Monte-Carlo integration with PandasBiogeme

Michel Bierlaire

December 31, 2019

Report TRANSP-OR 191231
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
transp-or.epfl.ch

Series on Biogeme

Biogeme is an open source Python package designed for the maximum likelihood estimation of parametric models in general, with a special emphasis on discrete choice models. It relies on the package Python Data Analysis Library called Pandas.

In this document, we investigate some aspects related to Monte-Carlo integration, which is particularly useful when estimating mixtures choice models, as well as choice models with latent variables. We assume that the reader is already familiar with discrete choice models, with PandasBiogeme, and with simulation methods, although a short summary is provided. This document has been written using PandasBiogeme 3.2.5.

1 Monte-Carlo integration

Monte-Carlo integration consists in approximating an integral with the sum of a large number of terms. It comes from the definition of the expectation of a continuous random variable. Consider the random variable X with probability density function (pdf) $f_X(x)$. Assuming that X can take any value in the interval $[a, b]$, where $a \in \mathbb{R} \cup \{-\infty\}$ and $b \in \mathbb{R} \cup \{+\infty\}$, the expected value of X is given by

$$\mathrm{E}[X] = \int_a^b x f_X(x) dx. \tag{1}$$

Also, if $g : \mathbb{R} \to \mathbb{R}$ is a function, then

$$\mathrm{E}[g(X)] = \int_a^b g(x) f_X(x) dx. \tag{2}$$

The expectation of a random variable can be approximated using simulation. The idea is simple: generate a sample of realizations of X, that is generate R draws x_r, $r = 1, \ldots, R$ from X, and calculate the sample mean:

$$\mathrm{E}[g(X)] \approx \frac{1}{R} \sum_{r=1}^{R} g(x_r). \tag{3}$$

Putting (2) and (3) together, we obtain an approximation of the integral:

$$\int_a^b g(x) f_X(x) dx \approx \frac{1}{R} \sum_{r=1}^{R} g(x_r). \tag{4}$$

Also, we have

$$\int_a^b g(x) f_X(x) dx = \lim_{R \to \infty} \frac{1}{R} \sum_{r=1}^{R} g(x_r). \tag{5}$$

Therefore, the procedure to calculate the following integral

$$I = \int_a^b g(x)dx \tag{6}$$

is the following:

1. select a random variable X defined on the interval $[a, b]$ such that you can generate realizations of X, and such that the pdf f_X is known,
2. generate R draws x_r, $r = 1, \ldots, R$ from X,
3. calculate

$$I \approx \frac{1}{R} \sum_{r=1}^{R} \frac{g(x_r)}{f_X(x_r)}. \tag{7}$$

In order to obtain an estimate of the approximation error, we must calculate the variance of the random variable. The sample variance is an unbiased estimate of the true variance:

$$V_R = \frac{1}{R-1} \sum_{r=1}^{R} (\frac{g(x_r)}{f_X(x_r)} - I)^2. \tag{8}$$

Alternatively as

$$\mathrm{Var}[g(X)] = \mathrm{E}[g(X)^2] - \mathrm{E}[g(x)]^2, \tag{9}$$

the variance can be approximated by simulation as well:

$$V_R \approx \frac{1}{R} \sum_{r=1}^{R} \frac{g(x_r)^2}{f_X(x_r)} - I^2. \tag{10}$$

Note that, for the values of R that we are using in this document, dividing by R or by $R-1$ does not make much difference in practice. The approximation error is then estimated as

$$e_R = \sqrt{\frac{V_r}{R}}. \tag{11}$$

We refer the reader to Ross (2012) for a comprehensive introduction to simulation methods.

2 Uniform draws

There are many algorithms to draw from various distributions. All of them require at some point draws from the uniform distribution. There are several techniques that generate such uniform draws.

Each programming language provides a routine to draw a random number between 0 and 1. Such routines are deterministic, but the sequences of numbers that they generate share many properties with sequences of random numbers. Therefore, they are often called "pseudo random numbers".

Researchers have proposed to use other types of sequences to perform Monte-Carlo integration, called "quasi-random sequences" or "low-discrepancy sequences". PandasBiogeme implements the Halton draws, from Halton (1960). They have been reported to perform well for discrete choice models (Train, 2000, Bhat, 2001, Bhat, 2003, Sándor and Train, 2004).

An Halton sequence is defined by a prime number p, say $p = 3$, in the following way:

- divide the interval [0,1] into p parts to obtain the first $p - 1$ elements: 1/3 and 2/3, for our example;
- consider each new interval, and divide it again into p parts: [0,1/3] generates 1/9 and 2/9, [1/3,2/3] generates 4/9 and 5/9, and [2/3,1] generates 7/9 and 8/9;
- divide again each new interval into p parts, etc.

Note that Train (2000) suggests that the first 10 elements of the sequence are discarded, since the early elements have a tendency to be correlated over Halton sequences with different primes.

The third method to generate uniform random numbers implemented in PandasBiogeme is called "Modified Latin Hypercube Sampling", and has been proposed by Hess et al. (2006). The general idea is to generate a randomly perturbed uniform grid on the unit interval.

In the following, we are using these three options, and compare the accuracy of the corresponding Monte-Carlo integration.

3 Illustration with PandasBiogeme

We first illustrate the method on a simple integral. Consider

$$I = \int_0^1 e^x dx. \tag{12}$$

In this case, it can be solved analytically:

$$I = e - 1 = 1.718281828459045. \tag{13}$$

In order to use Monte-Carlo integration, we consider the random variable X that is uniformly distributed on $[0, 1]$, so that

$$f_X(x) = \begin{cases} 1 & \text{if } x \in [0, 1], \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Therefore, we can approximate I by generating R draws from X and

$$I = E[e^X] \approx \frac{1}{R} \sum_{r=1}^{R} \frac{e^{x_r}}{f_X(x_r)} = \frac{1}{R} \sum_{r=1}^{R} e^{x_r}. \tag{15}$$

Moreover, as

$$\begin{aligned} \mathrm{Var}[e^X] &= E[e^{2X}] - E[e^X]^2 \\ &= \int_0^1 e^{2x} dx - (e-1)^2 \\ &= (e^2 - 1)/2 - (e-1)^2 \\ &= 0.2420356075, \end{aligned} \tag{16}$$

the standard error is 0.0034787613 for $R = 20'000$, and 0.0011000809 for $R = 200'000$. These theoretical values are estimated also below using PandasBiogeme.

