Technische Universität München
Lehrstuhl für Informatik V
Dr. Tobias Neckel
M. Sc. Ionuț Farcaș

SS 2017

May 17, 2017

Algorithms for Uncertainty Quantification

Tutorial 4: More advanced Monte Carlo techniques

In this worksheet, we focus on more advanced sampling techniques.

Relative error

Since computational science (generally) relies on discrete versions of continuous quantities/models, we want to estimate the resulting error. To this end, there are several error indicators that can be employed. Among these, there is the *relative* error. Assuming that `approx` denotes the approximation and `ref` denotes the true (reference) value of the underlying quantity that we try to approximate, the relative error is

$$\text{rel}_{\text{error}} = \left|1 - \frac{\text{approx}}{\text{ref}}\right|. \tag{1}$$

However, note that Eq. (1) relies on the true value of the quantity that we try to approximate. The true value is available in a (very) limited number of situations. Therefore, most often, the "true" value is considered to be the most accurate approximation, e.g. the approximation on the finest grid or with the largest number of samples.

Improving standard Monte Carlo sampling

In the lecture, we saw that there exist several methods to improve the accuracy of standard Monte Carlo sampling. Remember that when approximating $I = \int_0^1 f(x)dx$ via standard standard Monte Carlo sampling, the associated standard error is

$$\hat{\sigma}_{\hat{I}_f} = \frac{\hat{\sigma}_f}{\sqrt{N}}, \quad \hat{\sigma}_f^2 = \frac{1}{N-1}\sum_{i=1}^{N}(f(U_i) - \hat{I}_f)^2, \quad \{U_i\}_{i=1}^N = \text{ samples from } \mathcal{U}(0,1). \tag{2}$$

One approach to reduce $\hat{\sigma}_{\hat{I}_f}$ is to reduce $\hat{\sigma}_f$. To this end, in the lecture we discussed the following strategies (see the lecture slides for more details)

- antithetic sampling
- stratified sampling
- control variates
- import sampling

Assignment 1

Consider $f : [0, 1] \to \mathbb{R}$, $f(x) = \exp(x)dx$. Compute analytically $\int_0^1 f(x)dx$; this will be your reference result. Estimate $\int_0^1 f(x)dx$ using $N = [1000, 10000, 100000, 1000000]$ samples and

- standard Monte Carlo sampling
- control variates with $\phi(x) = x, \phi(x) = 1 + x$
- importance sampling with a `beta` distribution $g_X(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}$, where $\mathrm{B}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-x}dx$. Choose $\alpha = 5, \beta = 1$ and $\alpha = 0.5, \beta = 0.5$ (*Hint: for the beta distribution: to sample, you could use* `numpy.random.beta(a, b, size=None)`*; to evaluate the PDF, you could use* `scipy.stats.beta`.)

Compute the relative error for each strategy using Eq. (1). Plot the errors in a loglog plot. Run the code several times. What do you observe?

Quasi-Monte Carlo sampling for UQ

Besides reducing the variance of pseudo-random number-based approaches, an alternative way to improve standard Monte Carlo sampling is employing different number generation techniques, such as deterministic sequences. A prominent example is the low-discrepancy sequences, such as Halton or Sobol sequences; in the lecture, we discussed Halton sequences. Employing low-discrepancy sequences leads to the so-called quasi-Monte Carlo (QMC) method. The error of QMC is $\mathcal{O}(\frac{\log(N)^d}{N})$ (compared to standard Monte Carlo sampling, whose error reads $\mathcal{O}(\frac{1}{\sqrt{N}})$).
In chaospy, it is very easy to generate QMC samples. For example, if your distribution is $\mathcal{U}(0, 1)$ and you want to generate 1000 Halton sequence-based samples, the syntax is

```
import chaospy as cp

distr = cp.Uniform()
samples = distr.samples(size=1000, rule='H')
```

Quasi-Monte Carlo sampling in the model problem

Consider the model problem that we used so far, the linear damped oscillator

$$\begin{cases} \frac{d^2y}{dt^2}(t) + c\frac{dy}{dt}(t) + ky(t) = f\cos(\omega t) \\ y(0) = y_0 \\ \frac{dy}{dt}(0) = y_1, \end{cases} \tag{3}$$

where c is the damping coefficient, k the spring constant, f the forcing amplitude, ω the frequency, y_0 represents the initial position, whereas y_1 is the initial velocity.
Considering $t \in [0, 20]$, $\Delta t = 0.01$, $c = 0.5$, $k = 2.0$, $f = 0.5$, $\omega = 1.0$, $y_0 = 0.5$, $y_1 = 0.0$, use the `odeint`[1] function from `scipy.integrate` to discretize the model from Eq. (3).

Assignment 2.1

Assume that $\omega \sim \mathcal{U}(0.95, 1.05)$. For this setting, the reference results are $\mathbb{E}_{\text{ref}} = -0.43897$ and $\text{Var}_{\text{ref}} = 0.00019653$. Write a `python` program and employ the `chaospy` library to propagate the uncertainty in ω through the model in Eq. (3). Use both standard Monte Carlo sampling and quasi-Monte Carlo based on Halton sequences with N = [10, 100, 1000, 10000] samples. Compute the mean and variance of $y(10)$ and their relative error with respect to the reference results using Eq. (1). Plot the errors in a loglog plot. What do you observe?

Assignment 2.2

Repeat the above experiment assuming that $\omega \sim \mathcal{U}(0.8, 1.2)$. In this case, the reference results are $\mathbb{E}_{\text{ref}} = -0.26079$ and $\text{Var}_{\text{ref}} = 0.02599844$. What do you observe?

[1] see `https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.integrate.odeint.html`