SCALED, ell_travel_time)
power_car_tt = power_series(CAR_TT_SCALED)
car_tt_catalog = Catalog.from_dict(
    catalog_name='car_tt_catalog',
    dict_of_expressions={
        'linear': linear_car_tt,
        'boxcox': boxcox_car_tt,
        'power': power_car_tt,
    },
    controlled_by=train_tt_catalog.controlled_by,
)


# Definition of the utility functions
V1 = ASC_TRAIN + B_TIME * train_tt_catalog + B_COST * TRAIN_COST_SCALED
V2 = B_TIME * sm_tt_catalog + B_COST * SM_COST_SCALED
V3 = ASC_CAR + B_TIME * car_tt_catalog + 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.
```

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

# Create the Biogeme object
the_biogeme = bio.BIOGEME(database, logprob)
the_biogeme.modelName = 'b02nonlinear'
the_biogeme.generate_html = False
the_biogeme.generate_pickle = False

# Estimate the parameters
dict_of_results = the_biogeme.estimate_catalog()

report(dict_of_results)
```

8 Estimation of a catalog with alternative-specific coefficients

```python
"""File b03alt_spec.py

:author: Michel Bierlaire, EPFL
:date: Thu Jul 13 16:18:10 2023

Investigate alternative specific parameters:
- two specifications for the travel time coefficient: generic, and alternative specific,
- two specifications for the travel cost coefficient: generic, and alternative specific,
for a total of 4 specifications.
"""
import numpy as np
import biogeme.biogeme as bio
from biogeme import models
from biogeme.expressions import Beta
from biogeme.catalog import generic_alt_specific_catalogs

from results_analysis import report
from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
    CAR_CO_SCALED,
)
```

```python

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

(B_TIME_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_TIME', beta_parameters=[B_TIME], alternatives=('TRAIN', 'SM', 'CAR')
)

(B_COST_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_COST', beta_parameters=[B_COST], alternatives=('TRAIN', 'SM', 'CAR')
)

# Definition of the utility functions
V1 = (
    ASC_TRAIN
    + B_TIME_catalog_dict['TRAIN'] * TRAIN_TT_SCALED
    + B_COST_catalog_dict['TRAIN'] * TRAIN_COST_SCALED
)
V2 = B_TIME_catalog_dict['SM'] * SM_TT_SCALED + B_COST_catalog_dict['SM'] * SM_COST_SCALED
V3 = (
    ASC_CAR
    + B_TIME_catalog_dict['CAR'] * CAR_TT_SCALED
    + B_COST_catalog_dict['CAR'] * 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
the_biogeme = bio.BIOGEME(database, logprob)
the_biogeme.modelName = 'b01alt_spec'
the_biogeme.generate_html = False
the_biogeme.generate_pickle = False

# Estimate the parameters
dict_of_results = the_biogeme.estimate_catalog()
```

```

report(dict_of_results)
```

