Monte-Carlo integration with PythonBiogeme

Michel Bierlaire

August 6, 2015

Report TRANSP-OR 150806
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 PythonBiogeme (`biogeme.epfl.ch`) is designed to estimate the parameters of various models using maximum likelihood estimation. It is particularly designed for discrete choice models. In this document, we 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 PythonBiogeme 2.4, and with simulation methods, although a short summary is provided.

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

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

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

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

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:

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

Putting (2) and (3) together, we obtain an approximation to 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 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 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. In PythonBiogeme 2.4, one of them must be selected by setting the parameter RandomDistribution.

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”.

```
BIOGEME_OBJECT.PARAMETERS['RandomDistribution'] = "PSEUDO"
```

Researchers have proposed to use other types of sequences to perform Monte-Carlo integration, called “quasi-random sequences” or “low-discrepancy sequences”. PythonBiogeme 2.4 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).

```
BIOGEME_OBJECT.PARAMETERS['RandomDistribution'] = "HALTON"
```

The third method to generate uniform random numbers implemented in PythonBiogeme 2.4 is called “Modified Latin Hypercube Sampling”, and has been proposed by Hess et al. (2006).

```
BIOGEME_OBJECT.PARAMETERS['RandomDistribution'] = "MHLS"
```

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

3 Illustration with PythonBiogeme 2.4

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:

$$I = e - 1 = 1.7183. \tag{13}$$

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 $R = 20000$, and 0.0011000809 for $R = 200000$. These theoretical values are estimated also below using PythonBiogeme 2.4.

We use PythonBiogeme 2.4 to calculate (15). Note that PythonBiogeme 2.4 requires a data file, which is not necessary in this simplistic case. We use the simulation mode of PythonBiogeme 2.4. It generates output for each row of the data file. In our case, we just need one output, so that we take any data file, and exclude all rows of the file except the first one, using the following syntax:

```
__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1
```

For this specific example, the data included in the file are irrelevant. The generation of draws in PythonBiogeme 2.4 is performed using the command bioDraws('U'), where the argument 'U' provides the name of the random variable associated with the draws. The distribution of the random variable is specified using the following syntax:

```
BIOGEME_OBJECT.DRAWS = { 'U': 'UNIFORM'}
```

Note that the valid keywords are

- UNIFORM, for a uniform distribution on the interval $[0, 1]$,
- UNIFORMSYM, for a uniform distribution on the interval $[-1, 1]$,
- NORMAL, for a standard normal distribution, and
- TRUNCNORMAL, for a truncated standard normal distribution. The truncation is defined by the parameter NormalTruncation:

  ```
  BIOGEME_OBJECT.PARAMETERS['NormalTruncation'] = "1.96"
  ```

The integrand is defined by the following statement:

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

and the Monte-Carlo integration is obtained as follows:

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

The number of draws is defined by the parameter NbrOfDraws:

```
BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "20000"
```

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

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

and the standard error (11):

```
stderr = (sampleVariance / 20000.0)**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 = {'01 Simulated Integral': simulatedI,
            '02 Analytical Integral': trueI,
            '03 Sample variance': sampleVariance,
            '04 Std Error': stderr,
            '05 Error': error}

rowIterator('obsIter')

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

We obtain the following results:

Simulated Integral	1.72007
Analytical Integral	1.71828
Sample variance	0.240135
Std Error	0.00346508
Error	0.00178739

Remember that the true variance is 0.2420356075, and the true standard error is 0.0034787613. If we use ten times more draws, that is 200,000 draws, we obtain a more precise value:

Simulated Integral	1.71902
Analytical Integral	1.71828
Sample variance	0.24175
Std Error	0.00109943
Error	0.000739329

Remember that the true variance is 0.2420356075, and the true standard error is 0.0011000809. The complete specification file for PythonBiogeme 2.4 is available in Appendix A.1.

4 Variance reduction

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. We introduce two of them in this document: antithetic draws, and control variates. As the focus of this document is on PythonBiogeme 2.4, we urge the reader to read an introduction to variance reduction methods in simulation, for instance in Ross (2012).

4.1 Antithetic draws

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 $\mathrm{E}[Y] = \mathrm{E}[X_1] = \mathrm{E}[X_2] = \mathrm{E}[X]$, we can rewrite (1) as follows:

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

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 10000 draws from Y, the standard error decreases from 0.0034787613 down to 0.0006254996. Moreover, as we use recycled draws, we need only 10000 draws instead of 20000.

To apply this technique in PythonBiogeme 2.4, the integrand is defined as follows:

```
integrand = 0.5 * (exp(bioDraws('U')) + exp(1.0-bioDraws('U')))
```

and the number of draws reduced to 10000:

```
stderr = (sampleVariance / 10000.0)**0.5
BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "10000"
```

We obtain the following results:

Simulated Integral	1.71708
Analytical Integral	1.71828
Sample variance	0.00380337
Std Error	0.000616715
Error	-0.00120542

The reader can compare these values with the theoretical derivation presented above. The complete specification file for PythonBiogeme 2.4 is available in Appendix A.2.

4.2 Control variate

The control variate method reduces the variance by exploiting information from another random variable, correlated with $g(X)$, with a known mean.

Consider the random variable Y such that $\mathrm{E}[\mathsf{Y}] = \mu$. We define a new random variable Z as follows:

$$\mathsf{Z} = \mathsf{g}(\mathsf{X}) + \mathsf{c}(\mathsf{Y} - \mu) \tag{29}$$

where $\mathsf{c} \in \mathbb{R}$ is a parameter. By construction, $\mathrm{E}[\mathsf{Z}] = \mathrm{E}[\mathsf{g}(\mathsf{X})]$ for any c, so that draws from Z can be used instead of draws from $\mathsf{g}(\mathsf{X})$ for Monte-Carlo integration. Note that we do not need any assumption on g here. The idea is to identify the value of c that minimizes the variance of Z. We have

$$\mathrm{Var}[\mathsf{Z}] = \mathrm{Var}[\mathsf{g}(\mathsf{X}) + \mathsf{c}\mathsf{Y}] = \mathrm{Var}[\mathsf{g}(\mathsf{X})] + \mathsf{c}^2 \,\mathrm{Var}[\mathsf{Y}] + 2\mathsf{c}\,\mathrm{Cov}(\mathsf{g}(\mathsf{X}), \mathsf{Y}), \tag{30}$$

which is minimized for

$$\mathsf{c}^* = -\frac{\mathrm{Cov}(\mathsf{g}(\mathsf{X}), \mathsf{Y})}{\mathrm{Var}[\mathsf{Y}]}. \tag{31}$$

Therefore, we use for Monte-Carlo integration the random variable

$$\mathsf{Z}^* = \mathsf{g}(\mathsf{X}) - \frac{\mathrm{Cov}(\mathsf{g}(\mathsf{X}), \mathsf{Y})}{\mathrm{Var}\,\mathsf{Y}}(\mathsf{Y} - \mu), \tag{32}$$

with variance

$$\mathrm{Var}[\mathsf{Z}^*] = \mathrm{Var}[\mathsf{g}(\mathsf{X})] - \frac{\mathrm{Cov}(\mathsf{g}(\mathsf{X}), \mathsf{Y})^2}{\mathrm{Var}\,\mathsf{Y}} \leq \mathrm{Var}[\mathsf{g}(\mathsf{X})]. \tag{33}$$

Note that, as for antithetic draws, this technique exploits the correlation between two random variables. If Y is independent from $\mathsf{g}(\mathsf{X})$, no variance reduction is achieved.

In our example, $\mathsf{g}(\mathsf{X}) = e^{\mathsf{X}}$. If we select $\mathsf{Y} = \mathsf{X}$, we know that

$$\mathrm{E}[\mathsf{Y}] = \frac{1}{2} \text{ and } \mathrm{Var}[\mathsf{Y}] = \frac{1}{12}. \tag{34}$$

Moreover,

$$\mathrm{Cov}(\mathsf{g}(\mathsf{X}), \mathsf{Y}) = \mathrm{Cov}(e^{\mathsf{X}}, \mathsf{X}) = (3 - e)/2 = 0.1408590858. \tag{35}$$

Therefore, we obtain

$$\mathsf{c}^* = -\frac{\mathrm{Cov}(\mathsf{g}(\mathsf{X}), \mathsf{Y})}{\mathrm{Var}\,\mathsf{Y}} = -6(3 - e) = -1.6903090292, \tag{36}$$

and the variance of Z^* is 0.0039402229, which is much lower than the variance of x, that is 0.2420356075. It means that, for 20000 draws, the standard error is 0.0004438594, as opposed to 0.0034787613. With this method, only

326 draws are sufficient to achieve the same precision as the Monte-Carlo integration without control variate. Indeed,

$$\sqrt{\frac{0.0039402229}{326}} = 0.003476575. \tag{37}$$

This is a tremendous saving. The control variate method is invoked in PythonBiogeme 2.4 using the following statement:

```
simulatedI = MonteCarloControlVariate(integrand,bioDraws('U'),0.5),
```

where the second argument `bioDraws('U')` is Y, and the third, `0.5`, is μ. Note that, in addition to the output requested by the user, PythonBiogeme 2.4 also generates a report containing statistics on $g(X)$, Y and Z^*. In particular, it reports both the simulated value of Y and μ to detect any implementation error.

The results of the Monte-Carlo integration are:

Simulated Integral (E[Z^*])	1.71759
Simulated Integral (E[X])	1.72007
Analytical Integral	1.71828
Sample variance (Var[X])	0.239849
Std Error ($\sqrt{\text{Var}[Z^*]/20000}$)	0.000440564
Error	-0.00069233

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.3.

Finally, we present in Table 1 the results of the three methods, using different types of uniform draws as described in Section 2. For each technique, the standard errors for the three types of draws are comparable, with the antithetic draws achieving the best value, followed by the control variate. However, the precision actually achieved is much better for Halton, and even more for MLHS.

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

$$\int_0^1 \left(x - \frac{1}{2}\right)^2 dx = \frac{1}{12} \tag{38}$$

or

$$\int_{-2}^2 \left(e^{-x} + \frac{1}{2+\varepsilon-x}\right) dx = e^2 - e^{-2} + \log\frac{4+\varepsilon}{\varepsilon}, \tag{39}$$

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

	Pseudo	Halton	MHLS
Monte-Carlo	1.71902	1.71814	1.71829
Standard error	0.00109943	0.00109999	0.00110009
Actual error	0.000739329	-0.000145885	9.38555e-06
Antithetic	1.71708	1.71828	1.71828
Standard error	0.000616715	0.000625455	0.0006255
Actual error	-0.00120542	-2.27865e-06	-6.13416e-10
Control variate	1.71759	1.71828	1.71828
Standard error	0.000440564	0.000443827	0.000443872
Actual error	-0.00069233	-2.84647e-06	1.52591e-07

Table 1: Comparison of variants of Monte-Carlo integration on the simple example

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:

$$P_n(i|x, \theta, \mathcal{C}_n), \tag{40}$$

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 $i \in \mathcal{C}_n$:

$$U_{in}(x, \theta) = V_{in}(x, \theta) + \varepsilon_{in}(\theta), \tag{41}$$

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

$$P_n(i|x, \theta, \mathcal{C}_n) = \Pr(U_{in}(x, \theta) \geq U_{jn}(x, \theta), \forall j \in \mathcal{C}_n). \tag{42}$$

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:

$$P_n(i|x, \theta, \mathcal{C}_n) = \frac{e^{V_{in}(x,\theta)}}{\sum_{j \in \mathcal{C}_n} e^{V_{jn}(x,\theta)}}. \tag{43}$$

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

$$P_n(i|x, \theta_f, \theta_d, \mathcal{C}_n). \tag{44}$$

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:

$$P_n(i|x, \theta_f, \gamma, \mathcal{C}_n) = \int_\xi P_n(i|x, \theta_f, \xi, \mathcal{C}_n) f_{\theta_d}(\xi) d\xi, \tag{45}$$

where θ_f and γ must be estimated from data. The above integral has no analytical solution, 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`. Consider the following specification:

- Variables x: see variables in the data file and new variables defined in Section A.4.
- 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 **not** the standard deviation, and can be positive of 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 (45) can be calculated using numerical integration. It is done in PythonBiogeme 2.4 using the following procedure:

1. Mention that omega is a random variable:

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

2. Define its pdf:

```
density = normalpdf(omega).
```

   Make sure that the library distributions is loaded in order to use the function normalpdf, using the following statement:

```
from distributions import *
```

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):

```
integrand = bioLogit(V,av,CHOICE)
```

4. Calculate the integral:

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

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.4. The value of the choice model for first observation in the data file is

$$I = \int_{\xi} P_n(i|x, \theta_f, \xi, \mathcal{C}_n) f_{\theta_d}(\xi) d\xi = 0.637849835578. \quad (46)$$

Note that, in order ot obtain so many significant digits, we have used the following statement:

```
BIOGEME_OBJECT.PARAMETERS['decimalPrecisionForSimulation'] = "12"
```

To calculate the same integral with Monte-Carlo integration, we use the same syntax as described earlier in this document:

```
omega = bioDraws('B_TIME_RND')
BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "20000"
BIOGEME_OBJECT.DRAWS = { 'B_TIME_RND': 'NORMAL' }
B_TIME_RND = B_TIME + B_TIME_S * omega
integrand = bioLogit(V,av,CHOICE)
simulatedI = MonteCarlo(integrand)
```

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.5. Using the result of the numerical integration as the "true" value of the integral, We obtain the following results:

Simulated integral	0.637263
Numerical integration	0.63785
Sample variance	0.0299885
Std Error	0.000387224
Error	-0.000586483

We now apply the variance reduction methods. The antithetic draws described in Section 4.1 are generated as follows:

1. As we are dealing with draws from the normal distribution, the antithetic draw of x_r is $-x_r$. We create two versions of the parameter, one with the draw, and one with its antithetic:

```
B_TIME_RND = B_TIME + B_TIME_S * bioDraws('B_TIME_RND')
B_TIME_RND_MINUS = B_TIME - B_TIME_S * bioDraws('B_TIME_RND')
```

2. Consistently, we then generate two versions of the model:

```
V1_MINUS = ASC_TRAIN + \
           B_TIME_RND_MINUS * TRAIN_TT_SCALED + \
           B_COST * TRAIN_COST_SCALED
V2_MINUS = ASC_SM + \
           B_TIME_RND_MINUS * SM_TT_SCALED + \
           B_COST * SM_COST_SCALED
```

```
V3_MINUS = ASC_CAR + \
           B_TIME_RND_MINUS * CAR_TT_SCALED + \
           B_COST * CAR_CO_SCALED
V_MINUS = {1: V1_MINUS,
           2: V2_MINUS,
           3: V3_MINUS}
```

3. The integrand is the average of the integrands generated by the two versions of the model:

```
integrand_plus = bioLogit(V,av,CHOICE)
integrand_minus = bioLogit(V_MINUS,av,CHOICE)
integrand = 0.5 * (integrand_plus + integrand_minus)
```

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.6.

The control variate method, described in Section 4.2, requires an output of the simulation such that the analytical integral is known. We propose here to consider

$$\int_0^1 e^{V_{in}(x,\theta_f,\xi)} d\xi = \frac{e^{V_{in}(x,\theta_f,1)} - e^{V_{in}(x,\theta_f,0)}}{\partial V_{in}(x,\theta_f,\xi)/\partial\xi}, \tag{47}$$

if $\partial V_{in}(x,\theta_f,\xi)/\partial\xi$ does not depend on ξ. This integral is calculated by Monte-Carlo after recycling the uniform draws used to generate the normal draws for the original integration. We follow the following procedure:

1. We recycle the draws:

```
UNIFDRAW = bioRecycleDraws('B_TIME_RND')
```

2. We calculate the control variate integrand:

```
VCV = ASC_TRAIN + \
      (B_TIME + B_TIME_S * UNIFDRAW) * TRAIN_TT_SCALED + \
      B_COST * TRAIN_COST_SCALED
```

   Note that the derivative with respect to UNIFDRAW is

```
B_TIME_S * TRAIN_TT_SCALED$.
```

3. We provide the analytical value of the control variate integral:

```
VCV_ZERO = ASC_TRAIN + \
           B_TIME * TRAIN_TT_SCALED + \
           B_COST * TRAIN_COST_SCALED
VCV_ONE = ASC_TRAIN + \
          (B_TIME + B_TIME_S ) * TRAIN_TT_SCALED + \
          B_COST * TRAIN_COST_SCALED
VCV_INTEGRAL = (exp(VCV_ONE) - exp(VCV_ZERO)) / \
               (B_TIME_S * TRAIN_TT_SCALED)
```

4. We perform the Monte-Carlo integration:

```
simulatedI = MonteCarloControlVariate(integrand,\
                                      exp(VCV),\
                                      VCV_INTEGRAL)
```

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.7.

Table 2 provides the results of the Monte-Carlo integration using different variance reduction methods (none, antithetic and control variates), different uniform draws (pseudo, Halton and MLHS), and different number of draws.

We can observe the following:

- In terms of standard errors of the draws, the Monte-Carlo integration without variance reduction has a standard error about 7.5 times as large as the anthitetic version, and 2 times as large as the control variate.
- In terms of absolute error, when compared to the value provided by the numerical integration, the error of the Monte-Carlo integration without variance reduction is about the same for the pseudo draws, 12 times larger for the Halton draws, and 25 times larger for the MHLS draws, when compared to the antithetic draws. If it between 3 and 4 times larger, when compared to the control variates.
- There is no significant difference in terms of standard errors across the types of draws, irrespectively of the variance reduction method used.
- In terms of actual error, though, the Halton draws improves the precision of the output, and the MHLS even more.
- Using the antithetic draws with 1000 draws achieves a similar precision as no variance reduction with 20000 draws.
- Using the control variates with 2000 draws achieves a similar precision as no variance reduction with 20000 draws.
- Reducing the number of draws to 500 does not deteriorate much the precision for the antithetic draws.

It would be useful to perform the same experiment for some other observations in the data file. Such experiments can give useful insights to for the choice of the most appropriate integration technique. In the following, we compare some of these techniques for the maximum likelihood estimation of the parameters of the model.

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
BIOGEME_OBJECT.EXCLUDE = exclude
```

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

```
integrand = bioLogit(V,av,CHOICE)
prob = Integrate(integrand*density,'omega')
l = log(prob)
rowIterator('obsIter')
BIOGEME_OBJECT.ESTIMATE = Sum(l,'obsIter')
```

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.8.

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

```
prob = bioLogit(V,av,CHOICE)
l = mixedloglikelihood(prob)
rowIterator('obsIter')
BIOGEME_OBJECT.ESTIMATE = Sum(l,'obsIter')
```

where the statement l = mixedloglikelihood(prob) is equivalent to

```
integral = MonteCarlo(prob)
l = log(integral)
```

The complete specification file for PythonBiogeme 2.4 is available in Appendix A.9.

The following estimation results are presented:

- Table 4: numerical integration;
- Table 5: Monte-Carlo integration, no variance reduction, 2000 MHLS draws;
- Table 6: antithetic draws, 1000 MHLS draws;
- Table 7: control variates, 2000 MHLS draws;
- Table 8: Monte-Carlo integration, 500 MHLS draws;
- Table 9: antithetic draws, 250 MHLS draws;

- Table 10: control variates, 500 MHLS draws.

The final log likelihood in each case, as well as the estimation time are summarized in Table 3. In this experiment, when looking at the estimates, it seems that the MLHS draws provide relatively good precision, even for a lower number of draws, and with no variance reduction. Clearly, this result cannot be generalized, and should be investigated on a case by case basis. Note however that the default type of draws in PythonBiogeme 2.4 is MLHS, because it is performing particularly well in this example.

6 Conclusion

This document describes the variants of Monte-Carlo integration, and suggests how to perform some analysis using the SIMULATE operator of PythonBiogeme 2.4, that helps investigating the performance of each of them before starting a maximum likelihod 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.

	Draws	Pseudo	Halton	MHLS
		20000 draws		
Monte-Carlo	20000	0.637263	0.637923	0.637845
Standard error		0.000387224	0.000390176	0.000390301
Actual error		-0.000586483	7.35443e-05	-5.08236e-06
Antithetic	10000	0.638383	0.637856	0.63785
Standard error		5.13243e-05	5.24484e-05	5.24949e-05
Actual error		0.000533174	6.1286e-06	1.96217e-07
Control variate	20000	0.6377	0.637871	0.637848
Standard error		0.000176759	0.000179054	0.00017928
Actual error		-0.000149889	2.127e-05	-1.72413e-06
		2000 draws		
Antithetic	1000	0.638783	0.637965	0.637853
Standard error		5.05914e-05	5.17454e-05	5.24619e-05
Actual error		0.000933592	0.000114998	3.32666e-06
Control variate	2000	0.637876	0.637975	0.637835
Standard error		0.000551831	0.00056032	0.000567009
Actual error		2.66122e-05	0.000125218	-1.50796e-05
		500 draws		
Antithetic	250	0.639205	0.638459	0.637869
Standard error		5.17638e-05	4.97379e-05	5.23141e-05
Actual error		0.00135483	0.000609069	1.87082e-05
Control variate	500	0.637587	0.638158	0.637798
Standard error		0.00111188	0.00109022	0.00113287
Actual error		-0.000262395	0.000308626	-5.2274e-05

