Estimating MEV models with samples of alternatives

Michel Bierlaire Evangelos Paschalidis

December 25, 2023

Report TRANSP-OR 231225
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
transp-or.epfl.ch

Series on Biogeme

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

This document describes how to use Biogeme to estimate Multivariate Extreme Value (MEV) models using only a sample of alternatives.

We assume that the reader is already familiar with discrete choice models, and has successfully installed the package. Biogeme is a Python package written in Python and C++, that relies on the Pandas library for the management of the data. This document has been written using Biogeme 3.2.13.

In the rest of the document, we walk the reader through a concrete example. The theoretical background is available in the Appendix.

1 The context

The example that we are considering is the development of a restaurant choice model.

1.1 Data: the alternatives

We have a list of 100 restaurants, which constitutes our full choice set $\mathcal{C}$. For each restaurant, we know:

- its rating, on a scale from 1 to 5,
- its price level, on a scale from 1 (cheap) to 4 (very expensive),
- its category, which is one element of the following list: Chinese, Japanese, Korean, Indian, French, Mexican, Lebanese, or Ethiopian,
- its exact location, in an arbitrary coordinate system, where the "latitudes" and "longitudes" range from 0 to 100.

We consider also two other variables:

- a dummy variable that is 1 if the restaurant is Asian, that is, if it is Chinese, Japanese, Korean or Indian,
- a dummy variable that is 1 if the restaurant is located "downtown", which means at a distance less or equal than 80 from the location (0,0), as illustrated in Figure 1.

The data is stored in the file `restaurants.dat` and the column labels are:

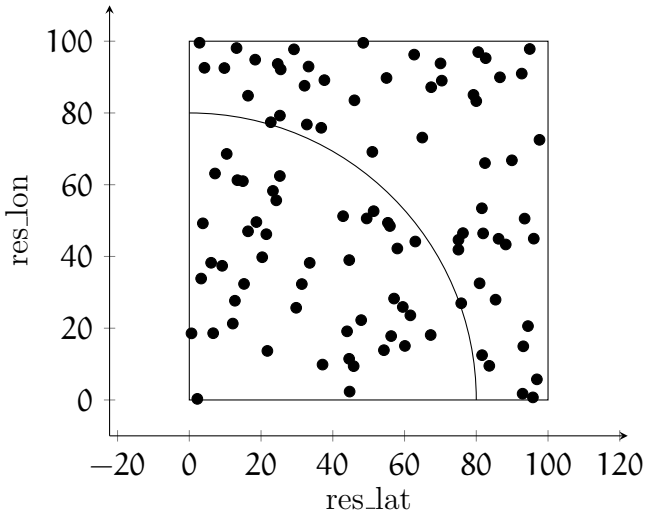

Figure 1: Location of the restaurants

```
ID, rating, price, category_Chinese, category_Japanese,
    category_Korean, category_Indian, category_French,
    category_Mexican, category_Lebanese, category_Ethiopian,
    Asian, rest_lat, rest_lon, distance, downtown
```

1.2 Data: observed choices

We have access to a sample of 10'000 individuals. For each of them we know

- her location in the same coordinate system as the restaurants,
- the chosen restaurant.

The "observed" choice is synthetic. It has been generated by simulation using a postulated ground truth. The utility function of each alternative contains the following variables, each associated with a coefficient:

- the rating of the restaurant,
- the price of the restaurant,
- a dummy variable that is 1 if the restaurant is Chinese,
- a dummy variable that is 1 if the restaurant is Japanese,
- a dummy variable that is 1 if the restaurant is Korean,

- a dummy variable that is 1 if the restaurant is Indian,
- a dummy variable that is 1 if the restaurant is French,
- a dummy variable that is 1 if the restaurant is Mexican,
- a dummy variable that is 1 if the restaurant is Lebanese,
- a dummy variable that is 1 if the restaurant is Ethiopian,
- the log of the distance between the decision maker and the restaurant.

We have postulated three different models to generate the synthetic choices:

1. a logit model;
2. a nested logit model where all the Asian alternatives are grouped in a nest, while each other alternative is alone in a nest;
3. a cross-nested logit model where all the Asian alternatives are grouped in a nest, and all the restaurant located downtown are grouped in a nest, while each other alternative is alone in a nest.

The assumed value of the parameters are reported in Table 1.

β_{rating}	0.75
β_{price}	−0.4
β_{chinese}	0.75
β_{japanese}	1.25
β_{korean}	0.75
β_{indian}	1.0
β_{french}	0.75
β_{mexican}	1.25
β_{lebanese}	0.75
$\beta_{\text{ethiopian}}$	0.5
β_{logdist}	−0.6
μ_{asian}	2.0
μ_{downtown}	2.0

Table 1: "True" values of the parameters

For each model, 5 synthetic choices have been generated. The data is stored in the file `obs_choice.dat` and the column labels are

```
  user_lat, user_lon,
  logit_0, logit_1, logit_2, logit_3, logit_4,
  nested_0, nested_1, nested_2, nested_3, nested_4,
  cnl_0, cnl_1, cnl_2, cnl_3, cnl_4
```

2 Sets of alternatives

The script `alternatives.py` (see Section E.1) provides sets of alternatives from the data file, and defines partitions for the sampling. In the example that we are considering, there are 100 alternatives, numbered from 0 to 99.

Among them, they are 33 Asian restaurants:

```
0, 1, 3, 13, 15, 17, 18, 27, 31, 33, 34, 37, 40, 45, 47, 50,
   51, 55, 57, 68, 70, 72, 76, 78, 79, 80, 81, 87, 89, 91, 92,
   94, 98
```

They are 44 restaurants located downtown:

```
0, 7, 9, 11, 13, 14, 15, 17, 18, 20, 22, 24, 30, 32, 33, 36,
   37, 38, 45, 46, 47, 48, 49, 50, 54, 55, 56, 60, 61, 63, 70,
   74, 75, 77, 80, 81, 83, 84, 86, 87, 88, 90, 93, 97, 98, 99
```

In addition to those two sets, the script defines the following sets:

```
all_alternatives
# Set of Asian restaurants in downtown
asian_and_downtown = asian & downtown

# Set of Asian restaurants, and of restaurants in downtown
asian_or_downtown = asian | downtown

# Set of Asian restaurants not in downtown
only_asian = asian - asian_and_downtown

# Set of non Asian restaurants in downtown
only_downtown = downtown - asian_and_downtown

# Set of restaurants that are neither Asian nor in downtown
others = all_alternatives - asian_or_downtown
```

The script also defines partitions, using the `Partition` class, that can be imported as follows:

```
from biogeme.partition import Partition
```

A partition is defined by a list of sets of alternatives, and by the complete set of all alternatives. Five partitions are defined in the script:

- A partition of the full choice set into Asian and non-Asian restaurants:

```
partition_asian = Partition([asian, complement(asian)],
   full_set=all_alternatives)
```

Note the use of the `complement` function here, that basically returns

```
  all_alternatives - asian.
```

- A partition of the full choice set into downtown and non-downtown restaurants:

```
partition_downtown = Partition(
    [downtown, complement(downtown)],
       full_set=all_alternatives
)
```

- A "partition" of the full choice set containing only one segment with all alternatives:

```
partition_uniform = Partition([all_alternatives],
   full_set=all_alternatives)
```

  This is designed to perform a uniform sampling.

