TUM
Algorithms for UQ
Start date: May 14th, 2025
End date: June 1st, 2025

Kislaya Ravi, Dr. Tobias Neckel
Ivana Jovanovic Buha
Iryna Burak

Exercise 1: Monte Carlo sampling and variance reduction techniques

Submission

Submit your code and a `.pdf` with your report on Moodle as a `.zip` file named `uq_exercise1_<groupnumber>.zip`. Please add **name** and **matriculation number** of all the group members to the report. Remember to properly describe your approach and analyze the results in the text. Submitting only code is not enough!

1 Introduction to Monte Carlo sampling estimator

We saw in the lecture that the sampling estimator for the mean value of a random variable X is given as

$$\mathbb{E}[X] \approx \bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i, \quad X_i \text{ are i.i.d. samples,}$$

while the unbiased variance estimator is given as

$$\mathbb{V}[X] \approx \hat{\sigma}_X^2 = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X})^2.$$

We will show in *Tutorial 4* that the associated root mean squared error (RMSE or standard error) for the *mean* estimator is

$$\hat{\sigma}_{\bar{X}} = \frac{\hat{\sigma}_X}{\sqrt{N}}. \tag{1}$$

Note on plotting. When plotting errors vs. the number of samples, it is useful to use a log-log plot. This is because the errors are often proportional to $N^{-1/2}$ which corresponds to a straight line in a log-log plot.

Assignment 1.1

Assume that a vector G represents a set of exam grades for a course. First, compute the mean and the variance of G using basic `Python` functionality. Then, compute the same qunatities of interest using `numpy`'s functions. Do you get identical resutls? If not, why?
Fill out the missing code in `assignment_1.1.py` and discuss your findings. The values of G are provided in `G.txt`.

Assignment 1.2

Now, we can apply the same idea to estimate mean and covariance of a multivariate random variable X. For this, we follow *Task 5, Tutorial 2* and consider a bivariate normal distribution with the given mean $\mu = [-0.4, 1.1]^T$ and covariance matrix $V = [[2, 0.4], [0.4, 1]]$.
First, generate $N = [10, 100, 1000, 10000]$ samples from the given bivariate normal distribution. Next, use Monte Carlo sampling over the N samples to approximate the mean values and covariance matrix. To compute the mean and covariance values, one computes the entries dimensionwise, i.e., the j-th entry of the mean vector is given by

$$\bar{X}_j = \frac{1}{N} \sum_{n=1}^{N} X_{j,n}, \tag{2}$$

where $X_{j,n}$ is the j-th component of the n-th sample, and the (i, j)-th entry of the covariance matrix is

$$cov_{i,j} = \frac{1}{N-1} \sum_{n=1}^{N} (X_{i,n} - \bar{X}_i)(X_{j,n} - \bar{X}_j). \tag{3}$$

Report the mean and covariance values for an increasing number of samples. As you know the true values of the mean and covairiance, plot the aboslute error of your estimator. It is enough if you only report results for one mean value and for two covariance values: one diagonal, one off-diagonal. Additionally, plot the RMSE (eq. (1)) for the mean estimator. Can we use the same formula for the covariance estimator? Why?
Fill out the missing code in `assignment_1.2.py` and discuss your findings.

Monte Carlo integration

In this task, we will use Monte Carlo sampling to approximate several integrals. The Monte Carlo approximation of $I = \int_a^b f(x)dx$ reads

$$I_f \approx \hat{I}_f = \frac{1}{N} \sum_{i=1}^{N} f(U_i), \tag{4}$$

where $U_i \sim \mathcal{U}(a, b)$.

Assignment 2.1

Your goal is to estimate $F = \int_0^1 \sin(x)dx$ using eq. (4) and compute the associated RMSE of the mean estimator using eq. (1) for

$$N = [10, 100, 1000, 10000].$$

You can use the `chaospy` library to generate random variables (see Appendix for an introduction). To evaluate the results, you should calculate the exact value of F analytically, compute the exact error $\epsilon = |F - \hat{I}|$ for each N, and compare it to the corresponding RMSE.
Fill out the missing code in `assignment_2.py`, plot both the exact error and RMSE, and discuss your findings.

Assignment 2.2

Similarly to the previous task, here you are asked to estimate $F = \int_2^4 \sin(x)dx$ using Monte Carlo integration. However, there are now two approaches to sample from the uniform distribution $\mathcal{U}(2, 4)$:

- Draw samples directly as $X \sim \mathcal{U}(2, 4)$.
- Draw samples as $X \sim \mathcal{U}(0, 1)$ and apply a linear transformation to obtain samples from $\mathcal{U}(2, 4)$.

Adapt the code in `assignment_2.py` and report your results similarly to the previous task. Does the RMSE have the same interpretration as in Assignment 2.1?

Improving standard Monte Carlo sampling

In *Lecture 4*, you saw several methods to improve the accuracy of standard Monte Carlo sampling. Remember that when approximating $I = \int_0^1 f(x)dx$ via standard Monte Carlo sampling, the associated standard error is

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

One approach to reduce $\hat{\sigma}_{\hat{I}_f}$ is to reduce $\hat{\sigma}_f$, and we discussed the following strategies of doing this:

- antithetic sampling
- stratified sampling
- control variates
- importance sampling

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 with the error of $\mathcal{O}\left(\frac{\log(N)^d}{N}\right)$. You will work with one of these sequence in *Assignment 4* of this exercise.