Table 2: Comparison of variants of Monte-Carlo integration on the mixture of logit example

Method	Draws	Log likelihood	Run time
Numerical	—	-5214.879	02:37
Monte-Carlo	2000	-5214.835	31:11
Antithetic	1000	-5214.899	39:26
Control variate	2000	-5214.835	42:11
Monte-Carlo	500	-5214.940	09:26
Antithetic	250	-5214.897	09:21
Control variate	500	-5214.940	08:59

Table 3: Final log likelihood and run time for each integration method

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.401	0.0656	-6.12	0.00
3	B_COST	-1.29	0.0863	-14.90	0.00
4	B_TIME	-2.26	0.117	-19.38	0.00
5	B_TIME_S	-1.65	0.125	-13.26	0.00

Summary statistics

Number of observations = 6768

Number of excluded observations = 3960

Number of estimated parameters = 5

Number of iterations = 13

Estimation time: $00:02:37$

$$
\begin{aligned}
\mathcal{L}(\beta_0) &= -7157.671 \\
\mathcal{L}(\hat{\beta}) &= -5214.879 \\
-2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 3885.585 \\
\rho^2 &= 0.271 \\
\bar{\rho}^2 &= 0.271
\end{aligned}
$$

Table 4: Estimation results with numerical integration

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.402	0.0658	-6.10	0.00
3	B_COST	-1.29	0.0864	-14.89	0.00
4	B_TIME	-2.26	0.117	-19.31	0.00
5	B_TIME_S	1.66	0.132	12.59	0.00

Summary statistics

Number of observations = 6768

Number of excluded observations = 3960

Number of estimated parameters = 5

Number of iterations = 9

Estimation time: $00:31:11$

$$\begin{aligned}\mathcal{L}(\beta_0) &= -6964.663\\ \mathcal{L}(\hat{\beta}) &= -5214.835\\ -2[\mathcal{L}(\beta_0)-\mathcal{L}(\hat{\beta})] &= 3499.656\\ \rho^2 &= 0.251\\ \bar{\rho}^2 &= 0.251\end{aligned}$$

Table 5: Estimation results with Monte-Carlo, no variance reduction, 2000 MHLS draws

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.402	0.0658	-6.10	0.00
3	B_COST	-1.29	0.0863	-14.89	0.00
4	B_TIME	-2.26	0.117	-19.31	0.00
5	B_TIME_S	1.66	0.132	12.59	0.00

Summary statistics

Number of observations = 6768

Number of excluded observations = 3960

Number of estimated parameters = 5

Number of iterations = 12

Estimation time: $00:39:26$

$$\begin{aligned}
\mathcal{L}(\beta_0) &= -7155.875 \\
\mathcal{L}(\hat{\beta}) &= -5214.899 \\
-2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 3881.952 \\
\rho^2 &= 0.271 \\
\bar{\rho}^2 &= 0.271
\end{aligned}$$

Table 6: Estimation results with antithetic draws, 1000 MHLS draws

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.402	0.0658	-6.10	0.00
3	B_COST	-1.29	0.0864	-14.89	0.00
4	B_TIME	-2.26	0.117	-19.31	0.00
5	B_TIME_S	1.66	0.132	12.59	0.00

Summary statistics
Number of observations = 6768
Number of excluded observations = 3960
Number of estimated parameters = 5
Number of iterations = 12
Estimation time: $00 : 42 : 11$

$$\begin{aligned}\mathcal{L}(\beta_0) &= -7155.867\\ \mathcal{L}(\hat{\beta}) &= -5214.835\\ -2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 3882.063\\ \rho^2 &= 0.271\\ \bar{\rho}^2 &= 0.271\end{aligned}$$

Table 7: Estimation results with control variates, 2000 MHLS draws

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.402	0.0658	-6.10	0.00
3	B_COST	-1.29	0.0864	-14.88	0.00
4	B_TIME	-2.26	0.117	-19.33	0.00
5	B_TIME_S	1.66	0.131	12.63	0.00

Summary statistics

Number of observations = 6768

Number of excluded observations = 3960

Number of estimated parameters = 5

Number of iterations = 12

Estimation time: $00 : 09 : 26$

$$\begin{aligned} \mathcal{L}(\beta_0) &= -7155.962 \\ \mathcal{L}(\hat{\beta}) &= -5214.940 \\ -2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 3882.044 \\ \rho^2 &= 0.271 \\ \bar{\rho}^2 &= 0.271 \end{aligned}$$

Table 8: Estimation results with Monte-Carlo integration, no variance reduction, 500 MHLS draws

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.402	0.0658	-6.11	0.00
3	B_COST	-1.29	0.0864	-14.88	0.00
4	B_TIME	-2.26	0.117	-19.33	0.00
5	B_TIME_S	1.66	0.131	12.61	0.00

Summary statistics

Number of observations = 6768

Number of excluded observations = 3960

Number of estimated parameters = 5

Number of iterations = 12

Estimation time: 00 : 09 : 21

$$\begin{aligned}
\mathcal{L}(\beta_0) &= -7155.877 \\
\mathcal{L}(\hat{\beta}) &= -5214.897 \\
-2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 3881.960 \\
\rho^2 &= 0.271 \\
\bar{\rho}^2 &= 0.271
\end{aligned}$$