- For the nested logit model, we have a partition of the set of Asian restaurants, with only one segment in order to perform a uniform sampling:

```
partition_uniform_asian = Partition([asian], full_set=asian)
```

- For the cross-nested logit model, we have a partition of the set of restaurants that are either Asian or downtown, with only one segment in order to perform a uniform sampling:

```
partition_uniform_asian_or_downtown = Partition(
    [asian_or_downtown], full_set=asian_or_downtown
)
```

Those partitions are gathered into a dictionary so that they can be obtained from their name.

3 Model specification

The specification of the utility function, as well as of additional variables, are available in the script `specification.py` (Section E.2). In the context of sampling of alternatives, alternatives are unlabeled. Therefore, any alternative specific constant must be captured by dummy variables. In our example, the specification of the utility function is

```
beta_rating = Beta('beta_rating', 0, None, None, 0)
beta_price = Beta('beta_price', 0, None, None, 0)
beta_chinese = Beta('beta_chinese', 0, None, None, 0)
beta_japanese = Beta('beta_japanese', 0, None, None, 0)
beta_korean = Beta('beta_korean', 0, None, None, 0)
beta_indian = Beta('beta_indian', 0, None, None, 0)
beta_french = Beta('beta_french', 0, None, None, 0)
beta_mexican = Beta('beta_mexican', 0, None, None, 0)
beta_lebanese = Beta('beta_lebanese', 0, None, None, 0)
beta_ethiopian = Beta('beta_ethiopian', 0, None, None, 0)
beta_log_dist = Beta('beta_log_dist', 0, None, None, 0)
V = (
    beta_rating * Variable('rating')
    + beta_price * Variable('price')
    + beta_chinese * Variable('category_Chinese')
    + beta_japanese * Variable('category_Japanese')
    + beta_korean * Variable('category_Korean')
    + beta_indian * Variable('category_Indian')
    + beta_french * Variable('category_French')
    + beta_mexican * Variable('category_Mexican')
    + beta_lebanese * Variable('category_Lebanese')
    + beta_ethiopian * Variable('category_Ethiopian')
    + beta_log_dist * Variable('log_dist')
    )
```

Note that the names of most variables correspond to the names of the columns in the data frame with alternatives. There is one exception in our example: the variable `log_dist`. Indeed, this captures the logarithm of the distance between each individual and each restaurant. This can be calculated only after the sampling of alternatives has been performed. We call such variables "combined variables". Those are variables that combine attributes of the alternatives and characteristics of the individuals. Such variables are defined using a `CrossVariableTuple`, that can be imported as follows:

```
from biogeme.sampling_of_alternatives import CrossVariableTuple
```

It has two entries: the name of the variable, and its expression. In our example, we have:

```
    CrossVariableTuple(
        'log_dist',
        log(
            (
                (Variable('user_lat') - Variable('rest_lat'))
                   ** 2
                + (Variable('user_lon') - Variable('rest_lon'))
                   ** 2
            )
            ** 0.5
```

```
        ),
    )
```

Note that a list of such tuples must be provided, even if there is only one. Therefore, the syntax for our example is

```
combined_variables = [
    CrossVariableTuple(
        'log_dist',
        log(
            (
                (Variable('user_lat') - Variable('rest_lat'))
                    ** 2
                + (Variable('user_lon') - Variable('rest_lon'))
                    ** 2
            )
            ** 0.5
        ),
    )
]
```

4 Estimation of a logit model

All the information needed to perform the estimation of the model must be gathered in an object of type `SamplingContext`, that involves 12 elements, 3 of them being necessary only for MEV models:

```
    the_partition: Partition
    sample_sizes: Iterable[int]
    individuals: pd.DataFrame
    choice_column: str
    alternatives: pd.DataFrame
    id_column: str
    biogeme_file_name: str
    utility_function: Expression
    combined_variables: list[CrossVariableTuple]
    mev_partition: Optional[Partition] = None
    mev_sample_sizes: Optional[Iterable[int]] = None
    cnl_nests: Optional[NestsForCrossNestedLogit] = None
```

1. `the_partition`: we first define a partition of the full choice set into K segments of size R_k: $J = \sum_{k=1}^{K} R_k$. To form a partition, the segments must be mutually exclusive and collectively exhaustive. In other words, each alternative must be contained in exactly one segment. In this example, we use the partition `'asian'` defined in the script `alternatives.py`.

2. `sample_sizes`: we then define a list of integers that provides the number of alternatives that must be sampled in each segment of the partition.

   It is often useful to use a "balanced" list, that is a list where a similar number of alternatives is sampled from each segment. This can be provided by the following statement:

   ```
   segment_sizes = generate_segment_size(SAMPLE_SIZE,
       the_partition.number_of_segments())
   ```

   where the function `generate_segment_size` is imported as follows:

   ```
   from biogeme.sampling_of_alternatives import
       generate_segment_size
   ```

   The function takes two arguments: the total size of the sample, and the number of segments in the partition. Here are some examples of what the function generates:

   ```
   >>> generate_segment_size(10, 3)
   [4, 3, 3]
   >>> generate_segment_size(11, 3)
   [4, 4, 3]
   >>> generate_segment_size(12, 3)
   [4, 4, 4]
   >>> generate_segment_size(13, 3)
   [5, 4, 4]
   >>> generate_segment_size(14, 3)
   [5, 5, 4]
   >>> generate_segment_size(15, 3)
   [5, 5, 5]
   ```

3. `individuals`: the next entry is the Pandas data frame containing the list of individuals and their observed choice. In our example, the data frame is obtained as follows:

   ```
   OBS_FILE = 'obs_choice.dat'
   observations = pd.read_csv(OBS_FILE)
   ```

4. `choice_column`: the next entry identifies the label of the column where the chosen alternative is reported. In our example, the file contains several synthetic choices. For example, we can have:

   ```
   CHOICE_COLUMN = 'logit_4'
   ```

5. `alternatives`: the next entry is the Pandas data frame containing the list of alternatives and their attributes. In our example, it is available from the script `alternatives.py` (see Section E.1):

```
from alternatives import alternatives
```

6. `id_column`: the next entry identifies the label of the column where the ID of the alternatives is reported. In our example, it happens to be `ID`, and it also available from the script `alternatives.py`:

   ```
   from alternatives import ID_COLUMN
   ```

7. `biogeme_file_name`: a data file in Biogeme format is generated for the model estimation. The name of this file must be provided here. For instance, `logit_asian_10_alt.dat`.

8. `utility_function`: the next entry is the specification of the utility function. In our example, it is defined in the script `specification.py`.

9. `combined_variables`: the next entry is the list of combined variables, that can be calculated only after the sample has been generated. In our example, it is defined in the script `specification.py`.