We use PandasBiogeme to calculate (15). Note that PandasBiogeme requires a database, which is not necessary in this simplistic case. We use the simulation mode of PandasBiogeme. It generates output for each row of the data file. In our case, we just need one output, so that we take any database. For this specific example, the data included in the database is irrelevant.

```
pandas = pd.DataFrame()
pandas['FakeColumn'] = [1.0]
database = db.Database("fakeDatabase",pandas)
```

The generation of draws in PandasBiogeme is performed using the command bioDraws('U','UNIFORM'), where the first argument, 'U', provides the name of the random variable associated with the draws, and the second argument, 'UNIFORM', provides the distribution of the random variable. Note that the valid keywords for the type of draws are

- UNIFORM: pseudo-random draws from a uniform distribution U[0,1],
- UNIFORM_ANTI: antithetic pseudo-random draws from a uniform distribution U[0,1],

- UNIFORM_HALTON2: Halton draws with base 2, skipping the first 10,
- UNIFORM_HALTON3: Halton draws with base 3, skipping the first 10,
- UNIFORM_HALTON5: Halton draws with base 5, skipping the first 10,
- UNIFORM_MLHS: Modified Latin Hypercube Sampling on [0,1],
- UNIFORM_MLHS_ANTI: Antithetic Modified Latin Hypercube Sampling on [0,1],
- UNIFORMSYM: Uniform U[-1,1],
- UNIFORMSYM_ANTI: Antithetic uniform U[-1,1],
- UNIFORMSYM_HALTON2: Halton draws on [-1,1] with base 2, skipping the first 10,
- UNIFORMSYM_HALTON3: Halton draws on [-1,1] with base 3, skipping the first 10,
- UNIFORMSYM_HALTON5: Halton draws on [-1,1] with base 5, skipping the first 10,
- UNIFORMSYM_MLHS: Modified Latin Hypercube Sampling on [-1,1],
- UNIFORMSYM_MLHS_ANTI: Antithetic Modified Latin Hypercube Sampling on [-1,1],
- NORMAL: Normal N(0,1) draws,
- NORMAL_ANTI: Antithetic normal draws,
- NORMAL_HALTON2: Normal draws from Halton base 2 sequence,
- NORMAL_HALTON3: Normal draws from Halton base 3 sequence,
- NORMAL_HALTON5: Normal draws from Halton base 5 sequence,
- NORMAL_MLHS: Normal draws from Modified Latin Hypercube Sampling,
- NORMAL_MLHS_ANTI: Antithetic normal draws from Modified Latin Hypercube Sampling

The integrand is defined by the following statement:

```
integrand = exp(bioDraws('U','UNIFORM'))
```

and the Monte-Carlo integration is obtained as follows:

```
simulatedI = MonteCarlo(integrand)
```

The number of draws is defined by the parameter NbrOfDraws:

```
R = 2000
biogeme = bio.BIOGEME(database,simulate,numberOfDraws=R)
```

We calculate as well the simulated variance, using (10):

```
sampleVariance = \
  MonteCarlo(integrand*integrand) - simulatedI * simulatedI
```

and the standard error (11):

```
stderr = (sampleVariance / R)**0.5
```

Also, as we know the true value of the integral

```
trueI = exp(1.0) - 1.0
```

we can calculate the error:

```
error = simulatedI - trueI
```

The calculation is obtained using the following statements:

```
simulate = {'Analytical Integral': trueI,
            'Simulated Integral': simulatedI,
            'Sample variance   ': sampleVariance,
            'Std Error         ': stderr,
            'Error             ': error}

biogeme = bio.BIOGEME(database,simulate,numberOfDraws=R)
biogeme.modelName = f'01simpleIntegral_{R}'
results = biogeme.simulate()
```

We obtain the following results:

Analytical Integral:	1.718282,
Simulated Integral:	1.695553,
Sample variance:	0.230498,
Std Error:	0.010735,
Error:	-0.022728.

Remember that the true variance is 0.2420356075, and the true standard error for 2'000 draws is error is 0.011000809231598. If we use ten times more draws, that is 20'000 draws, we obtain a more precise value:

Analytical Integral:	1.718282,
Simulated Integral:	1.721436,
Sample variance:	0.240849,
Std Error:	0.010974,
Error:	0.003155.

Remember that the true variance is 0.2420356075, and the true standard error for R = 20'000 is 0.003478761327685. The complete specification file for PandasBiogeme is available in Appendix A.1.

We now illustrate the Monte-Carlo integration using Halton draws and MLHS draws. By default, PandasBiogeme provides Halton sequences for bases 2, 3 and 5. But the user can also specify her own draws, by implementing a Python function that takes two arguments: the sample size, that is, the number of observations, and the number of draws to generate per observation. We illustrate this with the Halton sequence for base 13. We implement a function with the requested arguments, wrapping the Halton generation function provided in the draws module.

```
import biogeme.draws as draws
def halton13(sampleSize,numberOfDraws):
    return draws.getHaltonDraws(sampleSize,
                                numberOfDraws,
                                base=13,
                                skip=10)
```

This function needs next to be associated with a keyword, and a short description.

```
mydraws = {'HALTON13':(halton13,'Halton draws, base 13, skipping 10')}
```

And this must be communicated to the database.

```
database.setRandomNumberGenerators(mydraws)
```

We recompute the integral using uniform draws, Halton draws in base 2, Halton draws in base 13, and MLHS draws, using the following statements:

```
integrand = exp(bioDraws('U','UNIFORM'))
integrand_halton = exp(bioDraws('U_halton','UNIFORM_HALTON2'))
integrand_halton13 = exp(bioDraws('U_halton13','HALTON13'))
integrand_mlhs = exp(bioDraws('U_mlhs','UNIFORM_MLHS'))
```

The results are reported in Table 1.

	Uniform	Halton	Halton13	MLHS
Simulated	1.71612	1.71815	1.718	1.71828
Sample var.	0.239978	0.241995	0.241998	0.242035
Std error	0.00346394	0.00347847	0.00347849	0.00347876
Error	-0.00216237	-0.000132989	-0.000286219	9.41272e-08

Table 1: Estimation with different types of draws, with 20'000 draws

The complete specification file for PandasBiogeme is available in Appendix A.2.

4 Variance reduction: antithetic draws

There are several techniques to reduce the variance of the draws used for the Monte-Carlo integration. Reducing the variance improves the precision of the approximation for the same number of draws. Equivalently, they allow to use less draws to achieve the same precision. In this document, we introduce the concept of antithetic draws. As the focus of this document is on PandasBiogeme, we urge the reader to read an introduction to variance reduction methods in simulation, for instance in Ross (2012).

Instead of drawing from X, consider two random variables X_1 and X_2, identically distributed with pdf $f_X = f_{X_1} = f_{X_2}$, and define a new random variable

$$Y = \frac{X_1 + X_2}{2}. \tag{17}$$

Then, as $E[Y] = E[X_1] = E[X_2] = E[X]$, we can rewrite (1) as follows:

$$E[Y] = \frac{1}{2}E[X_1] + \frac{1}{2}E[X_2] = E[X] = \int_a^b x f_X(x) dx. \tag{18}$$

The variance of this quantity is

$$\mathrm{Var}[Y] = \frac{1}{4}(\mathrm{Var}(X_1) + \mathrm{Var}(X_2) + 2\,\mathrm{Cov}(X_1, X_2)). \tag{19}$$

If X_1 and X_2 are independent, this variance is equal to

$$\mathrm{Var}[Y] = \frac{1}{2}\mathrm{Var}[X]. \tag{20}$$

Therefore, using Y for Monte-Carlo integration is associated with a variance divided by two, but requires twice more draws (R draws for X_1 and R draws for X_2). It has no advantage on drawing directly R draws from X. Formally, we can compare the standard errors of the two methods for the same number of draws. Drawing $2R$ draws from X, we obtain the following standard error:

$$\sqrt{\frac{\mathrm{Var}[X]}{2R}}. \tag{21}$$

Drawing R draws from X_1 and R draws from X_2 to generate R draws from Y, we obtain the same standard error

$$\sqrt{\frac{\mathrm{Var}[Y]}{R}} = \sqrt{\frac{\mathrm{Var}[X]}{2R}}. \tag{22}$$

However, if the variables X_1 and X_2 happen to be negatively correlated, that is if $\mathrm{Cov}(X_1, X_2) < 0$, then $\mathrm{Var}[Y] < \mathrm{Var}[X]/2$, and drawing from Y

reduces the standard error. For instance, if X_1 is uniformly distributed on $[0,1]$, then $X_2 = 1 - X_1$ is also uniformly distributed on $[0,1]$, and

$$\mathrm{Cov}(X_1, X_2) = \mathrm{E}[X_1(1-X_1)] - \mathrm{E}[X_1]\,\mathrm{E}[1-X_1] = -\frac{1}{12} < 0. \tag{23}$$

If X_1 has a standard normal distribution, that is such that $\mathrm{E}[X_1] = 0$ and $\mathrm{Var}[X_1] = 1$, then $X_2 = -X_1$ has also a standard normal distribution, and is negatively correlated with X_1, as

$$\mathrm{Cov}(X_1, X_2) = \mathrm{E}[-X_1^2] - \mathrm{E}[X_1]\,\mathrm{E}[-X_1] = -1 < 0. \tag{24}$$

The other advantage of this method is that we can recycle the draws. Once we have generated the draws x_r from X_1, the draws from X_2 are obtained using $1 - x_r$ and $-x_r$, respectively.