Table 9: Estimation results with antithetic draws, 250 MHLS draws

Parameter number	Description	Coeff. estimate	Robust Asympt. std. error	t-stat	p-value
1	ASC_CAR	0.137	0.0517	2.65	0.01
2	ASC_TRAIN	-0.402	0.0658	-6.10	0.00
3	B_COST	-1.29	0.0864	-14.88	0.00
4	B_TIME	-2.26	0.117	-19.33	0.00
5	B_TIME_S	1.66	0.131	12.63	0.00

Summary statistics

Number of observations = 6768

Number of excluded observations = 3960

Number of estimated parameters = 5

Number of iterations = 12

Estimation time: $00:08:59$

$$\begin{aligned}
\mathcal{L}(\beta_0) &= -7155.962 \\
\mathcal{L}(\hat{\beta}) &= -5214.940 \\
-2[\mathcal{L}(\beta_0) - \mathcal{L}(\hat{\beta})] &= 3882.044 \\
\rho^2 &= 0.271 \\
\bar{\rho}^2 &= 0.271
\end{aligned}$$

Table 10: Estimation results with control variates, 500 MHLS draws

A Complete specification files

A.1 01simpleIntegral.py

```
#######################################
#
# File: 01simpleIntegral.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 11:41:13 2015
#
#######################################

from biogeme import *
from headers import *

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

trueI = exp(1.0) - 1.0

sampleVariance = \
  MonteCarlo(integrand*integrand) - simulatedI * simulatedI
stderr = (sampleVariance / 200000.0)**0.5
error = simulatedI - trueI

simulate = {'01 Simulated Integral': simulatedI,
            '02 Analytical Integral': trueI,
            '03 Sample variance': sampleVariance,
            '04 Std Error': stderr,
            '05 Error': error}

rowIterator('obsIter')

BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')
BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "200000"
BIOGEME_OBJECT.PARAMETERS['RandomDistribution'] = "PSEUDO"
__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1
BIOGEME_OBJECT.DRAWS = { 'U': 'UNIFORM'}
```

A.2 02antithetic.py

```
#######################################
#
# File: 02antithetic.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 12:21:10 2015
#
```

```
#######################################

from biogeme import *
from headers import *

integrand = 0.5 * (exp(bioDraws('U')) + exp(1.0-bioDraws('U')))
simulatedI = MonteCarlo(integrand)

trueI = exp(1.0) - 1.0

sampleVariance = \
  MonteCarlo(integrand*integrand) - simulatedI * simulatedI
stderr = (sampleVariance / 10000.0)**0.5
error = simulatedI - trueI

simulate = {'01_Simulated Integral': simulatedI,
            '02_Analytical Integral': trueI,
            '03_Sample variance': sampleVariance,
            '04_Std Error': stderr,
            '05_Error': error}

rowIterator('obsIter')

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

BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "10000"
__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1
BIOGEME_OBJECT.DRAWS = { 'U': 'UNIFORM'}
```

A.3 03controlVariate.py

```
#######################################
#
# File: 03controlVariate.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 12:24:25 2015
#
#######################################
#

from biogeme import *
from headers import *

integrand = exp(bioDraws('U'))
simulatedI = MonteCarloControlVariate(integrand,bioDraws('U'),0.5)

trueI = exp(1.0) - 1.0

```

```
error = simulatedI - trueI

simulate = {'01_Simulated Integral': simulatedI,
            '02_Analytical Integral': trueI,
            '05_Error': error}

rowIterator('obsIter')

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

BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "20000"
__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1
BIOGEME_OBJECT.DRAWS = { 'U': 'UNIFORM'}
```

A.4 05normalMixtureTrueAnalytical.py

```
########################################
#
# File: 05normalMixtureTrueAnalytical.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 18:50:11 2015
#
########################################

from biogeme import *
from headers import *
from distributions import *
from loglikelihood import *

#Parameters
ASC_CAR = 0.137
ASC_TRAIN = -0.402
ASC_SM = 0
B_TIME = -2.26
B_TIME_S = 1.66
B_COST = -1.29

# Define a random parameter, normally distributed,
# designed to be used for integration
omega = RandomVariable('omega')
density = 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)
TRAIN_COST_SCALED = \
  DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100)
SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0)
SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100)
CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100)
CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100)

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

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

# The choice model is a logit, with availability conditions
integrand = bioLogit(V,av,CHOICE)

```

```

analyticalI = Integrate(integrand*density,'omega')
simulate = {'Analytical': analyticalI}

rowIterator('obsIter')

BIOGEME_OBJECT.PARAMETERS['decimalPrecisionForSimulation'] = "12"
BIOGEME_OBJECT.SIMULATE = Enumerate(simulate,'obsIter')

__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1
```

A.5 06normalMixture.py

```
########################################
#
# File: 06normalMixture.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 18:37:37 2015
#
########################################

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

#Parameters
ASC_CAR = 0.137
ASC_TRAIN = -0.402
ASC_SM = 0
B_TIME = -2.26
B_TIME_S = 1.66
B_COST = -1.29

# Define a random parameter, normally distributed,
# designed to be used for integration
omega = bioDraws('B_TIME_RND')
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)
TRAIN_COST_SCALED = \
  DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100)
SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0)
SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100)
CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100)
CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100)

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

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

# The choice model is a logit, with availability conditions
integrand = bioLogit(V,av,CHOICE)
simulatedI = MonteCarlo(integrand)

trueI = 0.637849835578

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