The context is therefore defined as follows:

```
context = SamplingContext(
    the_partition=the_partition,
    sample_sizes=segment_sizes,
    individuals=observations,
    choice_column=CHOICE_COLUMN,
    alternatives=alternatives,
    id_column=ID_COLUMN,
    biogeme_file_name=FILE_NAME,
    utility_function=V,
    combined_variables=combined_variables,
)
```

Once the context is defined, the process consists of the following steps:

- Generation of the choice sets:

  ```
  the_data_generation = ChoiceSetsGeneration(context=context)
  ```

- Generation of the Biogeme model:

  ```
  the_model_generation = GenerateModel(context=context)
  ```

- Generation of the Biogeme database:

  ```
  biogeme_database =
     the_data_generation.sample_and_merge(recycle=False)
  ```

Note that, once the database has been generated, it is dumped in CSV format in a file, using the provided file name. The content of this file is described in Section D. Its content can be recycled by changing the argument of the above function to True.

- We then retrieve the specification of the model itself:

```
logprob = the_model_generation.get_logit()
```

- The rest is a standard estimation procedure for Biogeme:

```
the_biogeme = bio.BIOGEME(biogeme_database, logprob)
the_biogeme.modelName = MODEL_NAME

# Calculate the null log likelihood for reporting.
the_biogeme.calculateNullLoglikelihood({i: 1 for i in
   range(SAMPLE_SIZE)})

# Estimate the parameters
results = the_biogeme.estimate(recycle=False)
print(results.short_summary())
estimated_parameters = results.getEstimatedParameters()
print(estimated_parameters)
```

- Finally, in this example, we compare the estimated values of the parameters with the true values:

```
df, msg = compare(estimated_parameters)
print(df)
print(msg)
```

The output generated by the script is the following:

```
Number of asian restaurants: 33
Size of the choice set: 100
Main partition: 2 segment(s) of size 33, 67
Main sample: 10: 5/33, 5/67

Generating 10 + None alternatives for 10000 observations
Define new variables
File logit_asian_10_alt.dat has been created.
...
Results saved in file logit_asian_10_alt.html
Results saved in file logit_asian_10_alt.pickle
Results for model logit_asian_10_alt
Nbr of parameters:              11
Sample size:                    10000
Excluded data:                  0
Null log likelihood:            -23025.85
```

```
Final log likelihood:           -18431.26
Likelihood ratio test (null):           9189.181
Rho square (null):                      0.2
Rho bar square (null):                  0.199
Akaike Information Criterion:   36884.52
Bayesian Information Criterion: 36963.84

                   Value  Rob. Std err  Rob. t-test  Rob.
                      p-value
beta_chinese    0.587478      0.050580    11.614717
   0.0
beta_ethiopian  0.469240      0.050444     9.302233
   0.0
beta_french     0.650104      0.061684    10.539239
   0.0
beta_indian     0.914430      0.042991    21.270247
   0.0
beta_japanese   1.158041      0.046356    24.981289
   0.0
beta_korean     0.735705      0.042533    17.297299
   0.0
beta_lebanese   0.679695      0.062281    10.913295
   0.0
beta_log_dist  -0.590966      0.015000   -39.397598
   0.0
beta_mexican    1.194666      0.036449    32.776230
   0.0
beta_price     -0.415953      0.012768   -32.577617
   0.0
beta_rating     0.749331      0.015382    48.714011
   0.0
              Name  True Value  Estimated Value    T-Test
0      beta_rating         0.75          0.749331  0.043496
1       beta_price        -0.40         -0.415953  1.249472
2     beta_chinese         0.75          0.587478  3.213145
3    beta_japanese         1.25          1.158041  1.983737
4      beta_korean         0.75          0.735705  0.336084
5      beta_indian         1.00          0.914430  1.990416
6      beta_french         0.75          0.650104  1.619472
7     beta_mexican         1.25          1.194666  1.518113
8    beta_lebanese         0.75          0.679695  1.128823
9   beta_ethiopian         0.50          0.469240  0.609783
10   beta_log_dist        -0.60         -0.590966 -0.602255
Parameters not estimated: ['mu_asian', 'mu_downtown']
```

The complete specification is available at Section E.3.

5 Estimation of a nested model

We estimate a nested logit model with a nest containing all the Asian restaurants.

For the estimation of the nested logit model, we need to define a sampling protocol for the MEV terms. This is done with the `mev_partition` and the `mev_sample_sizes` arguments of the context object. There are two important differences for this sampling protocol, compared to the other one, as explained in Section C. First, the chosen alternative does not play any role in the sampling procedure. And, second, it is not necessary to partition the full choice set. Alternatives that are alone in a nest do not contribute to the calculation of the MEV terms, and can therefore be excluded.

In this case, we sample from Asian alternatives only. We use a partition with a unique segment to perform a uniform sampling.

Then, the definition of the nests is done in the exact same way as for regular Biogeme models:

```
mu_asian = Beta('mu_asian', 1.0, 1.0, None, 0)
nest_asian = OneNestForNestedLogit(
    nest_param=mu_asian, list_of_alternatives=asian,
        name='asian'
)
nests = NestsForNestedLogit(
    choice_set=all_alternatives,
    tuple_of_nests=(nest_asian,),
)
```

The other difference with the estimation of the logit model is the extraction of the model itself:

```
logprob = the_model_generation.get_nested_logit(nests)
```

The complete specification is available at Section E.4.

The output generated by the script is the following:

```
Number of asian restaurants: 33
Size of the choice set: 100
Main partition: 2 segment(s) of size 46, 54
Main sample: 20: 10/46, 10/54
Nbr of MEV alternatives: 33
MEV partition: 1 segment(s) of size 33
MEV sample: 33: 33/33

Generating 20 + 33 alternatives for 10000 observations
Define new variables
File nested_downtown_20.dat has been created.
File biogeme.toml has been parsed.
...
```

```
Results saved in file nested_downtown_20.html
Results saved in file nested_downtown_20.pickle
Results for model nested_downtown_20
Nbr of parameters:              12
Sample size:                    10000
Excluded data:                  0
Null log likelihood:            -29957.32
Final log likelihood:           -22998.21
Likelihood ratio test (null):           13918.22
Rho square (null):                      0.232
Rho bar square (null):                  0.232
Akaike Information Criterion:   46020.43
Bayesian Information Criterion: 46106.95

                  Value  Rob. Std err  Rob. t-test  Rob.
                     p-value
beta_chinese    0.696196      0.071740     9.704438
   0.0
beta_ethiopian  0.499692      0.040332    12.389559
   0.0
beta_french     0.725736      0.048952    14.825608
   0.0
beta_indian     0.964207      0.063969    15.073097
   0.0
beta_japanese   1.220851      0.054594    22.362450
   0.0
beta_korean     0.689393      0.062379    11.051632
   0.0
beta_lebanese   0.718121      0.049852    14.405038
   0.0
beta_log_dist  -0.599724      0.012922   -46.412163
   0.0
beta_mexican    1.191239      0.029009    41.063987
   0.0
beta_price     -0.400633      0.012225   -32.771281
   0.0
beta_rating     0.762716      0.015255    49.999109
   0.0
mu_asian        2.020386      0.059454    33.982519
   0.0
              Name  True Value  Estimated Value    T-Test
0      beta_rating         0.75          0.762716 -0.833553
1       beta_price        -0.40         -0.400633  0.051812
2     beta_chinese         0.75          0.696196  0.749980
3    beta_japanese         1.25          1.220851  0.533933
4      beta_korean         0.75          0.689393  0.971587
5      beta_indian         1.00          0.964207  0.559546
6      beta_french         0.75          0.725736  0.495671
7     beta_mexican         1.25          1.191239  2.025583
```

```
8     beta_lebanese         0.75          0.718121   0.639477
9    beta_ethiopian         0.50          0.499692   0.007648
10    beta_log_dist        -0.60         -0.599724  -0.021395
11         mu_asian         2.00          2.020386  -0.342885
Parameters not estimated: ['mu_downtown']
```

The complete specification is available at Section E.4.