Now, we have to be careful when this technique is used for the general case (2). Indeed, it must be verified first that $g(X_1)$ and $g(X_2)$ are indeed negatively correlated. And it is not guaranteed by the fact that X_1 and X_2 are negatively correlated. Consider two examples.

First, consider $g(X) = \left(x - \frac{1}{2}\right)^2$. Applying the antithetic method with

$$X_1 = \left(X - \frac{1}{2}\right)^2 \text{ and } X_2 = \left((1-X) - \frac{1}{2}\right)^2 \tag{25}$$

does not work, as

$$\mathrm{Cov}(X_1, X_2) = \frac{1}{180} > 0. \tag{26}$$

Actually, applying the antithetic method would *increase* the variance here, which is not desirable.

Second, consider $g(X) = e^X$, as in the example presented in Section 3. We apply the antithetic method using

$$Y = \frac{e^X + e^{1-X}}{2}. \tag{27}$$

Here, the two transformed random variables are negatively correlated:

$$\begin{aligned} \mathrm{Cov}(e^X, e^{1-X}) &= \mathrm{E}[e^X e^{1-X}] - \mathrm{E}[e^X]\,\mathrm{E}[e^{1-X}] \\ &= e - (e-1)^2 \\ &= -0.2342106136. \end{aligned} \tag{28}$$

Therefore, the variance of Y given by (19) is 0.0039124969, as opposed to $0.2420356075/2 = 0.1210178037$ if the two sets of draws were independent.

It means that for 10'000 draws from Y, the standard error decreases from 0.0034787613 down to 0.0006254996. Moreover, as we use recycled draws, we need only 10'000 draws instead of 20'000.

There are two ways to apply this technique in PandasBiogeme. First, it is possible to explicitly write the integrand as follows:

```
U = bioDraws('U','UNIFORM')
integrand = exp(U) + exp(1-U)
simulatedI = MonteCarlo(integrand) / 2.0
```

and to divide the number of draws by two

```
R = 10000
```

	Uniform (Anti)	Halton13 (Anti)	MLHS (Anti)
Simulated	1.71779	1.71829	1.71828
Error	-0.000494215	1.02955e-05	1.9781e-07

Table 2: Estimation with different types of draws, with 20'000 antithetic draws, using the explicit way

The complete specification file for PandasBiogeme is available in Appendix A.3. But the simplest way is to instruct PandasBiogeme to use antithetic draws

```
integrand = exp(bioDraws('U','UNIFORM_ANTI'))
simulatedI = MonteCarlo(integrand)
```

In this case, if $R = 20'000$ are requested, PandasBiogeme will actually generate half of them, and complete them with their antithetic version.

	Uniform (Anti)	Halton13 (Anti)	MLHS (Anti)
Simulated	1.71894	1.71829	1.71828
Error	0.000661971	1.02955e-05	1.27776e-07

Table 3: Estimation with different types of draws, with 20'000 antithetic draws, using PandasBiogeme's draws

The complete specification file for PandasBiogeme is available in Appendix A.4.

We encourage the reader to perform similar tests for other simple integrals. For instance,

$$\int_0^1 \left(x - \frac{1}{2}\right)^2 dx = \frac{1}{12} \tag{29}$$

or

$$\int_{-2}^{2} \left(e^{-x} + \frac{1}{2+\varepsilon-x} \right) dx = e^2 - e^{-2} + \log \frac{4+\varepsilon}{\varepsilon}, \tag{30}$$

where $\varepsilon > 0$. Note that the domain of integration is not $[0, 1]$.

5 Mixtures of logit

Consider an individual n, a choice set $\mathcal{C}_n$, and an alternative $i \in \mathcal{C}_n$. The probability to choose i is given by the choice model:

$$P_n(i|x, \theta, \mathcal{C}_n), \tag{31}$$

where x is a vector of explanatory variables and θ is a vector of parameters to be estimated from data. In the random utility framework, a utility function is defined for each individual n and each alternative $i \in \mathcal{C}_n$:

$$U_{in}(x, \theta) = V_{in}(x, \theta) + \varepsilon_{in}(\theta), \tag{32}$$

where $V_{in}(x, \theta)$ is deterministic and ε_{in} is a random variable independent from x. The model is then written:

$$P_n(i|x, \theta, \mathcal{C}_n) = \Pr(U_{in}(x, \theta) \geq U_{jn}(x, \theta), \forall j \in \mathcal{C}_n). \tag{33}$$

Specific models are obtained from assumptions about the distribution of ε_{in}. Namely, if ε_{in} are i.i.d. (across both i and n) extreme value distributed, we obtain the logit model:

$$P_n(i|x, \theta, \mathcal{C}_n) = \frac{e^{V_{in}(x,\theta)}}{\sum_{j \in \mathcal{C}_n} e^{V_{jn}(x,\theta)}}. \tag{34}$$

Mixtures of logit are obtained when some of the parameters θ are distributed instead of being fixed. Denote $\theta = (\theta_f, \theta_d)$, where θ_f is the vector of fixed parameters, while θ_d is the vector of distributed parameters, so that the choice model, conditional on θ_d, is

$$P_n(i|x, \theta_f, \theta_d, \mathcal{C}_n). \tag{35}$$

A distribution is to be assumed for θ_d. We denote the pdf of this distribution by $f_{\theta_d}(\xi; \gamma)$, where γ contains the parameters of the distribution. Parameters γ are sometimes called the *deep parameters* of the model. Therefore, the choice model becomes:

$$P_n(i|x, \theta_f, \gamma, \mathcal{C}_n) = \int_{\xi} P_n(i|x, \theta_f, \xi, \mathcal{C}_n) f_{\theta_d}(\xi) d\xi, \tag{36}$$

where θ_f and γ must be estimated from data. The above integral has no analytical expression, even when the kernel $P_n(i|x, \theta_f, \xi, \mathcal{C}_n)$ is a logit model. Therefore, it must be calculated with numerical integration or Monte-Carlo integration. We do both here to investigate the precision of the variants of Monte-Carlo integration.

5.1 Comparison of integration methods on one integral

We consider the Swissmetro example (Bierlaire et al., 2001). The data file is available from `biogeme.epfl.ch`. At first, we keep only the first observation, using the following statements:

```
p = pd.read_csv("swissmetro.dat",sep='\t')
p = p.drop(p[p.index != 0].index)
database = db.Database("swissmetro",p)
```

Consider the following specification:

- Variables x: see variables in the data file and new variables defined in Section A.5.
- Fixed parameters θ_f

  ```
  ASC_CAR = 0.137
  ASC_TRAIN = -0.402
  ASC_SM = 0
  B_COST = -1.29
  ```

- Deep parameters γ:

  ```
  B_TIME = -2.26
  B_TIME_S = 1.66
  ```

- We define the coefficient of travel time to be distributed, using the random variable omega, that is assumed to be normally distributed:

  ```
  B_TIME_RND = B_TIME + B_TIME_S * omega
  ```

  The parameter B_TIME is the mean of B_TIME_RND, and B_TIME_S^2 is its variance. Note that B_TIME_S is just a parameter, and is **not** the standard deviation. It can be positive or negative.