Assignment 3

Consider $f : [0,1] \to \mathbb{R}$, $f(x) = \exp(x)$. Your goal is to estimate $\int_0^1 f(x)dx$ via several variations of the Monte Carlo method.
Use $N = [10, 100, 1000, 10000]$ samples with

- standard Monte Carlo sampling.
- control variates with $\phi_1(x) = x$, $\phi_2(x) = 1 + x$, and $\phi_3(x) = 1 + x + \frac{x^2}{2}$.

- importance sampling with a Beta distribution with parameters α and β. Consider two options: $\alpha = 5, \beta = 1$ and $\alpha = 0.5, \beta = 0.5$.

Fill out the missing code in `assignment_3.py`, report the absolute error for each strategy using the true value of the integral, and discuss your findings.

Monte Carlo sampling for forward propagation UQ

Assume that we have a model $G : \mathbb{X} \to \mathbb{Y}$, where $\mathbb{X}$, $\mathbb{Y}$ are some domains. For example, G can be given in terms of an ordinary or partial differential equation. In the standard deterministic setting, the inputs x of G (e.g., physical parameters, initial and boundary conditions, etc.) are considered to be known with certainty. Therefore, the problem consists in computing $y = G(x)$ by evaluating G once. In the case when G is a differential equation, "evaluating G" can be read as solving the given equation with a numerical solver.
However, in most realistic scenarios, the inputs of a physical or engineering model are not know with certainty. Let's assume that the input x is decomposed into $[X_r, x_d]$, where X_r contains the stochastic parameters and x_d contains the deterministic ones. Hence, in this setting, the problem consists in propagating the uncertainty in X_r through G and computing a quantity of interest $F(G([X_r, x_d]))$. For example, $F(\cdot) = \mathbb{E}[\cdot]$ if we are interested in the mean value. Note that no matter what method we employ for the uncertainty propagation, G needs be evaluated multiple times in this setting. Then, we can compute an estivation of $F(G([X_r, x_d]))$ by a corresponding estimator $\hat{F}$.
Good news, we can use Monte Carlo sampling for this task as well! The algorithm then contains the following steps:

1. Model X_r as a random vector/variable with a distibution $\mathcal{D}$
2. Draw N samples $\{x_r^i\}_{i=1}^N$ from $\mathcal{D}$
3. Evaluate $y_i = G([x_r^i, x_d]), i = 1, \ldots, N$
4. Estimate the quantity of interest $q = \hat{F}(y_1, \ldots, y_N)$

In the previous assignments, we used the absolute error as a measure of accuracy. However, in the context of physical processes, the relative error is

often more informative. The relative error is defined as

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

where `approx` denotes the approximation and `ref` denotes the true (or reference) value of the underlying quantity that we try to approximate.

Assignment 4

Here we come back to the linear damped oscillator model from *Tutorial 1* defined as

$$\begin{cases} \frac{d^2y}{dt^2}(t) + c\frac{dy}{dt}(t) + ky(t) = f\cos(\omega t) \\ y(0) = y_0 \\ \frac{dy}{dt}(0) = y_1, \end{cases} \tag{5}$$

where c is the damping coefficient, k is the spring constant, f is the forcing amplitude, ω is the frequency, y_0 is the initial position, and y_1 is the initial velocity. We are interested in solving this system for $t \in [0, 10]$ with a time step $\Delta t = 0.01$.
First, consider the deterministic scenario from *Tutorial 1* where all the parameters are fixed as

$$c = 0.5, \quad k = 2.0, \quad f = 0.5, \quad \omega = 1.0, \quad y_0 = 0.5, \quad y_1 = 0.0.$$

Next, consider a more realistic setting where we only know the value of ω up to some uncertainty. Here, we assume that $\omega \sim \mathcal{U}(0.95, 1.05)$. Your task is to propagate this unvertainty through the system using two approaches:

- Standard Monte Carlo sampling
- Quasi-Monte Carlo based on Halton sequences

As there the "true" statistics are not known, we should compare the results to a reference solution given in `oscillator_ref.txt` as

$$\mathbb{E}_{\text{ref}}[y(10)] = -0.43893703, \quad \mathbb{V}_{\text{ref}}[y(10)] = 0.00019678$$

This solution was computed with $N_{\text{ref}} = 1000000$ samples, which took significant computational resources, so you don't need to recompute it. In reality, the "true" value of the quantity of interest is usually unknown, so 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.
Employ both methods specified above to propagate the uncertainty in ω through the model with $N = [10, 100, 1000, 10000]$ samples. To analyse the convergence, compute the mean and the variance of $y(10)$ and compare the results to the reference values. As a visualization, plot the relative error over the number of samples. Additionally, report a plot where you show ten solution trajectories, i.e., ten solutions of the ODE plotted over time $t \in [0, 10]$ which result from ten different samples of ω with the standard Monte Carlo method. Fill out the missing code in `assignment_4.py` and discuss your findings.

Appendix: Short introduction to chaospy

The online documentation and a user quide is available at `http://chaospy.readthedocs.io/en/master/`.

```
import chaospy as cp

# Define the distribution of interest.
a, b = -1, 2
distr = cp.Uniform(a, b)
# Sample from the defined distribution.
sample_size = 1000
samples = distr.sample(size=sample_size)
# To ensure reproducibility, pass the 'seed' parameter.
# You can also provide additional parameters to control
# the generation.
samples = distr.sample(size=sample_size, seed=42)
```