6 Estimation of a cross-nested model

We estimate a cross-nested logit model with a nest containing all the Asian restaurants, and another nest containing all the downtown restaurants. The procedure is similar to the one of the previous section. Except that the definition of the nests must be provided in the context object. The reason is that the membership parameters α are included in the data file. In our example, all alternatives belonging to both nests are associated with a membership parameter set to 0.5.

The definition of the nests is done as follows:

```
mu_downtown = Beta('mu_downtown', 1, 1, None, 0)
downtown_alpha_dict = {i: 0.5 for i in asian_and_downtown} | {
    i: 1 for i in only_downtown
}
downtown_nest = OneNestForCrossNestedLogit(
    nest_param=mu_downtown, dict_of_alpha=downtown_alpha_dict,
        name='downtown'
)

mu_asian = Beta('mu_asian', 1, 1, None, 0)
asian_alpha_dict = {i: 0.5 for i in asian_and_downtown} | {i:
   1.0 for i in only_asian}
asian_nest = OneNestForCrossNestedLogit(
    nest_param=mu_asian, dict_of_alpha=asian_alpha_dict,
        name='asian'
)

cnl_nests = NestsForCrossNestedLogit(
    choice_set=all_alternatives,
    tuple_of_nests=(downtown_nest, asian_nest),
)
```

And the definition of the context is

```
context = SamplingContext(
    the_partition=the_partition,
    sample_sizes=segment_sizes,
    individuals=observations,
    choice_column=CHOICE_COLUMN,
```

```
    alternatives=alternatives,
    id_column=ID_COLUMN,
    biogeme_file_name=FILE_NAME,
    utility_function=V,
    combined_variables=combined_variables,
    mev_partition=mev_partition,
    mev_sample_sizes=mev_segment_sizes,
    cnl_nests=cnl_nests,
)
```

The other difference with the estimation of the nested logit model is the extraction of the model itself:

```
logprob = the_model_generation.get_cross_nested_logit()
```

The complete specification is available at Section E.5. The output generated by the script is the following:

```
Number of asian restaurants: 33
Size of the choice set: 100
Main partition: 2 segment(s) of size 46, 54
Main sample: 10: 5/46, 5/54
Nbr of MEV alternatives: 63
MEV partition: 1 segment(s) of size 63
MEV sample: 63: 63/63

Generating 10 + 63 alternatives for 10000 observations
Define new variables
File cnl_10_63.dat has been created.
File biogeme.toml has been parsed.
...
Results saved in file cnl_10_63.html
Results saved in file cnl_10_63.pickle
Results for model cnl_10_63
Nbr of parameters:              13
Sample size:                    10000
Excluded data:                  0
Null log likelihood:            -23025.85
Final log likelihood:           -14075.7
Likelihood ratio test (null):          17900.29
Rho square (null):                     0.389
Rho bar square (null):                 0.388
Akaike Information Criterion:   28177.41
Bayesian Information Criterion: 28271.14

                   Value  Rob. Std err  Rob. t-test  Rob.
                      p-value
beta_chinese    0.773087      0.061077    12.657588
   0.0
beta_ethiopian  0.530632      0.041761    12.706267
   0.0
```

```
beta_french      0.781766       0.049737     15.717875
    0.0
beta_indian      1.104800       0.052860     20.900552
    0.0
beta_japanese    1.321261       0.046652     28.321448
    0.0
beta_korean      0.800695       0.053581     14.943756
    0.0
beta_lebanese    0.777865       0.049342     15.764714
    0.0
beta_log_dist   -0.578576       0.012611    -45.878882
    0.0
beta_mexican     1.256891       0.030219     41.592729
    0.0
beta_price      -0.412996       0.012435    -33.212769
    0.0
beta_rating      0.749742       0.014775     50.742764
    0.0
mu_asian         2.110265       0.067011     31.491220
    0.0
mu_downtown      1.932841       0.030166     64.072716
    0.0
              Name  True Value  Estimated Value    T-Test
0      beta_rating         0.75          0.749742  0.017432
1       beta_price        -0.40         -0.412996  1.045136
2     beta_chinese         0.75          0.773087 -0.378002
3    beta_japanese         1.25          1.321261 -1.527495
4      beta_korean         0.75          0.800695 -0.946153
5      beta_indian         1.00          1.104800 -1.982602
6      beta_french         0.75          0.781766 -0.638673
7     beta_mexican         1.25          1.256891 -0.228048
8    beta_lebanese         0.75          0.777865 -0.564732
9   beta_ethiopian         0.50          0.530632 -0.733511
10   beta_log_dist        -0.60         -0.578576 -1.698804
11         mu_asian         2.00          2.110265 -1.645472
12     mu_downtown         2.00          1.932841  2.226278
```

References

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

Bierlaire, M. and Krueger, R. (2020). Sampling and discrete choice, *Technical Report TRANSP-OR 201109*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Guevara, C. A. and Ben-Akiva, M. E. (2013). Sampling of alternatives in Multivariate Extreme Value (MEV) models, *Transportation Research Part B* **48**: 31–52.

A Theoretical background

We denote by $\mathcal{C}$ the full choice set, containing J alternatives. We assume, without loss of generality, that it is the same for all individuals. And we consider a choice model

$$\Pr(i|\mathcal{C};\theta),$$

for which we want to estimate the vector of parameters θ from data.

When the choice set is large, we are using a sample of alternatives. We consider a sampling protocol, based on importance sampling, that generates a subset $\mathcal{D}_n$ from $\mathcal{C}$ for each individual n in the sample. Note that the sampling protocol must be exogenous, in the sense that the probability for each non chosen alternative to be in the choice set must not depend on the chosen alternative or the choice model. For estimation purposes, the chosen alternative must always be included in $\mathcal{D}_n$.

The maximum likelihood estimation procedure, using the full choice set, amounts to solving the following optimization problem:

$$\max_{\theta} \sum_{n=1}^{N} \ln \Pr(i_n|\mathcal{C};\theta). \tag{1}$$

The estimator is consistent and asymptotically efficient, but complicated for large choice sets. We are considering instead the conditional maximum likelihood estimator

$$\max_{\theta} \sum_{n=1}^{N} \ln \Pr(i_n|\mathcal{D}_n,\mathcal{C};\theta), \tag{2}$$

which is consistent, but not asymptotically efficient[1]. In order to be useful, the terms in the sum must not depend on the full choice set, that is:

$$\Pr(i_n|\mathcal{D}_n, \mathcal{C}; \theta) = \Pr(i_n|\mathcal{D}_n; \theta).$$

In order to derive $\Pr(i_n|\mathcal{D}_n, \mathcal{C}; \theta)$, we rely on Bayes' theorem:

$$\Pr(i_n|\mathcal{D}_n, \mathcal{C}; \theta) = \frac{\Pr(\mathcal{D}_n|i_n)\Pr(i_n|\mathcal{C}; \theta)}{\sum_{j\in\mathcal{D}_n}\Pr(\mathcal{D}_n|j)\Pr(j|\mathcal{C}; \theta)}, \tag{3}$$

where the quantity $\Pr(\mathcal{D}_n|j)$ represents the probability that the sample of alternatives $\mathcal{D}_n$ has been generated, conditional on the fact that alternative j has been chosen by n.