- Utility functions V_{in}:

  ```
  V1 = ASC_TRAIN + \
       B_TIME_RND * TRAIN_TT_SCALED + \
       B_COST * TRAIN_COST_SCALED
  V2 = ASC_SM + \
  ```

```
      B_TIME_RND * SM_TT_SCALED + \
      B_COST * SM_COST_SCALED
V3 = ASC_CAR + \
      B_TIME_RND * CAR_TT_SCALED + \
      B_COST * CAR_CO_SCALED
V = {1: V1, 2: V2, 3: V3}
```

- Choice set $\mathcal{C}_n$, characterized by the availability conditions:

```
CAR_AV_SP = \
  DefineVariable('CAR_AV_SP',CAR_AV * ( SP != 0 ))
TRAIN_AV_SP = \
  DefineVariable('TRAIN_AV_SP',TRAIN_AV * ( SP != 0 ))
av = {1: TRAIN_AV_SP,
      2: SM_AV,
      3: CAR_AV_SP}
```

As there is only one random parameter, the model (36) can be calculated using numerical integration. It is done in PandasBiogeme using the following procedure:

1. Mention that omega is a random variable:

   ```
   omega = RandomVariable('omega')
   ```

2. Define its pdf:

   ```
   import biogeme.distributions as dist
   density = dist.normalpdf(omega).
   ```

3. Define the integrand from the logit model, where the probability of the alternative observed to be chosen is calculated (which is typical when calculating a likelihood function):

   ```
   import biogeme.models as models
   integrand = models.logit(V,av,CHOICE)
   ```

4. Calculate the integral:

   ```
   analyticalI = Integrate(integrand*density,'omega')
   ```

The complete specification file for PandasBiogeme is available in Appendix A.5. The value of the choice model for first observation in the data file is

$$I = \int_{\xi} P_n(i|x, \theta_f, \xi, \mathcal{C}_n) f_{\theta_d}(\xi) d\xi = 0.6378498363459125. \tag{37}$$

We then compare Monte-Carlo integration with various types of draws. To do that, we implement a function that takes the parameter β as an argument, and returns the integrand (the logit model), using the following syntax:

```
def logit(THE_B_TIME_RND):
    V1 = ASC_TRAIN + \
         THE_B_TIME_RND * TRAIN_TT_SCALED + \
         B_COST * TRAIN_COST_SCALED
    V2 = ASC_SM + \
         THE_B_TIME_RND * SM_TT_SCALED + \
         B_COST * SM_COST_SCALED
    V3 = ASC_CAR + \
         THE_B_TIME_RND * 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}

    # The choice model is a logit, with availability conditions
    integrand = models.logit(V,av,CHOICE)
    return integrand
```

Each version of the beta is specified with different types of draws, as described earlier:

```
B_TIME_RND_normal = B_TIME + B_TIME_S * \
                    bioDraws('B_NORMAL','NORMAL')
B_TIME_RND_anti = B_TIME + B_TIME_S * \
                  bioDraws('B_ANTI','NORMAL_ANTI')
B_TIME_RND_halton = B_TIME + B_TIME_S * \
                    bioDraws('B_HALTON','NORMAL_HALTON2')
B_TIME_RND_mlhs = B_TIME + B_TIME_S * bioDraws('B_MLHS','NORMAL_MLHS')
B_TIME_RND_antimlhs = B_TIME + B_TIME_S * \
                      bioDraws('B_ANTIMLHS','NORMAL_MLHS_ANTI')
```

The complete specification file for PandasBiogeme is available in Appendix A.6. Using the result of the numerical integration as the "true" value of the integral, We obtain the following results:

Analytical integral	0.63785
Monte-Carlo integral	0.637356
Monte-Carlo integral, antithetic	0.637549
Monte-Carlo integral, Halton	0.637908
Monte-Carlo integral, MLHS	0.637839
Monte-Carlo integral, MLHS antithetic	0.637861

5.2 Comparison of integration methods for maximum likelihood estimation

We now estimate the parameters of the model using all observations in the data set associated with work trips. Observations such that the dependent variable CHOICE is 0 are also removed.

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

The estimation using numerical integration is performed using the following statements:

```
condprob = models.logit(V,av,CHOICE)
prob = Integrate(condprob * density,'omega')
logprob = log(prob)
biogeme = bio.BIOGEME(database,logprob)
biogeme.modelName = '06estimationIntegral'
results = biogeme.estimate()
```

The complete specification file for PandasBiogeme is available in Appendix A.7.

Number of estimated parameters	5
Sample size	6768
Excluded observations	3960
Init log likelihood	-6879.406
Final log likelihood	-5214.879
Likelihood ratio test for the init. model	3329.055
Rho-square for the init. model	0.242
Rho-square-bar for the init. model	0.241
Akaike Information Criterion	10439.76
Bayesian Information Criterion	10473.86
Final gradient norm	8.0436E-05
Diagnostic	b'CONVERGENCE: NORM_OF_PROJECTED_GRA
Database readings	16
Iterations	15
Optimization time	0:00:22.584674
Nbr of threads	8

Table 4: General statistics for the model estimation with analytical integral

For Monte-Carlo integration, we use the following statements:

```
prob = models.logit(V,av,CHOICE)
```

	Value	Rob. Std err	Rob. t-test	Rob. p-value
ASC_CAR	0.137	0.0517	2.65	0.00803
ASC_TRAIN	-0.401	0.0656	-6.12	9.64e-10
B_COST	-1.29	0.0863	-14.9	0.0
B_TIME	-2.26	0.117	-19.4	0.0
B_TIME_S	1.65	0.125	13.3	0.0

Table 5: Parameter estimates for the model estimation with analytical integral

```
logprob = log(MonteCarlo(prob))
R= 20000
biogeme = bio.BIOGEME(database,logprob,numberOfDraws=R)
```

The complete specification file for PandasBiogeme is available in Appendix A.8.

We now compare the estimation results for various types of draws: normal draws, antithetic, Halton, MLHS and antithetic MLHS. We provide the results for 2'000 and 500 draws. The final log likelihood in each case, as well as the estimation time are summarized in Table 6.

It appears with that example that the use of pure random draws is the least precise way to calculate Monte-Carlo integrals. The use of antithetic, Halton or MLHS draws is therefore recommended.

Method	Draws	Log likelihood	Run time
Numerical	—	-5214.879	00:14
Monte-Carlo	2000	-5213.99	04:19
Antithetic	2000	-5214.302	04:06
Halton	2000	-5214.95	04:06
MLHS	2000	-5214.637	04:14
Antithetic MLHS	2000	-5215.253	04:08
Monte-Carlo	500	-5219.581	00:57
Antithetic	500	-5214.856	01:00
Halton	500	-5215.069	01:01
MLHS	500	-5215.013	01:01
Antithetic MLHS	500	-5215.292	01:02

Table 6: Final log likelihood and run time for each integration method

6 Conclusion

This document describes the variants of Monte-Carlo integration. It is recommended to perform some analysis using the simulation mode of Pandas-Biogeme, in order to investigate the performance of each type of draws before starting a maximum likelihood estimation, that may take a while to converge. In the example provided in this document, the antithetic draws method, combined with MLHS appeared to be the most precise. This result is not universal. The analysis must be performed on a case by case basis.