```
stderr = (sampleVariance / 200000.0)**0.5
error = simulatedI - trueI

simulate = {'01 Simulated Integral': simulatedI,
            '02 Analytical Integral': trueI,
            '03 Sample variance': sampleVariance,
            '04 Std Error': stderr,
            '05 Error': error}

rowIterator('obsIter')

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

__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1

BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "20000"
BIOGEME_OBJECT.DRAWS = { 'B_TIME_RND': 'NORMAL' }
```

A.6 07normalMixtureAntithetic.py

```
#######################################
#
# File: 07normalMixtureAntithetic.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 19:14:42 2015
#
#######################################

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

#Parameters
ASC_CAR = 0.137
ASC_TRAIN = -0.402
ASC_SM = 0
B_TIME = -2.26
B_TIME_S = 1.66
B_COST = -1.29

# Define a random parameter, normally distributed,
# designed to be used for integration,
# and its antithetic.
B_TIME_RND = B_TIME + B_TIME_S * bioDraws('B_TIME_RND')
B_TIME_RND_MINUS = B_TIME - B_TIME_S * bioDraws('B_TIME_RND')

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

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

V1_MINUS = ASC_TRAIN + \
           B_TIME_RND_MINUS * TRAIN_TT_SCALED + \
           B_COST * TRAIN_COST_SCALED
V2_MINUS = ASC_SM + \
           B_TIME_RND_MINUS * SM_TT_SCALED + \
           B_COST * SM_COST_SCALED
V3_MINUS = ASC_CAR + \
           B_TIME_RND_MINUS * CAR_TT_SCALED + \
           B_COST * CAR_CO_SCALED

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

V_MINUS = {1: V1_MINUS,
             2: V2_MINUS,
             3: V3_MINUS}

# Associate the availability conditions with the alternatives

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}

# The choice model is a logit, with availability conditions
integrand_plus = bioLogit(V,av,CHOICE)
integrand_minus = bioLogit(V_MINUS,av,CHOICE)
integrand = 0.5 * (integrand_plus + integrand_minus)
simulatedI = MonteCarlo(integrand)

trueI = 0.637849835578

sampleVariance = \
  MonteCarlo(integrand*integrand) - simulatedI * simulatedI
stderr = (sampleVariance / 200000.0)**0.5
error = simulatedI - trueI

simulate = {'01 Simulated Integral': simulatedI,
            '02 Analytical Integral': trueI,
            '03 Sample variance': sampleVariance,
            '04 Std Error': stderr,
            '05 Error': error}

rowIterator('obsIter')

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

__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1

BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "10000"
BIOGEME_OBJECT.DRAWS = { 'B_TIME_RND': 'NORMAL' }
```

A.7 08normalMixtureControlVariate.py

```
#######################################
#
# File: 08normalMixtureControlVariate.py
# Author: Michel Bierlaire, EPFL
# Date: Sat Jul 25 18:50:11 2015
#
########################################

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

#Parameters
ASC_CAR = 0.137
ASC_TRAIN = -0.402
ASC_SM = 0
B_TIME = -2.26
B_TIME_S = 1.66
B_COST = -1.29

# Define a random parameter, normally distributed,
# designed to be used for Monte-Carlo simulation
B_TIME_RND = B_TIME + B_TIME_S * bioDraws('B_TIME_RND')

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

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

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

# The choice model is a logit, with availability conditions
integrand = bioLogit(V,av,CHOICE)

# Control variate

# Recycle the uniform draws used to generate the
#normal draws of B_TIME_RND
UNIFDRAW = bioRecycleDraws('B_TIME_RND')

# Utility function with the uniform draws instead of the normal.
VCV = ASC_TRAIN + \
       (B_TIME + B_TIME_S * UNIFDRAW) * TRAIN_TT_SCALED + \
       B_COST * TRAIN_COST_SCALED
# The analytical integral of exp(VCV) between 0 and 1
# is now calculated
VCV_ZERO = ASC_TRAIN + \
             B_TIME  * TRAIN_TT_SCALED + \
             B_COST * TRAIN_COST_SCALED
VCV_ONE = ASC_TRAIN + \
            (B_TIME + B_TIME_S ) * TRAIN_TT_SCALED + \
             B_COST * TRAIN_COST_SCALED
VCV_INTEGRAL = (exp(VCV_ONE) - exp(VCV_ZERO)) / \
                  (B_TIME_S * TRAIN_TT_SCALED)


simulatedI = MonteCarloControlVariate(integrand, \
                                        exp(VCV), \
                                                 VCV_INTEGRAL)

trueI = 0.637849835578

error = simulatedI - trueI

simulate = {'01 Simulated Integral': simulatedI,
            '02 Analytical Integral': trueI,
            '05 Error': error}

rowIterator('obsIter')

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

__rowId__ = Variable('__rowId__')
BIOGEME_OBJECT.EXCLUDE = __rowId__ >= 1

BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "20000"
BIOGEME_OBJECT.DRAWS = { 'B_TIME_RND': 'NORMAL' }
```

A.8 11estimationNumerical.py

```
########################################
#
# File: 11estimationNumerical.py
# Author: Michel Bierlaire, EPFL
# Date: Thu Jul 30 10:40:49 2015
#
########################################

from biogeme import *
from headers import *
from distributions import *
from loglikelihood import *
from statistics import *