A.1 Logit model

If we consider the logit model, we have

$$\Pr(i_n|\mathcal{C}; \theta) = \frac{e^{\mu V_{i_n,n}}}{\sum_{j\in\mathcal{C}} e^{\mu V_{jn}}} = \frac{e^{\mu V_{i_n,n}}}{\gamma_n},$$

where γ_n is the denominator. The point is that γ_n cancels out in (3), so that we obtain

$$\Pr(i_n|\mathcal{D}_n, \mathcal{C}; \theta) = \Pr(i_n|\mathcal{D}_n; \theta) = \frac{e^{\mu V_{i_n,n}+\ln(\Pr(\mathcal{D}_n|i_n))}}{\sum_{j\in\mathcal{D}_n} e^{\mu V_{jn}+\ln(\Pr(\mathcal{D}_n|j))}}, \tag{4}$$

which does not depend on $\mathcal{C}$ anymore. Concretely, it means that we simply need to estimate a logit model on the sampled set of alternatives, where the utility functions include the correction term $\ln(\Pr(\mathcal{D}_n|j))$.

A.2 MEV model

The derivation is similar for the MEV model, defined as

$$\Pr(i_n|\mathcal{C}; \theta) = \frac{e^{V_{i_n,n}+\ln G_{i_n}\left(e^{V_{1n}},\dots,e^{V_{Jn}}\right)}}{\sum_{j\in\mathcal{C}} e^{V_{jn}+\ln G_j\left(e^{V_{1n}},\dots,e^{V_{Jn}}\right)}} = \frac{e^{V_{i_n}+\ln G_{i_n}\left(e^{V_{1n}},\dots,e^{V_{Jn}}\right)}}{\gamma_n}.$$

where G is the MEV generating function, γ_n is the denominator, and G_i is its ith partial derivative. For example, for the nested logit model,

$$\ln G_i = \ln\mu + (\mu_m - 1)V_{in} + (\frac{\mu}{\mu_m} - 1)(\ln\sum_{j\in\mathcal{C}_m}\exp(\mu_m V_{jn})), \tag{5}$$

[1]One of the implications is that the Rao Cramer bound cannot be used to estimate the variance-covariance matrix. The robust estimator or bootstrapping must be used instead.

where $\mathcal{C}_m$ is the nest m. And for the cross-nested logit model,

$$G_i = \mu \sum_{m=1}^{M} \alpha_{im}^{\frac{\mu_m}{\mu}} \exp((\mu_m - 1)V_{in}) \left(\sum_{j \in \mathcal{C}} \alpha_{jm}^{\frac{\mu_m}{\mu}} \exp(\mu_m V_{jn}) \right)^{\frac{\mu}{\mu_m}-1}. \quad (6)$$

Again, γ_n cancels out in (3), so that we obtain

$$\Pr(i_n|\mathcal{D}_n, \mathcal{C}; \theta) = \frac{e^{V_{i_n,n} + \ln G_{i_n}\left(e^{V_{1n}},\ldots,e^{V_{Jn}}\right) + \ln(\Pr(\mathcal{D}_n|i_n))}}{\sum_{j \in \mathcal{D}_n} e^{V_{jn} + \ln G_j\left(e^{V_{1n}},\ldots,e^{V_{Jn}}\right) + \ln(\Pr(\mathcal{D}_n|j))}}. \quad (7)$$

Contrarily to the logit case, the expression above still involves the full choice set, which is needed to calculate G_i. Guevara and Ben-Akiva (2013) have proposed to approximate any sum of alternatives in the G_i by a sum involving only sampled alternatives. We therefore need another sample of alternatives. Note that the chosen alternative does not need to belong to it. We denote by $\mathcal{M}_n$ the set of alternatives sampled for this purpose. The index n indicates that this set varies across individuals. Then, the sums involved in the MEV terms can be approximated in the following way:

$$\sum_{j \in \mathcal{C}_m} \exp(\mu_m V_{jn}) \approx \sum_{j \in \mathcal{C}_m \cap \mathcal{M}} w_{jn} \exp(\mu_m V_{jn}), \quad (8)$$

in (5), or

$$\sum_{j \in \mathcal{C}} \alpha_{jm}^{\frac{\mu_m}{\mu}} \exp(\mu_m V_{jn}) \approx \sum_{j \in \mathcal{M}} w_{jn} \alpha_{jm}^{\frac{\mu_m}{\mu}} \exp(\mu_m V_{jn}) \quad (9)$$

in (6), where

$$w_{jn} = \frac{1}{\Pr(j)}, \quad (10)$$

where $\Pr(j)$ is the probability that alternative j is integrated in the sample.

We refer the reader to Ben-Akiva and Lerman (1985), Guevara and Ben-Akiva (2013), Bierlaire and Krueger (2020) for more details about the theoretical background.

B Implementation

For the implementation of the MEV models, we need to deal with the fact that there are two numbering of alternatives:

- the numbering associated with the original choice set,

- a specific numbering for each sample.

We refer to the "number" of the alternative in the original choice set as its "identifier", or "ID". We use the letter i to refer to them. We refer to the number of the alternative in the sampled choice set as its "index", and we use the letters j for the main sample, and k for the MEV sample. For observation n, the identifier of alternative with index j is denoted by i_{jn}.

B.1 Nested logit

Consider nest $\mathcal{C}_m$, with parameter μ_m. The calculation of (8) is implemented as

$$\gamma_m = \sum_{k \in \mathcal{M}} \Delta(i_{kn} \in \mathcal{C}_m) w_{kn} \exp(\mu_m V_{kn}),$$

where $\Delta(i_{kn} \in \mathcal{C}_m)$ is 1 if $i_{kn} \in \mathcal{C}_m$ and 0 otherwise. Then, the calculation of the MEV term for alternative $j \in \mathcal{D}$ is

$$\ln G_j = \ln \mu + (\mu_m - 1) V_{jn} + \sum_m \Delta(i_{jn} \in \mathcal{C}_m) (\frac{\mu}{\mu_m} - 1) \ln \gamma_m.$$

B.2 Cross Nested logit

The difficulty with the cross nested logit model is the handling of the nest membership parameters α, that are associated with a nest and an alternative from the original choice set. We assume that the α parameters are fixed and available in the data set.

The calculation of (9) is implemented as

$$\gamma_m = \sum_{k \in \mathcal{M}} \Delta(\alpha_{km} \neq 0) w_{kn} \alpha_{km}^{\frac{\mu_m}{\mu}} \exp(\mu_m V_{kn}).$$

Note that the factor $\Delta(\alpha_{km} \neq 0)$ is redundant from a mathematical point of view, but is designed to speed up the calculation. Then, the calculation of the MEV term for alternative $j \in \mathcal{D}$ is

$$G_j = \mu \sum_{m=1}^{M} \Delta(\alpha_{jm} \neq 0) \alpha_{jm}^{\frac{\mu_m}{\mu}} \exp((\mu_m - 1) V_{jn}) \gamma_m^{\frac{\mu}{\mu_m} - 1}.$$

Finally, we shift each utility by $\ln G_j$.