9 Estimation of a catalog with segmentations

```
"""File b04segmentation.py

:author: Michel Bierlaire, EPFL
:date: Thu Jul 13 16:18:10 2023

Investigate the segmentations of parameters.

We consider 4 specifications for the constants:
- Not segmented
- Segmented by GA (yearly subscription to public transport)
- Segmented by luggage
- Segmented both by GA and luggage

We consider 3 specifications for the time coefficients:
- Not Segmented
- Segmented with first class
- Segmented with trip purpose

We obtain a total of 12 specifications.
"""
import numpy as np
import biogeme.biogeme as bio
from biogeme import models
from biogeme.expressions import Beta
from biogeme.catalog import segmentation_catalogs
from results_analysis import report
from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
    CAR_CO_SCALED,
)

segmentation_ga = database.generate_segmentation(
    variable='GA', mapping={0: 'noGA', 1: 'GA'}
)
```

```

segmentation_luggage = database.generate_segmentation(
    variable='LUGGAGE', mapping={0: 'no_lugg', 1: 'one_lugg',
        3: 'several_lugg'}
)

segmentation_first = database.generate_segmentation(
    variable='FIRST', mapping={0: '2nd_class', 1: '1st_class'}
)

# We consider two trip purposes: 'commuters' and anything else.
    We
# need to define a binary variable first

database.data['COMMUTERS'] = np.where(database.data['PURPOSE']
    == 1, 1, 0)

segmentation_purpose = database.generate_segmentation(
    variable='COMMUTERS', mapping={0: 'non_commuters', 1:
        'commuters'}
)


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

ASC_TRAIN_catalog, ASC_CAR_catalog = segmentation_catalogs(
    generic_name='ASC',
    beta_parameters=[ASC_TRAIN, ASC_CAR],
    potential_segmentations=(
        segmentation_ga,
        segmentation_luggage,
    ),
    maximum_number=2,
)

# Note that the function returns a list of catalogs. Here, the
    list
# contains only one of them.  This is why there is a comma after
# "B_TIME_catalog".
(B_TIME_catalog,) = segmentation_catalogs(
    generic_name='B_TIME',
    beta_parameters=[B_TIME],
    potential_segmentations=(
        segmentation_first,
        segmentation_purpose,
```

```
    ),
    maximum_number=1,
)

# Definition of the utility functions
V1 = ASC_TRAIN_catalog + B_TIME_catalog * TRAIN_TT_SCALED +
    B_COST * TRAIN_COST_SCALED
V2 = B_TIME_catalog * SM_TT_SCALED + B_COST * SM_COST_SCALED
V3 = ASC_CAR_catalog + B_TIME_catalog * 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
the_biogeme = bio.BIOGEME(database, logprob)
the_biogeme.modelName = 'b04segmentation'
the_biogeme.generate_html = False
the_biogeme.generate_pickle = False

# Estimate the parameters
dict_of_results = the_biogeme.estimate_catalog()

report(dict_of_results)
```

10 Estimation of a catalog with segmentations and alternative-specific coefficients

```
"""File b05alt_spec_segmentation.py

:author: Michel Bierlaire, EPFL
:date: Thu Jul 13 16:18:10 2023

Investigate segmentations of parameters and alternative
    specific specification

We consider 4 specifications for the constants:
- Not segmented
- Segmented by GA (yearly subscription to public transport)
- Segmented by luggage
- Segmented both by GA and luggage
```

```

We consider 6 specifications for the time coefficients:
- Generic and not segmented
- Generic and segmented with first class
- Generic and segmented with trip purpose
- Alternative specific and not segmented
- Alternative specific and segmented with first class
- Alternative specific and segmented with trip purpose

We consider 2 specifications for the cost coefficients:
- Generic
- Alternative specific

We obtain a total of 48 specifications.
"""
import numpy as np
import biogeme.biogeme as bio
from biogeme import models
from biogeme.expressions import Beta
from biogeme.catalog import segmentation_catalogs,
    generic_alt_specific_catalogs
from results_analysis import report
from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
    CAR_CO_SCALED,
)

segmentation_ga = database.generate_segmentation(
    variable='GA', mapping={0: 'noGA', 1: 'GA'}
)

segmentation_luggage = database.generate_segmentation(
    variable='LUGGAGE', mapping={0: 'no_lugg', 1: 'one_lugg',
        3: 'several_lugg'}
)

segmentation_first = database.generate_segmentation(
    variable='FIRST', mapping={0: '2nd_class', 1: '1st_class'}
)