A Complete specification files

A.1 01simpleIntegral.py

```
"""File: 01simpleIntegral.py
 Author: Michel Bierlaire, EPFL
 Date: Wed Dec 11 16:20:24 2019

Calculation of a simple integral using Monte-Carlo integration.

"""

import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.draws as draws
from biogeme.expressions import exp, bioDraws, MonteCarlo

# We create a fake database with one entry, as it is required
# to store the draws
pandas = pd.DataFrame()
pandas['FakeColumn'] = [1.0]
database = db.Database('fakeDatabase',pandas)

integrand = exp(bioDraws('U','UNIFORM'))
simulatedI = MonteCarlo(integrand)

trueI = exp(1.0) - 1.0

R = 2000

sampleVariance = \
  MonteCarlo(integrand*integrand) - simulatedI * simulatedI
stderr = (sampleVariance / R)**0.5
error = simulatedI - trueI

simulate = {'Analytical Integral': trueI,
            'Simulated Integral': simulatedI,
            'Sample variance   ': sampleVariance,
            'Std Error         ': stderr,
            'Error             ': error}

biogeme = bio.BIOGEME(database,simulate,numberOfDraws=R)
biogeme.modelName = f'01simpleIntegral_{R}'
results = biogeme.simulate()
print(f'Number of draws: {R}')
for c in results.columns:
    print(f'{c}: {results.loc[0,c]}')
```

```
45
46 # With 10 times more draws
47 biogeme2 = bio.BIOGEME(database,simulate,numberOfDraws=10*R)
48 biogeme2.modelName = '01simpleIntegral_{10*R}'
49 results2 = biogeme2.simulate()
50 print(f'Number of draws: {10*R}')
51 for c in results.columns:
52     print(f'{c}: {results2.loc[0,c]}')
```

A.2 02simpleIntegral.py

```
1  """File: 02simpleIntegral.py
2
3   Author: Michel Bierlaire, EPFL
4   Date: Wed Dec 11 16:57:51 2019
5
6  Calculation of a simple integral using numerical integration and
7  Monte-Carlo integration with various types of draws, including Halton
8  draws base 13. It illustrates how to use draws that are not directly
9  available in Biogeme.
10
11 """
12
13 import pandas as pd
14 import biogeme.database as db
15 import biogeme.biogeme as bio
16 import biogeme.draws as draws
17 from biogeme.expressions import exp, bioDraws, MonteCarlo
18
19
20 # We create a fake database with one entry, as it is required
21 # to store the draws
22 pandas = pd.DataFrame()
23 pandas['FakeColumn'] = [1.0]
24 database = db.Database('fakeDatabase',pandas)
25
26 # The user can define new draws. For example, Halton draws
27 # with base 13, skipping the first 10 draws.
28
29 def halton13(sampleSize,numberOfDraws):
30     return draws.getHaltonDraws(sampleSize,
31                                 numberOfDraws,
32                                 base=13,
33                                 skip=10)
34
35 mydraws = {'HALTON13':(halton13,'Halton draws, base 13, skipping 10')}
36 database.setRandomNumberGenerators(mydraws)
37
38
```

```
integrand = exp(bioDraws('U','UNIFORM'))
simulatedI = MonteCarlo(integrand)

integrand_halton = exp(bioDraws('U_halton','UNIFORM_HALTON2'))
simulatedI_halton = MonteCarlo(integrand_halton)

integrand_halton13 = exp(bioDraws('U_halton13','HALTON13'))
simulatedI_halton13 = MonteCarlo(integrand_halton13)

integrand_mlhs = exp(bioDraws('U_mlhs','UNIFORM_MLHS'))
simulatedI_mlhs = MonteCarlo(integrand_mlhs)

trueI = exp(1.0) - 1.0

R = 20000

sampleVariance = \
  MonteCarlo(integrand*integrand) - simulatedI * simulatedI
stderr = (sampleVariance / R)**0.5
error = simulatedI - trueI

sampleVariance_halton = \
  MonteCarlo(integrand_halton*integrand_halton) \
  - simulatedI_halton * simulatedI_halton
stderr_halton = (sampleVariance_halton / R)**0.5
error_halton = simulatedI_halton - trueI

sampleVariance_halton13 = \
  MonteCarlo(integrand_halton13*integrand_halton13) \
  - simulatedI_halton13 * simulatedI_halton13
stderr_halton13 = (sampleVariance_halton13 / R)**0.5
error_halton13 = simulatedI_halton13 - trueI


sampleVariance_mlhs = \
  MonteCarlo(integrand_mlhs*integrand_mlhs) \
  - simulatedI_mlhs * simulatedI_mlhs
stderr_mlhs = (sampleVariance_mlhs / R)**0.5
error_mlhs = simulatedI_mlhs - trueI


simulate = {'Analytical Integral': trueI,
            'Simulated Integral': simulatedI,
            'Sample variance   ': sampleVariance,
            'Std Error         ': stderr,
            'Error             ': error,
            'Simulated Integral (Halton)': simulatedI_halton,
            'Sample variance (Halton)   ': sampleVariance_halton,
            'Std Error (Halton)         ': stderr_halton,
```

```
            'Error (Halton)              ': error_halton,
            'Simulated Integral (Halton13)': simulatedI_halton13,
            'Sample variance (Halton13)   ': sampleVariance_halton13,
            'Std Error (Halton13)         ': stderr_halton13,
            'Error (Halton13)             ': error_halton13,
            'Simulated Integral (MLHS)': simulatedI_mlhs,
            'Sample variance (MLHS)   ': sampleVariance_mlhs,
            'Std Error (MLHS)         ': stderr_mlhs,
            'Error (MLHS)             ': error_mlhs}


biogeme = bio.BIOGEME(database, simulate, numberOfDraws=R)
biogeme.modelName = '02simpleIntegral'
results = biogeme.simulate()
print(f"Analytical integral:{results.iloc[0]['Analytical Integral']:.6g}")
print(f"\t\tUniform\t\tHalton\t\tHalton13\tMLHS")
print(f"Simulated\t{results.iloc[0]['Simulated Integral']:.6g}\t{results.iloc[0]['S
print(f"Sample var.\t{results.iloc[0]['Sample variance   ']:.6g}\t{results.iloc[0]
']:.6g}\t{results.iloc[0]['Sample variance (Halton13)   ']:.6g}\t{results.iloc[0][
']:.6g}")
print(f"Std error\t{results.iloc[0]['Std Error         ']:.6g}\t{results.iloc[0]['S
']:.6g}\t{results.iloc[0]['Std Error (Halton13)         ']:.6g}\t{results.iloc[0][
']:.6g}")
print(f"Error\t\t{results.iloc[0]['Error             ']:.6g}\t{results.iloc[0]['Er
']:.6g}\t{results.iloc[0]['Error (Halton13)             ']:.6g}\t{results.iloc[0][
']:.6g}")
```