C Sampling protocol

For the set $\mathcal{D}_n$, the Biogeme implementation assumes the following sampling procedure:

1. The full choice set is partitioned into K segments of size R_k: $J = \sum_{k=1}^{K} R_k$.
2. Let r_k be the number of alternatives to be sampled in each segment, so that the size of $\mathcal{D}_n$: $\sum_{k=1}^{K} r_k$.
3. Denote $k(i)$ the segment containing the chosen alternative i.
4. Randomly draw $r_{k(i)} - 1$ alternatives among the non chosen ones in segment $k(i)$, and add i to obtain $\mathcal{D}_{k(i)}$.
5. Randomly draw r_k alternatives in each segment k, $k \neq k(i)$ to obtain $\mathcal{D}_k$.
6. The sample is composed of the chosen alternative and all draws: $\mathcal{D} = \cup_k \mathcal{D}_k$.

We can therefore calculate:

$$\begin{aligned}
\Pr(\mathcal{D}|i) &= \left(\begin{array}{c} R_{k(i)} - 1 \\ r_{k(i)} - 1 \end{array} \right) \prod_{k=2}^{K} \left(\begin{array}{c} R_k \\ r_k \end{array} \right) \\
&= \frac{R_{k(i)}}{r_{k(i)}} \prod_{k=1}^{K} \left(\begin{array}{c} R_k \\ r_k \end{array} \right) \\
&= \frac{1}{\Pr(i)} \Pr(\mathcal{D}).
\end{aligned}$$

Note that the quantity $\Pr(\mathcal{D})$ is a constant, that cancels out in all the expressions above. Therefore, instead of using $\Pr(\mathcal{D}|i)$ as a correction term in (4) and (7), it is equivalent to use

$$\frac{R_{k(i)}}{r_{k(i)}} \propto \Pr(\mathcal{D}|i), \tag{11}$$

which is the number of alternatives in the segment containing alternative i, divided by the number of alternatives sampled from this segment, that is the inverse of the probability to sample alternative i. We can also write

$$\ln \Pr(\mathcal{D}|i) = \frac{1}{\Pr(i)} + K = \ln R_{k(i)} - \ln r_{k(i)} + K, \tag{12}$$

where K is a constant independent from i that cancels out in (4) and (7). The correction term calculated by Biogeme is the opposite of this term:

$$\ln \Pr(i) = \ln r_{k(i)} - \ln R_{k(i)}. \tag{13}$$

Note that it is always negative.

For the set M_n, the Biogeme implementation is the same as above, with two important differences:

1. the chosen alternative does not play any role in the sampling procedure,
2. it is not necessary to partition the full choice set. Alternatives that are alone in a nest do not contribute to the calculation of the MEV terms, and can therefore be excluded.

The procedure works as follows:

1. The set of alternatives involved in nests is partitioned into K segments of size R_k: $J = \sum_{k=1}^{K} R_k$.
2. Let r_k be the number of alternatives to be sampled in each segment, so that the size of $\mathcal{M}_n$: $\sum_{k=1}^{K} r_k$.
3. Randomly draw r_k alternatives in each segment k to obtain $\mathcal{M}_k$.
4. The sample is composed of all draws: $\mathcal{M}_n = \cup_k \mathcal{M}_k$.

We can therefore calculate (10) as:

$$w_{jn} = \frac{1}{\Pr(j)} = \frac{R_{k(j)}}{r_{k(j)}}. \tag{14}$$

D Generated data file

We denote

- J the number of sampled alternatives, that is the cardinality of $\mathcal{D}_n$,
- M the number of sampled alternatives for the nest, that is the cardinality of $\mathcal{M}$,
- K_o the number of columns in the observed choices data file,
- K_a the number of columns in the alternatives data file,
- K_c the number of combined variables,

- L the number of nests in the cross-nested logit model.

The columns of the generated data files are:

- The K_o columns of the data file containing the observed choices, copied as such.
- For $j = 0, \dots J-1$, where $j = 0$ always corresponds to the chosen alternative, we have $K_a + 1$ columns, for a total of $J(K_a + 1)$ columns:
  - The K_a columns of the alternatives data file, where the name of the column is appended with a suffix `_j`. For instance, if $j = 11$, `category_Japanese` is labeled `category_Japanese_11`.
  - A column labeled `_log_proba_j` containing the correction term (13).
- For $k = 0, \dots M-1$, we have $K_a + L + 1$ columns, for a total of $M(K_a + L + 1)$ columns:
  - The K_a columns of the alternatives data file, where the name of the column is appended with a prefix `MEV_` and a suffix `_k`. For instance, if $k = 7$, `price_` is labeled `MEV_price_7`.
  - For each of the L nests of the cross-nested logit model, the alpha parameter for alternative k. The name of the corresponding column is the name of the nest, with a prefix `MEV_CNL_` and a suffix `_k`. For instance, if $k = 6$, and the nest is `asian`, the name of the column is `MEV_CNL_asian_6`.
  - A column labeled `MEV__mev_weight_k` containing the correction factor (14).
- For $j = 0, \dots J-1$, where $j = 0$ always corresponds to the chosen alternative, we have K_c columns, for a total of JK_c. The name of each column is the name of the combined variable with a suffix `_j`. For instance, if $j = 4$, we have `log_dist_4`.
- For $k = 0, \dots M-1$, we have K_c columns, for a total of MK_c. The name of each column is the name of the combined variable with a prefix `MEV_` and a suffix `_k`. For instance, if $k = 4$, we have `log_dist_4`.

The total number of columns in the generated file is therefore

$$K_o + J(K_a + 1) + M(K_a + L + 1) + JK_c + MK_c$$

or

$$K_o + J(K_a + K_c + 1) + M(K_a + K_c + L + 1).$$

For the logit model described in Section 4, we have $K_o = 17$, $J = 10$, $M = 0$, $K_a = 16$, $K_c = 1$, $L = 0$, for a total of 197 columns.

For the nested logit model described in Section 5, we have $K_o = 17$, $J = 20$, $M = 33$, $K_a = 16$, $K_c = 1$, $L = 0$, for a total of 971 columns.

For the cross-nested logit model described in Section 6, we have $K_o = 17$, $J = 10$, $M = 63$, $K_a = 16$, $K_c = 1$, $L = 2$, for a total of 1457 columns.