```

```

# We consider two trip purposes: 'commuters' and anything else. We
# need to define a binary variable first

database.data['COMMUTERS'] = np.where(database.data['PURPOSE'] == 1, 1, 0)

segmentation_purpose = database.generate_segmentation(
    variable='COMMUTERS', mapping={0: 'non_commuters', 1: 'commuters'}
)


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

ASC_TRAIN_catalog, ASC_CAR_catalog = segmentation_catalogs(
    generic_name='ASC',
    beta_parameters=[ASC_TRAIN, ASC_CAR],
    potential_segmentations=(
        segmentation_ga,
        segmentation_luggage,
    ),
    maximum_number=2,
)


(B_TIME_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_TIME',
    beta_parameters=[B_TIME],
    alternatives=['TRAIN', 'SM', 'CAR'],
    potential_segmentations=(
        segmentation_first,
        segmentation_purpose,
    ),
    maximum_number=1,
)

(B_COST_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_COST', beta_parameters=[B_COST], alternatives=['TRAIN', 'SM', 'CAR']
)

# Definition of the utility functions
V1 = (
```

```
    ASC_TRAIN_catalog
    + B_TIME_catalog_dict['TRAIN'] * TRAIN_TT_SCALED
    + B_COST_catalog_dict['TRAIN'] * TRAIN_COST_SCALED
)
V2 = B_TIME_catalog_dict['SM'] * SM_TT_SCALED + B_COST_catalog_dict['SM'] * SM_COST_SCALED
V3 = (
    ASC_CAR_catalog
    + B_TIME_catalog_dict['CAR'] * CAR_TT_SCALED
    + B_COST_catalog_dict['CAR'] * 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
the_biogeme = bio.BIOGEME(database, logprob)
the_biogeme.modelName = 'b05alt_spec_segmentation'
the_biogeme.generate_html = False
the_biogeme.generate_pickle = False

# Estimate the parameters
dict_of_results = the_biogeme.estimate_catalog()

report(dict_of_results)
```

11 Specification of a catalog with 432 configurations

```
"""File everything_spec.py

:author: Michel Bierlaire, EPFL
:date: Sat Jul 15 15:40:33 2023

We investigate various specifications:
- 3 models
    - logit
    - nested logit with two nests: public and private transportation
    - nested logit with two nests existing and future modes
- 3 functional forms for the travel time variables
```

```
    - linear specification,
    - Box-Cox transform,
    - power series,
- 2 specifications for the cost coefficients:
    - generic
    - alternative specific
- 2 specifications for the travel time coefficients:
    - generic
    - alternative specific
- 4 segmentations for the constants:
    - not segmented
    - segmented by GA (yearly subscription to public transport)
    - segmented by luggage
    - segmented both by GA and luggage
-  3 segmentations for the time coefficients:
    - not segmented
    - segmented with first class
    - segmented with trip purpose

This leads to a total of 432 specifications.
"""
import numpy as np
from biogeme import models
from biogeme.expressions import Beta
from biogeme.catalog import (
    Catalog,
    segmentation_catalogs,
    generic_alt_specific_catalogs,
)

from swissmetro_data import (
    database,
    CHOICE,
    SM_AV,
    CAR_AV_SP,
    TRAIN_AV_SP,
    TRAIN_TT_SCALED,
    TRAIN_COST_SCALED,
    SM_TT_SCALED,
    SM_COST_SCALED,
    CAR_TT_SCALED,
    CAR_CO_SCALED,
)

segmentation_ga = database.generate_segmentation(
    variable='GA', mapping={0: 'noGA', 1: 'GA'}
)

segmentation_luggage = database.generate_segmentation(
```

```
    variable='LUGGAGE', mapping={0: 'no_lugg', 1: 'one_lugg', 3: 'several_lugg'}
)

segmentation_first = database.generate_segmentation(
    variable='FIRST', mapping={0: '2nd_class', 1: '1st_class'}
)

# We consider two trip purposes: 'commuters' and anything else. We
# need to define a binary variable first

database.data['COMMUTERS'] = np.where(database.data['PURPOSE'] == 1, 1, 0)

segmentation_purpose = database.generate_segmentation(
    variable='COMMUTERS', mapping={0: 'non_commuters', 1: 'commuters'}
)


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

# Non linear specifications for the travel time

# Parameter of the Box-Cox transform
ell_travel_time = Beta('lambda_travel_time', 1, -10, 10, 0)

# Coefficients of the power series
square_tt_coef = Beta('square_tt_coef', 0, None, None, 0)
cube_tt_coef = Beta('cube_tt_coef', 0, None, None, 0)


def power_series(the_variable):
    """Generate the expression of a polynomial of degree 3

    :param the_variable: variable of the polynomial
    :type the_variable: biogeme.expressions.Expression
    """
    return (
        the_variable
        + square_tt_coef * the_variable**2
        + cube_tt_coef * the_variable * the_variable**3
    )