A.3 03 antitheticExplicit .py

```
"""File: 03antitheticExplicit.py

 Author: Michel Bierlaire, EPFL
 Date: Wed Dec 11 17:04:40 2019

Calculation of a simple integral using Monte-Carlo integration with
antithetic Halton draws base 13. It illustrates how to use draws that
are not directly available in Biogeme.

"""

import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.draws as draws
from biogeme.expressions import exp, bioDraws, MonteCarlo


# We create a fake database with one entry, as it is required to store the draws
```

```
pandas = pd.DataFrame()
pandas['FakeColumn'] = [1.0]
database = db.Database("fakeDatabase",pandas)

# The user can define new draws. For example, Halton draws with base 13, skipping

def halton13(sampleSize,numberOfDraws):
    return draws.getHaltonDraws(sampleSize,
                                numberOfDraws,
                                base=13,
                                skip=10)

mydraws = {'HALTON13':(halton13,'Halton draws, base 13, skipping 10')}
database.setRandomNumberGenerators(mydraws)

U = bioDraws('U','UNIFORM')
integrand = exp(U) + exp(1-U)
simulatedI = MonteCarlo(integrand) / 2.0

U_halton13 = bioDraws('U_halton13','HALTON13')
integrand_halton13 = exp(U_halton13) + exp(1-U_halton13)
simulatedI_halton13 = MonteCarlo(integrand_halton13) / 2.0

U_mlhs = bioDraws('U_mlhs','UNIFORM_MLHS')
integrand_mlhs = exp(U_mlhs) + exp(1-U_mlhs)
simulatedI_mlhs = MonteCarlo(integrand_mlhs) / 2.0

trueI = exp(1.0) - 1.0

R = 10000

error = simulatedI - trueI

error_halton13 = simulatedI_halton13 - trueI

error_mlhs = simulatedI_mlhs - trueI


simulate = {'Analytical Integral': trueI,
            'Simulated Integral': simulatedI,
            'Error             ': error,
            'Simulated Integral (Halton13)': simulatedI_halton13,
            'Error (Halton13)              ': error_halton13,
            'Simulated Integral (MLHS)': simulatedI_mlhs,
            'Error (MLHS)              ': error_mlhs}

biogeme  = bio.BIOGEME(database,simulate,numberOfDraws=R)
biogeme.modelName = "03antitheticExplicit"
results = biogeme.simulate()
```

```
print(f"Analytical integral:{results.iloc[0]['Analytical Integral']:.6f}")
print(f"\t\tUniform (Anti)\t\tHalton13 (Anti)\t\tMLHS (Anti)")
print(f"Simulated\t{results.iloc[0]['Simulated Integral']:.6g}\t{results.iloc[0]['S
print(f"Error\t\t{results.iloc[0]['Error              ']:.6g}\t{results.iloc[0]['Er
']:.6g}\t{results.iloc[0]['Error (MLHS)             ']:.6g}")
```

A.4 03antithetic.py

```
""" File: 03antithetic.py

 Author: Michel Bierlaire, EPFL
 Date: Wed Dec 11 17:00:05 2019

Calculation of a simple integral using Monte-Carlo integration with
Halton draws base 13. It illustrates how to use draws that
are not directly available in Biogeme.


"""

import numpy as np
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.draws as draws
from biogeme.expressions import exp, bioDraws, MonteCarlo


# We create a fake database with one entry, as it is required to store the draws
pandas = pd.DataFrame()
pandas['FakeColumn'] = [1.0]
database = db.Database("fakeDatabase",pandas)

# The user can define new draws. For example, Halton draws with base 13, skipping

def halton13_anti(sampleSize,numberOfDraws):
    R = int(numberOfDraws / 2)
    d = draws.getHaltonDraws(sampleSize,
                             R,
                             base=13,
                             skip=10)
    return np.concatenate((d,1-d),axis=1)

mydraws = {'HALTON13_ANTI':(halton13_anti,'Antithetic Halton draws, base 13, skipp
database.setRandomNumberGenerators(mydraws)


integrand = exp(bioDraws('U','UNIFORM_ANTI'))
simulatedI = MonteCarlo(integrand)
```

```
42
43 integrand_halton13 = exp(bioDraws('U_halton13','HALTON13_ANTI'))
44 simulatedI_halton13 = MonteCarlo(integrand_halton13)
45
46 integrand_mlhs = exp(bioDraws('U_mlhs','UNIFORM_MLHS_ANTI'))
47 simulatedI_mlhs = MonteCarlo(integrand_mlhs)
48
49 trueI = exp(1.0) - 1.0
50
51 R = 20000
52
53 error = simulatedI - trueI
54
55 error_halton13 = simulatedI_halton13 - trueI
56
57 error_mlhs = simulatedI_mlhs - trueI
58
59
60 simulate = {'Analytical Integral': trueI,
61             'Simulated Integral': simulatedI,
62             'Error             ': error,
63             'Simulated Integral (Halton13)': simulatedI_halton13,
64             'Error (Halton13)             ': error_halton13,
65             'Simulated Integral (MLHS)': simulatedI_mlhs,
66             'Error (MLHS)             ': error_mlhs}
67
68
69
70 biogeme  = bio.BIOGEME(database,simulate,numberOfDraws=R)
71 biogeme.modelName = "03antithetic"
72 results = biogeme.simulate()
73 print(f"Analytical integral:{results.iloc[0]['Analytical Integral']:.6f}")
74 print(f"\t\tUniform (Anti)\t\tHalton13 (Anti)\t\tMLHS (Anti)")
75 print(f"Simulated\t{results.iloc[0]['Simulated Integral']:.6g}\t{results.iloc[0]['S
76 print(f"Error\t\t{results.iloc[0]['Error             ']:.6g}\t{results.iloc[0]['Err
   ']:.6g}\t{results.iloc[0]['Error (MLHS)             ']:.6g}")
```

A.5 04normalMixtureNumerical.py

```
1  """ File: 04normalMixtureNumerical.py
2
3   Author: Michel Bierlaire, EPFL
4   Date: Wed Dec 11 17:06:52 2019
5
6  Calculation of a mixtures of logit models where the integral is
7  calculated using numerical integration.
8
9  """
10
```

```
11 import pandas as pd
12 import biogeme.database as db
13 import biogeme.biogeme as bio
14 import biogeme.draws as draws
15 import biogeme.distributions as dist
16 import biogeme.models as models
17
18 from biogeme.expressions import Beta, DefineVariable, RandomVariable, Integrate
19
20 p = pd.read_csv("swissmetro.dat",sep='\t')
21 # Use only the first observation (index 0)
22 p = p.drop(p[p.index != 0].index)
23 database = db.Database("swissmetro",p)
24
25 globals().update(database.variables)
26
27 #Parameters
28 ASC_CAR = 0.137
29 ASC_TRAIN = -0.402
30 ASC_SM = 0
31 B_TIME = -2.26
32 B_TIME_S = 1.66
33 B_COST = -1.29
34
35 # Define a random parameter, normally distributed,
36 # designed to be used for integration
37 omega = RandomVariable('omega')
38 density = dist.normalpdf(omega)
39 B_TIME_RND = B_TIME + B_TIME_S * omega
40
41 # Utility functions
42
43 #If the person has a GA (season ticket) her
44 #incremental cost is actually 0
45 #rather than the cost value gathered from the
46 # network data.
47 SM_COST = SM_CO   * (  GA   ==  0  )
48 TRAIN_COST =  TRAIN_CO   * (  GA   ==  0  )
49
50 # For numerical reasons, it is good practice to scale the data to
51 # that the values of the parameters are around 1.0.
52 # A previous estimation with the unscaled data has generated
53 # parameters around -0.01 for both cost and time.
54 # Therefore, time and cost are multipled my 0.01.
55
56 TRAIN_TT_SCALED = \
57   DefineVariable('TRAIN_TT_SCALED', TRAIN_TT / 100.0,database)
58 TRAIN_COST_SCALED = \
59   DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100,database)
```

```
60 SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0,database)
61 SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100,database)
62 CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100,database)
63 CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100,database)
64 CAR_AV_SP = DefineVariable('CAR_AV_SP',CAR_AV * ( SP != 0
   ),database)
65 TRAIN_AV_SP = DefineVariable('TRAIN_AV_SP',TRAIN_AV * ( SP !=
   0 ),database)
66
67 V1 = ASC_TRAIN + \
68       B_TIME_RND * TRAIN_TT_SCALED + \
69       B_COST * TRAIN_COST_SCALED
70 V2 = ASC_SM + \
71       B_TIME_RND * SM_TT_SCALED + \
72       B_COST * SM_COST_SCALED
73 V3 = ASC_CAR + \
74       B_TIME_RND * CAR_TT_SCALED + \
75       B_COST * CAR_CO_SCALED
76
77 # Associate utility functions with the numbering of alternatives
78 V = {1: V1,
79       2: V2,
80       3: V3}
81
82 # Associate the availability conditions with the alternatives
83
84 av = {1: TRAIN_AV_SP,
85        2: SM_AV,
86        3: CAR_AV_SP}
87
88 # The choice model is a logit, with availability conditions
89 integrand = models.logit(V,av,CHOICE)
90 numericalI = Integrate(integrand*density,'omega')
91
92 simulate = {'Numerical': numericalI}
93
94 biogeme = bio.BIOGEME(database,simulate)
95 results = biogeme.simulate()
96 print('Mixture of logit - numerical integration: ',results.iloc[0]['Numerical'])
```