E Python codes

E.1 Sets of alternatives

```
"""

List of alternatives
====================

Script reading the list of alternatives and identifying subsets

:author: Michel Bierlaire
:date: Mon Oct  9 10:53:03 2023
"""

import pandas as pd
from biogeme.partition import Partition

# %%
alternatives = pd.read_csv('restaurants.dat')
alternatives

# %%
ID_COLUMN = 'ID'

# %%
all_alternatives = set(list(alternatives[ID_COLUMN]))

# %%
# Set of Asian restaurants
asian = set(alternatives[alternatives['Asian'] == 1][ID_COLUMN])
print(f'Number of asian restaurants: {len(asian)}')

# %%
# Set of restaurants located in downtown
downtown = set(alternatives[alternatives['downtown'] ==
    1][ID_COLUMN])

# %%
```

```python
# Set of Asian restaurants in downtown
asian_and_downtown = asian & downtown

# %%
# Set of Asian restaurants, and of restaurants in downtown
asian_or_downtown = asian | downtown

# %%
# Set of Asian restaurants not in downtown
only_asian = asian - asian_and_downtown

# %%
# Set of non Asian restaurants in downtown
only_downtown = downtown - asian_and_downtown

# %%
# Set of restaurants that are neither Asian nor in downtown
others = all_alternatives - asian_or_downtown


# %%
def complement(a_set: set[int]) -> set[int]:
    """Returns the complement of a set"""
    return all_alternatives - a_set


# %%
# Partitions.
partition_asian = Partition([asian, complement(asian)], full_set=all_alternatives)
partition_downtown = Partition(
    [downtown, complement(downtown)], full_set=all_alternatives
)
partition_uniform = Partition([all_alternatives], full_set=all_alternatives)
partition_uniform_asian = Partition([asian], full_set=asian)
partition_uniform_asian_or_downtown = Partition(
    [asian_or_downtown], full_set=asian_or_downtown
)

# %%
partitions = {
    'uniform': partition_uniform,
    'asian': partition_asian,
    'downtown': partition_downtown,
    'uniform_asian_or_downtown': partition_uniform_asian_or_downtown,
    'uniform_asian': partition_uniform_asian,
}
```

E.2 Model specification

```
"""

Model specification
===================

Script containing the model specification.

:author: Michel Bierlaire
:date: Wed Nov  1 17:37:33 2023
"""
from biogeme.expressions import Beta, Variable, log
from biogeme.sampling_of_alternatives import CrossVariableTuple

# %%
# Variable combining attributes of the alternatives and
# characteristics of the decision-maker
combined_variables = [
    CrossVariableTuple(
        'log_dist',
        log(
            (
                (Variable('user_lat') - Variable('rest_lat'))
                ** 2
                + (Variable('user_lon') - Variable('rest_lon'))
                ** 2
            )
            ** 0.5
        ),
    )
]

# %%
# Parameters to estimate.
beta_rating = Beta('beta_rating', 0, None, None, 0)
beta_price = Beta('beta_price', 0, None, None, 0)
beta_chinese = Beta('beta_chinese', 0, None, None, 0)
beta_japanese = Beta('beta_japanese', 0, None, None, 0)
beta_korean = Beta('beta_korean', 0, None, None, 0)
beta_indian = Beta('beta_indian', 0, None, None, 0)
beta_french = Beta('beta_french', 0, None, None, 0)
beta_mexican = Beta('beta_mexican', 0, None, None, 0)
beta_lebanese = Beta('beta_lebanese', 0, None, None, 0)
beta_ethiopian = Beta('beta_ethiopian', 0, None, None, 0)
beta_log_dist = Beta('beta_log_dist', 0, None, None, 0)

# %%
# Utility function.
```

```
V = (
    beta_rating * Variable('rating')
    + beta_price * Variable('price')
    + beta_chinese * Variable('category_Chinese')
    + beta_japanese * Variable('category_Japanese')
    + beta_korean * Variable('category_Korean')
    + beta_indian * Variable('category_Indian')
    + beta_french * Variable('category_French')
    + beta_mexican * Variable('category_Mexican')
    + beta_lebanese * Variable('category_Lebanese')
    + beta_ethiopian * Variable('category_Ethiopian')
    + beta_log_dist * Variable('log_dist')
)
```

E.3 Estimation of the logit model

```
"""

Logit
=====

Estimation of a logit model using sampling of alternatives.

:author: Michel Bierlaire
:date: Wed Nov  1 17:39:47 2023
"""

import pandas as pd
from biogeme.sampling_of_alternatives import (
    SamplingContext,
    ChoiceSetsGeneration,
    GenerateModel,
    generate_segment_size,
)
import biogeme.biogeme_logging as blog
import biogeme.biogeme as bio
from compare import compare
from specification import V, combined_variables
from alternatives import (
    alternatives,
    ID_COLUMN,
    partitions,
)

# %%
logger = blog.get_screen_logger(level=blog.INFO)

# %%
# The data file contains several columns associated with
```

```
    synthetic
# choices. Here we arbitrarily select 'logit_4'.
CHOICE_COLUMN = 'logit_4'

# %%
SAMPLE_SIZE = 10
PARTITION = 'asian'
MODEL_NAME = f'logit_{PARTITION}_{SAMPLE_SIZE}_alt'
FILE_NAME = f'{MODEL_NAME}.dat'
OBS_FILE = 'obs_choice.dat'

# %%
the_partition = partitions.get(PARTITION)
if the_partition is None:
    raise ValueError(f'Unknown partition: {PARTITION}')

# %%
segment_sizes = generate_segment_size(SAMPLE_SIZE,
    the_partition.number_of_segments())

# %%
observations = pd.read_csv(OBS_FILE)

# %%
context = SamplingContext(
    the_partition=the_partition,
    sample_sizes=segment_sizes,
    individuals=observations,
    choice_column=CHOICE_COLUMN,
    alternatives=alternatives,
    id_column=ID_COLUMN,
    biogeme_file_name=FILE_NAME,
    utility_function=V,
    combined_variables=combined_variables,
)

# %%
logger.info(context.reporting())

# %%
the_data_generation = ChoiceSetsGeneration(context=context)

# %%
the_model_generation = GenerateModel(context=context)

# %%
biogeme_database =
    the_data_generation.sample_and_merge(recycle=False)

```

```
# %%
logprob = the_model_generation.get_logit()

# %%
the_biogeme = bio.BIOGEME(biogeme_database, logprob)
the_biogeme.modelName = MODEL_NAME

# %%
# Calculate the null log likelihood for reporting.
the_biogeme.calculateNullLoglikelihood({i: 1 for i in
    range(SAMPLE_SIZE)})

# %%
# Estimate the parameters
results = the_biogeme.estimate(recycle=False)

# %%
print(results.short_summary())

# %%
estimated_parameters = results.getEstimatedParameters()
estimated_parameters

# %%
df, msg = compare(estimated_parameters)

# %%
print(df)

# %%
print(msg)
```

E.4 Estimation of the nested logit model

```
"""

Nested logit
============

Estimation of a nested logit model using sampling of
    alternatives.

:author: Michel Bierlaire
:date: Wed Nov  1 18:00:15 2023
"""
import pandas as pd
from biogeme.sampling_of_alternatives import (
    SamplingContext,
    ChoiceSetsGeneration,
```

```
    GenerateModel,
    generate_segment_size,
)
from biogeme.expressions import Beta
from biogeme.nests import OneNestForNestedLogit,
    NestsForNestedLogit
import biogeme.biogeme_logging as blog
import biogeme.biogeme as bio
from specification import V, combined_variables
from compare import compare
from alternatives import (
    alternatives,
    ID_COLUMN,
    partitions,
    asian,
    all_alternatives,
)

# %%
logger = blog.get_screen_logger(level=blog.INFO)

# %%
SAMPLE_SIZE = 20  # out of 100
SAMPLE_SIZE_MEV = 33  # out of 33
CHOICE_COLUMN = 'nested_0'
PARTITION = 'downtown'
MEV_PARTITION = 'uniform_asian'
MODEL_NAME = f'nested_{PARTITION}_{SAMPLE_SIZE}'
FILE_NAME = f'{MODEL_NAME}.dat'

# %%
the_partition = partitions.get(PARTITION)
if the_partition is None:
    raise ValueError(f'Unknown partition: {PARTITION}')

# %%
segment_sizes = generate_segment_size(SAMPLE_SIZE,
    the_partition.number_of_segments())

# %%
# We use all alternatives in the nest.
mev_partition = partitions.get(MEV_PARTITION)
if mev_partition is None:
    raise ValueError(f'Unknown partition: {MEV_PARTITION}')
mev_segment_sizes = [SAMPLE_SIZE_MEV]

# %%
observations = pd.read_csv('obs_choice.dat')