```

```

linear_train_tt = TRAIN_TT_SCALED
boxcox_train_tt = models.boxcox(TRAIN_TT_SCALED,
    ell_travel_time)
power_train_tt = power_series(TRAIN_TT_SCALED)
train_tt_catalog = Catalog.from_dict(
    catalog_name='train_tt_catalog',
    dict_of_expressions={
        'linear': linear_train_tt,
        'boxcox': boxcox_train_tt,
        'power': power_train_tt,
    },
)

linear_sm_tt = SM_TT_SCALED
boxcox_sm_tt = models.boxcox(SM_TT_SCALED, ell_travel_time)
power_sm_tt = power_series(SM_TT_SCALED)
sm_tt_catalog = Catalog.from_dict(
    catalog_name='sm_tt_catalog',
    dict_of_expressions={
        'linear': linear_sm_tt,
        'boxcox': boxcox_sm_tt,
        'power': power_sm_tt,
    },
    controlled_by=train_tt_catalog.controlled_by,
)

linear_car_tt = CAR_TT_SCALED
boxcox_car_tt = models.boxcox(CAR_TT_SCALED, ell_travel_time)
power_car_tt = power_series(CAR_TT_SCALED)

car_tt_catalog = Catalog.from_dict(
    catalog_name='car_tt_catalog',
    dict_of_expressions={
        'linear': linear_car_tt,
        'boxcox': boxcox_car_tt,
        'power': power_car_tt,
    },
    controlled_by=train_tt_catalog.controlled_by,
)


ASC_TRAIN_catalog, ASC_CAR_catalog = segmentation_catalogs(
    generic_name='ASC',
    beta_parameters=[ASC_TRAIN, ASC_CAR],
    potential_segmentations=(
        segmentation_ga,
        segmentation_luggage,
    ),
```

```
    maximum_number=2,
)


(B_TIME_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_TIME',
    beta_parameters=[B_TIME],
    alternatives=['TRAIN', 'SM', 'CAR'],
    potential_segmentations=(
        segmentation_first,
        segmentation_purpose,
    ),
    maximum_number=1,
)

(B_COST_catalog_dict,) = generic_alt_specific_catalogs(
    generic_name='B_COST', beta_parameters=[B_COST], alternatives=['TRAIN', 'SM', 'CAR']
)

# Definition of the utility functions
V1 = (
    ASC_TRAIN_catalog
    + B_TIME_catalog_dict['TRAIN'] * train_tt_catalog
    + B_COST_catalog_dict['TRAIN'] * TRAIN_COST_SCALED
)
V2 = B_TIME_catalog_dict['SM'] * sm_tt_catalog + B_COST_catalog_dict['SM'] * SM_COST_SCALED
V3 = (
    ASC_CAR_catalog
    + B_TIME_catalog_dict['CAR'] * car_tt_catalog
    + B_COST_catalog_dict['CAR'] * 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_logit = models.loglogit(V, av, CHOICE)

MU_existing = Beta('MU_existing', 1, 1, 10, 0)
existing = MU_existing, [1, 3]
future = 1.0, [2]
nests_existing = existing, future
logprob_nested_existing = models.lognested(V, av,
```

```
    nests_existing , CHOICE)

MU_public = Beta('MU_public', 1, 1, 10, 0)
public = MU_public, [1, 2]
private = 1.0, [3]
nests_public = public, private
logprob_nested_public = models.lognested(V, av, nests_public,
    CHOICE)

model_catalog = Catalog.from_dict(
    catalog_name='model_catalog',
    dict_of_expressions={
        'logit': logprob_logit,
        'nested existing': logprob_nested_existing,
        'nested public': logprob_nested_public,
    },
)
```