A.6 05normalMixtureMonteCarlo.py

```
1 """File: 05normalMixtureMonteCarlo.py
2
3  Author: Michel Bierlaire, EPFL
4  Date: Wed Dec 11 17:11:45 2019
5
6 Calculation of a mixtures of logit models where the integral is
7 calculated using numerical integration and Monte-Carlo integration
```

```
8  with various types of draws.
9
10 """
11
12 import pandas as pd
13 import biogeme.database as db
14 import biogeme.biogeme as bio
15 import biogeme.draws as draws
16 import biogeme.distributions as dist
17 import biogeme.models as models
18
19 from biogeme.expressions import Beta, DefineVariable, RandomVariable, Integrate, M
20
21 p = pd.read_csv("swissmetro.dat",sep='\t')
22 # Use only the first observation
23 p = p.drop(p[p.index != 0].index)
24 database = db.Database("swissmetro",p)
25
26 globals().update(database.variables)
27
28 #Parameters
29 ASC_CAR = 0.137
30 ASC_TRAIN = -0.402
31 ASC_SM = 0
32 B_TIME = -2.26
33 B_TIME_S = 1.66
34 B_COST = -1.29
35
36 # Define a random parameter, normally distributed,
37 # designed to be used for integration
38 omega = RandomVariable('omega')
39 density = dist.normalpdf(omega)
40 B_TIME_RND = B_TIME + B_TIME_S * omega
41 B_TIME_RND_normal = B_TIME + B_TIME_S * \
42                     bioDraws('B_NORMAL','NORMAL')
43 B_TIME_RND_anti = B_TIME + B_TIME_S * \
44                   bioDraws('B_ANTI','NORMAL_ANTI')
45 B_TIME_RND_halton = B_TIME + B_TIME_S * \
46                     bioDraws('B_HALTON','NORMAL_HALTON2')
47 B_TIME_RND_mlhs = B_TIME + B_TIME_S * bioDraws('B_MLHS','NORMAL_MLHS')
48 B_TIME_RND_antimlhs = B_TIME + B_TIME_S * \
49                       bioDraws('B_ANTIMLHS','NORMAL_MLHS_ANTI')
50
51 # Utility functions
52
53 #If the person has a GA (season ticket) her
54 #incremental cost is actually 0
55 #rather than the cost value gathered from the
56 # network data.
```

```
57 SM_COST = SM_CO   * ( GA   == 0   )
58 TRAIN_COST = TRAIN_CO   * ( GA   == 0   )
59
60 # For numerical reasons, it is good practice to scale the data to
61 # that the values of the parameters are around 1.0.
62 # A previous estimation with the unscaled data has generated
63 # parameters around -0.01 for both cost and time.
64 # Therefore, time and cost are multipled my 0.01.
65
66 TRAIN_TT_SCALED = \
67   DefineVariable('TRAIN_TT_SCALED', TRAIN_TT / 100.0,database)
68 TRAIN_COST_SCALED = \
69   DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100,database)
70 SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0,database)
71 SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100,database)
72 CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100,database)
73 CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100,database)
74 CAR_AV_SP =  DefineVariable('CAR_AV_SP',CAR_AV  * ( SP   != 0
   ),database)
75 TRAIN_AV_SP =  DefineVariable('TRAIN_AV_SP',TRAIN_AV  * ( SP   !=
   0   ),database)
76
77 def logit(THE_B_TIME_RND):
78     V1 = ASC_TRAIN + \
79          THE_B_TIME_RND * TRAIN_TT_SCALED + \
80          B_COST * TRAIN_COST_SCALED
81     V2 = ASC_SM + \
82          THE_B_TIME_RND * SM_TT_SCALED + \
83          B_COST * SM_COST_SCALED
84     V3 = ASC_CAR + \
85          THE_B_TIME_RND * CAR_TT_SCALED + \
86          B_COST * CAR_CO_SCALED
87
88     # Associate utility functions with the numbering of alternatives
89     V = {1: V1,
90          2: V2,
91          3: V3}
92
93     # Associate the availability conditions with the alternatives
94     av = {1: TRAIN_AV_SP,
95           2: SM_AV,
96           3: CAR_AV_SP}
97
98     # The choice model is a logit, with availability conditions
99     integrand = models.logit(V,av,CHOICE)
100    return integrand
101
102 numericalI = Integrate(logit(B_TIME_RND)*density,'omega')
103 normal = MonteCarlo(logit(B_TIME_RND_normal))
```

```
anti = MonteCarlo(logit(B_TIME_RND_anti))
halton = MonteCarlo(logit(B_TIME_RND_halton))
mlhs = MonteCarlo(logit(B_TIME_RND_mlhs))
antimlhs = MonteCarlo(logit(B_TIME_RND_antimlhs))

simulate = {'Numerical': numericalI,
            'MonteCarlo': normal,
            'Antithetic': anti,
            'Halton': halton,
            'MLHS':mlhs,
            'Antithetic MLHS': antimlhs}

R = 20000
biogeme = bio.BIOGEME(database,simulate,numberOfDraws=R)
results = biogeme.simulate()
print(f'Number of draws: {10*R}')
for c in results.columns:
    print(f'{c}:\t{results.loc[0,c]}')
```