```

```
# %%
context = SamplingContext(
    the_partition=the_partition,
    sample_sizes=segment_sizes,
    individuals=observations,
    choice_column=CHOICE_COLUMN,
    alternatives=alternatives,
    id_column=ID_COLUMN,
    biogeme_file_name=FILE_NAME,
    utility_function=V,
    combined_variables=combined_variables,
    mev_partition=mev_partition,
    mev_sample_sizes=mev_segment_sizes,
)

# %%
logger.info(context.reporting())

# %%
the_data_generation = ChoiceSetsGeneration(context=context)
the_model_generation = GenerateModel(context=context)

# %%
biogeme_database =
    the_data_generation.sample_and_merge(recycle=False)

# %%
# Definition of the nest.
mu_asian = Beta('mu_asian', 1.0, 1.0, None, 0)
nest_asian = OneNestForNestedLogit(
    nest_param=mu_asian, list_of_alternatives=asian,
        name='asian'
)
nests = NestsForNestedLogit(
    choice_set=all_alternatives,
    tuple_of_nests=(nest_asian,),
)

# %%
logprob = the_model_generation.get_nested_logit(nests)

# %%
the_biogeme = bio.BIOGEME(biogeme_database, logprob)
the_biogeme.modelName = MODEL_NAME

# %%
# Calculate the null log likelihood for reporting.
the_biogeme.calculateNullLoglikelihood({i: 1 for i in
    range(context.total_sample_size)})
```

```
108
109 # %%
110 # Estimate the parameters
111 results = the_biogeme.estimate(recycle=False)
112
113 # %%
114 print(results.short_summary())
115
116 # %%
117 estimated_parameters = results.getEstimatedParameters()
118 estimated_parameters
119
120 # %%
121 df, msg = compare(estimated_parameters)
122
123 # %%
124 print(df)
125
126 # %%
127 print(msg)
```

E.5 Estimation of the cross-nested logit model

```
1  """
2
3  Cross-nested logit
4  ==================
5
6  Estimation of a cross-nested logit model using sampling of
       alternatives.
7
8  :author: Michel Bierlaire
9  :date: Wed Nov  1 18:00:33 2023
10 """
11 import pandas as pd
12 from biogeme.sampling_of_alternatives import (
13     SamplingContext,
14     ChoiceSetsGeneration,
15     GenerateModel,
16     generate_segment_size,
17 )
18 from biogeme.expressions import Beta
19 import biogeme.biogeme_logging as blog
20 import biogeme.biogeme as bio
21 from biogeme.nests import OneNestForCrossNestedLogit,
       NestsForCrossNestedLogit
22 from specification import V, combined_variables
23 from compare import compare
24 from alternatives import (
```

```
    alternatives,
    ID_COLUMN,
    partitions,
    all_alternatives,
    asian_and_downtown,
    only_downtown,
    only_asian,
)

# %%
logger = blog.get_screen_logger(level=blog.INFO)

# %%
PARTITION = 'downtown'
MEV_PARTITION = 'uniform_asian_or_downtown'
SAMPLE_SIZE = 10  # out of 100 alternatives
SAMPLE_SIZE_MEV = 63  # out of 63 alternatives
CHOICE_COLUMN = 'cnl_3'
MODEL_NAME = f'cnl_{SAMPLE_SIZE}_{SAMPLE_SIZE_MEV}'
FILE_NAME = f'{MODEL_NAME}.dat'

# %%
the_partition = partitions.get(PARTITION)
if the_partition is None:
    raise ValueError(f'Unknown partition: {PARTITION}')

# %%
segment_sizes = list(
    generate_segment_size(SAMPLE_SIZE,
        the_partition.number_of_segments())
)

# %%
# We use all alternatives in the nest.
mev_partition = partitions.get(MEV_PARTITION)
if mev_partition is None:
    raise ValueError(f'Unknown partition: {MEV_PARTITION}')
mev_segment_sizes = [
    SAMPLE_SIZE_MEV,
]

# %%
# Nests

# %%
# Downtown
mu_downtown = Beta('mu_downtown', 1, 1, None, 0)
downtown_alpha_dict = {i: 0.5 for i in asian_and_downtown} | {
    i: 1 for i in only_downtown
```

```
}
downtown_nest = OneNestForCrossNestedLogit(
    nest_param=mu_downtown, dict_of_alpha=downtown_alpha_dict,
        name='downtown'
)

# %%
# Asian
mu_asian = Beta('mu_asian', 1, 1, None, 0)
asian_alpha_dict = {i: 0.5 for i in asian_and_downtown} | {i:
    1.0 for i in only_asian}
asian_nest = OneNestForCrossNestedLogit(
    nest_param=mu_asian, dict_of_alpha=asian_alpha_dict,
        name='asian'
)

cnl_nests = NestsForCrossNestedLogit(
    choice_set=all_alternatives,
    tuple_of_nests=(downtown_nest, asian_nest),
)

# %%
observations = pd.read_csv('obs_choice.dat')

# %%
context = SamplingContext(
    the_partition=the_partition,
    sample_sizes=segment_sizes,
    individuals=observations,
    choice_column=CHOICE_COLUMN,
    alternatives=alternatives,
    id_column=ID_COLUMN,
    biogeme_file_name=FILE_NAME,
    utility_function=V,
    combined_variables=combined_variables,
    mev_partition=mev_partition,
    mev_sample_sizes=mev_segment_sizes,
    cnl_nests=cnl_nests,
)

# %%
logger.info(context.reporting())

# %%
the_data_generation = ChoiceSetsGeneration(context=context)
the_model_generation = GenerateModel(context=context)

# %%
biogeme_database =
```

```
    the_data_generation.sample_and_merge(recycle=False)

# %%
logprob = the_model_generation.get_cross_nested_logit()

# %%
the_biogeme = bio.BIOGEME(biogeme_database, logprob)
the_biogeme.modelName = MODEL_NAME

# %%
# Calculate the null log likelihood for reporting.
the_biogeme.calculateNullLoglikelihood({i: 1 for i in
    range(context.total_sample_size)})

# %%
# Estimate the parameters.
results = the_biogeme.estimate(recycle=False)

# %%
print(results.short_summary())

# %%
estimated_parameters = results.getEstimatedParameters()
estimated_parameters

# %%
df, msg = compare(estimated_parameters)

# %%
print(df)

# %%
print(msg)
```