#Parameters to be estimated
# Arguments:
#   1  Name for report. Typically, the same as the variable
#   2  Starting value
#   3  Lower bound
#   4  Upper bound
#   5  0: estimate the parameter, 1: keep it fixed

ASC_CAR = Beta('ASC_CAR',0,-10,10,0)
ASC_TRAIN = Beta('ASC_TRAIN',0,-10,10,0)
ASC_SM = Beta('ASC_SM',0,-10,10,1)
B_TIME = Beta('B_TIME',0,-10,10,0)
B_TIME_S = Beta('B_TIME_S',9,-10,10,0)
```

```
B_COST = Beta('B_COST',0,-10,10,0)

# Define a random parameter, normally distributed,
# designed to be used for  simulation
omega = RandomVariable('omega')
 density = 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)
TRAIN_COST_SCALED = \
  DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100)
SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0)
SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100)
CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100)
CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100)

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

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

# The choice model is a logit, with availability conditions
integrand = bioLogit(V,av,CHOICE)
prob = Integrate(integrand*density,'omega')
l = log(prob)

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

# Define the likelihood function for the estimation
BIOGEME_OBJECT.ESTIMATE = Sum(l,'obsIter')

# All observations verifying the following expression will not be
# considered for estimation
# The modeler here has developed the model only for work trips.
# Observations such that the dependent variable CHOICE is 0
# are also removed.
exclude = (( PURPOSE != 1 ) * (  PURPOSE   !=  3  ) + \
           ( CHOICE == 0 )) > 0

BIOGEME_OBJECT.EXCLUDE = exclude

# Statistics

nullLoglikelihood(av,'obsIter')
choiceSet = [1,2,3]
cteLoglikelihood(choiceSet,CHOICE,'obsIter')
availabilityStatistics(av,'obsIter')

BIOGEME_OBJECT.PARAMETERS['RandomDistribution'] = "MLHS"

BIOGEME_OBJECT.PARAMETERS['optimizationAlgorithm'] = "BIO"
BIOGEME_OBJECT.FORMULAS['Train utility'] = V1
BIOGEME_OBJECT.FORMULAS['Swissmetro utility'] = V2
BIOGEME_OBJECT.FORMULAS['Car utility'] = V3
```

A.9 12estimationMonteCarlo.py

```
########################################
#
```

```
# File: 12estimationMonteCarlo.py
# Author: Michel Bierlaire, EPFL
# Date: Thu Jul 30 10:45:21 2015
#
#######################################

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

#Parameters to be estimated
# Arguments:
#   1  Name for report. Typically, the same as the variable
#   2  Starting value
#   3  Lower bound
#   4  Upper bound
#   5  0: estimate the parameter, 1: keep it fixed

ASC_CAR = Beta('ASC_CAR',0,-10,10,0)
ASC_TRAIN = Beta('ASC_TRAIN',0,-10,10,0)
ASC_SM = Beta('ASC_SM',0,-10,10,1)
B_TIME = Beta('B_TIME',0,-10,10,0)
B_TIME_S = Beta('B_TIME_S',9,-10,10,0)
B_COST = Beta('B_COST',0,-10,10,0)

# Define a random parameter, normally distirbuted,
# designed to be used for Monte-Carlo simulation
B_TIME_RND = B_TIME + B_TIME_S * bioDraws('B_TIME_RND')

# 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)
TRAIN_COST_SCALED = \
  DefineVariable('TRAIN_COST_SCALED', TRAIN_COST / 100)
```

```
SM_TT_SCALED = DefineVariable('SM_TT_SCALED', SM_TT / 100.0)
SM_COST_SCALED = DefineVariable('SM_COST_SCALED', SM_COST / 100)
CAR_TT_SCALED = DefineVariable('CAR_TT_SCALED', CAR_TT / 100)
CAR_CO_SCALED = DefineVariable('CAR_CO_SCALED', CAR_CO / 100)

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

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

# The choice model is a logit, with availability conditions
prob = bioLogit(V,av,CHOICE)
l = mixedloglikelihood(prob)

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

# Define the likelihood function for the estimation
BIOGEME_OBJECT.ESTIMATE = Sum(l,'obsIter')

# All observations verifying the following expression will not be
# considered for estimation
# The modeler here has developed the model only for work trips.
# Observations such that the dependent variable CHOICE
# is 0 are also removed.
exclude = (( PURPOSE != 1 ) * (  PURPOSE   !=  3  ) + \
           ( CHOICE == 0 )) > 0

```

```
BIOGEME_OBJECT.EXCLUDE = exclude

# Statistics

nullLoglikelihood(av,'obsIter')
choiceSet = [1,2,3]
cteLoglikelihood(choiceSet,CHOICE,'obsIter')
availabilityStatistics(av,'obsIter')

BIOGEME_OBJECT.PARAMETERS['NbrOfDraws'] = "2000"
BIOGEME_OBJECT.PARAMETERS['RandomDistribution'] = "MLHS"

BIOGEME_OBJECT.PARAMETERS['optimizationAlgorithm'] = "BIO"
BIOGEME_OBJECT.DRAWS = { 'B_TIME_RND': 'NORMAL' }
BIOGEME_OBJECT.FORMULAS['Train utility'] = V1
BIOGEME_OBJECT.FORMULAS['Swissmetro utility'] = V2
BIOGEME_OBJECT.FORMULAS['Car utility'] = V3
```

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.