A.7 06estimationIntegral.py

```
""" File: 06estimationIntegral.py

 Author: Michel Bierlaire, EPFL
 Date: Wed Dec 11 17:17:05 2019

Estimation of a mixtures of logit models where the integral is
calculated using numerical integration.

"""

import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.distributions as dist
import biogeme.models as models
from biogeme.expressions import Beta, DefineVariable, RandomVariable, Integrate, lo

pandas = pd.read_csv("swissmetro.dat",sep='\t')
database = db.Database("swissmetro",pandas)

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

globals().update(database.variables)

# Removing some observations can be done directly using pandas.
#remove = (((database.data.PURPOSE != 1) & (database.data.PURPOSE != 3)) | (databa
```

```
#database.data.drop(database.data[remove].index,inplace=True)

# Here we use the "biogeme" way for backward compatibility
exclude = (( PURPOSE != 1 ) * (  PURPOSE  !=  3  ) +  ( CHOICE == 0 )) > 0
database.remove(exclude)


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

# Define a random parameter, normally distirbuted, designed to be used
# for Monte-Carlo simulation

omega = RandomVariable('omega')
density = dist.normalpdf(omega)
B_TIME_RND = B_TIME + B_TIME_S * omega


# Utility functions

#If the person has a GA (season ticket) her incremental cost is actually 0
#rather than the cost value gathered from the
# network data.
SM_COST = SM_CO   * (  GA   ==  0  )
TRAIN_COST =  TRAIN_CO   * (  GA   ==  0  )

# For numerical reasons, it is good practice to scale the data to
# that the values of the parameters are around 1.0.
# A previous estimation with the unscaled data has generated
# parameters around -0.01 for both cost and time. Therefore, time and
# cost are multipled my 0.01.

TRAIN_TT_SCALED = DefineVariable('TRAIN_TT_SCALED',\
                                 TRAIN_TT / 100.0,database)
TRAIN_COST_SCALED = DefineVariable('TRAIN_COST_SCALED',\
                                   TRAIN_COST / 100,database)
SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0,database)
SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100,database)
CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100,database)
CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100,database)

V1 = ASC_TRAIN + B_TIME_RND * TRAIN_TT_SCALED + B_COST * TRAIN_COST_SCALED
V2 = ASC_SM + B_TIME_RND * SM_TT_SCALED + B_COST * SM_COST_SCALED
V3 = ASC_CAR + B_TIME_RND * 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

CAR_AV_SP = DefineVariable('CAR_AV_SP',CAR_AV * ( SP != 0 ),database)
TRAIN_AV_SP = DefineVariable('TRAIN_AV_SP',TRAIN_AV * ( SP != 0 ),database)

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

# The choice model is a logit, with availability conditions
condprob = models.logit(V,av,CHOICE)
prob = Integrate(condprob * density,'omega')
logprob = log(prob)

biogeme = bio.BIOGEME(database,logprob)

biogeme.modelName = '06estimationIntegral'

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

A.8 07estimationMonteCarlo.py

```
""" File: 07estimationMonteCarlo.py

 Author: Michel Bierlaire, EPFL
 Date: Wed Dec 11 17:25:14 2019

Estimation of a mixtures of logit models where the integral is
approximated using MonteCarlo integration.

"""

import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
from biogeme.expressions import Beta, DefineVariable, bioDraws, MonteCarlo, log

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

```
19
20 # The Pandas data structure is available as database.data. Use all the
21 # Pandas functions to invesigate the database
22 #print(database.data.describe())
23
24 globals().update(database.variables)
25
26 # Removing some observations can be done directly using pandas.
27 #remove = (((database.data.PURPOSE != 1) & (database.data.PURPOSE != 3)) | (databa
28 #database.data.drop(database.data[remove].index,inplace=True)
29
30 # Here we use the "biogeme" way for backward compatibility
31 exclude = (( PURPOSE != 1 ) * ( PURPOSE != 3 ) + ( CHOICE == 0 )) > 0
32 database.remove(exclude)
33
34
35 ASC_CAR = Beta('ASC_CAR',0,None,None,0)
36 ASC_TRAIN = Beta('ASC_TRAIN',0,None,None,0)
37 ASC_SM = Beta('ASC_SM',0,None,None,1)
38 B_TIME = Beta('B_TIME',0,None,None,0)
39 B_TIME_S = Beta('B_TIME_S',1,None,None,0)
40 B_COST = Beta('B_COST',0,None,None,0)
41
42 # Define a random parameter, normally distributed, designed to be used
43 # for Monte-Carlo simulation
44 B_TIME_RND = B_TIME + B_TIME_S * bioDraws('B_TIME_RND','NORMAL')
45
46 # Utility functions
47
48 #If the person has a GA (season ticket) her incremental cost is actually 0
49 #rather than the cost value gathered from the
50 # network data.
51 SM_COST = SM_CO * ( GA == 0 )
52 TRAIN_COST = TRAIN_CO * ( GA == 0 )
53
54 # For numerical reasons, it is good practice to scale the data to
55 # that the values of the parameters are around 1.0.
56 # A previous estimation with the unscaled data has generated
57 # parameters around -0.01 for both cost and time. Therefore, time and
58 # cost are multipled my 0.01.
59
60 TRAIN_TT_SCALED = DefineVariable('TRAIN_TT_SCALED',\
61                                  TRAIN_TT / 100.0,database)
62 TRAIN_COST_SCALED = DefineVariable('TRAIN_COST_SCALED',\
63                                    TRAIN_COST / 100,database)
64 SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0,database)
65 SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100,database)
66 CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100,database)
67 CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100,database)
```

```
68
69 V1 = ASC_TRAIN + B_TIME_RND * TRAIN_TT_SCALED + B_COST * TRAIN_COST_SCALED
70 V2 = ASC_SM + B_TIME_RND * SM_TT_SCALED + B_COST * SM_COST_SCALED
71 V3 = ASC_CAR + B_TIME_RND * CAR_TT_SCALED + B_COST * CAR_CO_SCALED
72
73 # Associate utility functions with the numbering of alternatives
74 V = {1: V1,
75      2: V2,
76      3: V3}
77
78 # Associate the availability conditions with the alternatives
79
80 CAR_AV_SP =  DefineVariable('CAR_AV_SP',CAR_AV  * (  SP   !=  0
   ),database)
81 TRAIN_AV_SP =  DefineVariable('TRAIN_AV_SP',TRAIN_AV  * (  SP   !=
   0  ),database)
82
83 av = {1: TRAIN_AV_SP,
84       2: SM_AV,
85       3: CAR_AV_SP}
86
87 # The choice model is a logit, with availability conditions
88 prob = models.logit(V,av,CHOICE)
89 logprob = log(MonteCarlo(prob))
90
91 R= 2000
92 biogeme = bio.BIOGEME(database,logprob,numberOfDraws=R)
93
94 biogeme.modelName = '07estimationMonteCarlo'
95 results = biogeme.estimate()
96 results.writeLaTeX()
```

References

Bhat, C. (2001). Quasi-random maximum simulated likelihood estimation of the mixed multinomial logit model, *Transportation Research Part B* **35**: 677–693.

Bhat, C. R. (2003). Simulation estimation of mixed discrete choice models using randomized and scrambled halton sequences, *Transportation Research Part B: Methodological* **37**(9): 837 – 855.
URL: *http://www.sciencedirect.com/science/article/pii/S0191261502000905*

Bierlaire, M., Axhausen, K. and Abay, G. (2001). The acceptance of modal innovation: The case of swissmetro, *Proceedings of the Swiss Transport Research Conference*, Ascona, Switzerland.

Halton, J. H. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, *Numerische Mathematik* **2**(1): 84–90.
URL: *http://dx.doi.org/10.1007/BF01386213*

Hess, S., Train, K. and Polak, J. (2006). On the use of modified latin hypercube sampling (MLHS) method in the estimation of mixed logit model for vehicle choice, *Transportation Research Part B* **40**(2): 147–163.

Ross, S. (2012). *Simulation*, fifth edition edn, Academic Press.
URL: *http://books.google.ch/books?id=sZjDT6MQGF4C*

Sándor, Z. and Train, K. (2004). Quasi-random simulation of discrete choice models, *Transportation Research Part B: Methodological* **38**(4): 313 – 327.

Train, K. (2000). Halton sequences for mixed logit, *Technical Report E00-278*, Department of Economics, University of California, Berkeley.