12 Assisted Specification of a catalog with 432 configurations

```
"""File b07everything_assisted.py

:author: Michel Bierlaire, EPFL
:date: Sat Jul 15 15:02:20 2023

Investigate various specifications:
- 3 models
    - logit
    - nested logit with two nests: public and private
      transportation
    - nested logit with two nests existing and future modes
- 3 functional form for the travel time variables
    - linear specification,
    - Box-Cox transform,
    - power series,
- 2 specification for the cost coefficients:
    - generic
    - alternative specific
- 2 specification for the travel time coefficients:
    - generic
    - alternative specific
- 4 segmentations for the constants:
    - not segmented
    - segmented by GA (yearly subscription to public transport)
    - segmented by luggage
    - segmented both by GA and luggage
-  3 segmentations for the time coefficients:
```

```python
    - not segmented
    - segmented with first class
    - segmented with trip purpose

This leads to a total of 432 specifications.
The algorithm implemented in the AssistedSpecification object is used to
investigate some of these specifications.

"""
import biogeme.logging as blog
import biogeme.biogeme as bio
from biogeme.assisted import AssistedSpecification
from biogeme.multiobjectives import loglikelihood_dimension
from everything_spec import model_catalog, database
from results_analysis import report

logger = blog.get_screen_logger(level=blog.DEBUG)
logger.info('Example b07everything_assisted')

PARETO_FILE_NAME = 'b07everything_assisted.pareto'

def validity(results):
    """Function verifying that the estimation results are valid.

    The results are not valid if any of the time or cost coefficient is non negative.
    """
    for beta in results.data.betas:
        if 'TIME' in beta.name and beta.value >= 0:
            return False, f'{beta.name} = {beta.value}'
        if 'COST' in beta.name and beta.value >= 0:
            return False, f'{beta.name} = {beta.value}'
    return True, None

# Create the Biogeme object
the_biogeme = bio.BIOGEME(database, model_catalog)
the_biogeme.modelName = 'b07everything'
the_biogeme.generate_html = False
the_biogeme.generate_pickle = False

# Estimate the parameters
assisted_specification = AssistedSpecification(
    biogeme_object=the_biogeme,
    multi_objectives=loglikelihood_dimension,
    pareto_file_name=PARETO_FILE_NAME,
    validity=validity,
)

```

```
non_dominated_models = assisted_specification.run()

report(non_dominated_models)
```

13 Postprocessing

```
"""File b09post_processing.py

:author: Michel Bierlaire, EPFL
:date: Thu Jul 20 17:15:37 2023

We consider the model with 432 specifications:
- 3 models
    - logit
    - nested logit with two nests: public and private
      transportation
    - nested logit with two nests existing and future modes
- 3 functional form for the travel time variables
    - linear specification,
    - Box-Cox transform,
    - power series,
- 2 specification for the cost coefficients:
    - generic
    - alternative specific
- 2 specification for the travel time coefficients:
    - generic
    - alternative specific
- 4 segmentations for the constants:
    - not segmented
    - segmented by GA (yearly subscription to public transport)
    - segmented by luggage
    - segmented both by GA and luggage
-  3 segmentations for the time coefficients:
    - not segmented
    - segmented with first class
    - segmented with trip purpose

This leads to a total of 432 specifications.

After running the assisted specification algorithm, we use post
processing to re-estimate all Pareto optimal models, and
    display some
information about the algorithm

"""
try:
    import matplotlib.pyplot as plt
    can_plot = True
```

```
except ModuleNotFoundError:
    can_plot = False
import biogeme.logging as blog
import biogeme.biogeme as bio
from biogeme.assisted import ParetoPostProcessing

from everything_spec import model_catalog, database

logger = blog.get_screen_logger(level=blog.INFO)
logger.info('Example b08selected_specification')

PARETO_FILE_NAME = 'b07everything_assisted.pareto'

the_biogeme = bio.BIOGEME(database, model_catalog)
the_biogeme.modelName = 'b09post_processing'

post_processing = ParetoPostProcessing(
    biogeme_object=the_biogeme,
        pareto_file_name=PARETO_FILE_NAME
)

post_processing.reestimate(recycle=True)

if can_plot:
    _ = post_processing.plot(
        label_x='Nbr of parameters',
        label_y='Negative log likelihood',
        objective_x=1,
        objective_y=0,
    )
    plt.savefig('pareto.eps', format='eps', dpi=300)
```