Practice 2

MLCG 2021-2022

Ricardo Marques

ricardo.marques@ub.edu

Universitat de Barcelona – Master in Artificial Intelligence (MAI)

Introduction

In this practice you will manipulate the Bayesian Monte Carlo method. Similarly to Practice 2 (on classic Monte Carlo), this practice in divided in two parts: Part I, where you be able to get familiar with the BMC method by implementing in the appWorkBench; and Part II where you must apply BMC for rendering. To accomplish these tasks, you should download the file GaussianProcess.py which is available in the campus virtual and integrate it in your python project.

For the sake of simplicity, the provided code as well as the following of this text assumes that the prior mean function of the Gaussian Process is given by $f(\omega) = 0$. This allows a certain degree of simplification in the Bayesian Monte Carlo equations (compared to what you saw in the theory class), yielding:

$$\hat{I}_{BMC} = z^t \, Q^{-1} \, Y$$

Note that the above expression can be explicitly written in terms of the sample weights w:

$$\hat{I}_{BMC} = w \, Y = \sum_{i=1}^{N} w_i \, Y_i,$$

where $w = [w_1, \dots, w_N] = z^t \, Q^{-1}$. Furthermore, in this practice we will also assume that all the terms of our integrand are unknown, meaning that the known part of the integrand $p(x)$ in the BMC formalism is specified as $p(x) = 1$. In this case, the vector z becomes simpler than what you saw in the theory class, yielding:

$$\mathrm{z} = \left(\int k(\omega_1, \, \omega) \, d\omega, \; \dots, \int k(\omega_N, \, \omega) \, d\omega \right)$$

All these simplifications will make the task of getting familiar with the BMC estimator easier.

Part I: Bayesian Monte Carlo in WorkBench

The Gaussian Process module

In this Part I, you will have the first contact with the GaussianProcess.py file and its content. In the following, you can find a brief description of its most important functions.

CovarianceFunction: this abstract base class (ABC) serves as parent class for any covariance function one wish to use in the Gaussian Process (GP) framework. Examples of a covariance function are the squared exponential (which you saw in the theory class), or the Sobolev covariance function that is also provided in the code. In this course you will not be asked to implement any covariance function, you will only need to know how to use one.

GaussianProcess: this is the core of the GaussianProcess.py file. Please also go in detail through the comments which are present in the file GaussianProcess.py to understand the main methods of this class.

Assignment 3.1: Complete the Gaussian Process class

Your first assignment is to complete the missing code in the GaussianProcess class provided in the file GaussianProcess.py. This implies:

- Completing the method *compute_inv_Q()*
- Completing the method *compute_z()*
- Completing the method *compute_integral_BMC()*

Please refer to the code for more detailed instructions.

Assignment 3.2: Estimate the value of a hemispherical integral using BMC

Once your GP class is completed, your second task is to apply the BMC method to solve the sample integrals that you have solved in the previous practice. For example:

$$I = \int_{\Omega} \cos\theta_i \, \mathrm{d}\omega_i$$

You should use the same framework you have developed in the Part I of previous practice, and add another error line in the resulting plot corresponding to the BMC method. If everything goes well, the BMC method should perform better (i.e., yields a smaller error) than its Monte Carlo counterpart. Fig. 1 shows an example of the expected outcome. Note that, as opposed to the Monte Carlo experiments made in Practice 2, **you should not average the BMC estimate error over a large number of BMC estimates to obtain a stable result** (10 estimates are sufficient). Note also that, for the sake of time, the maximum number of samples studied should not be much larger than 100.

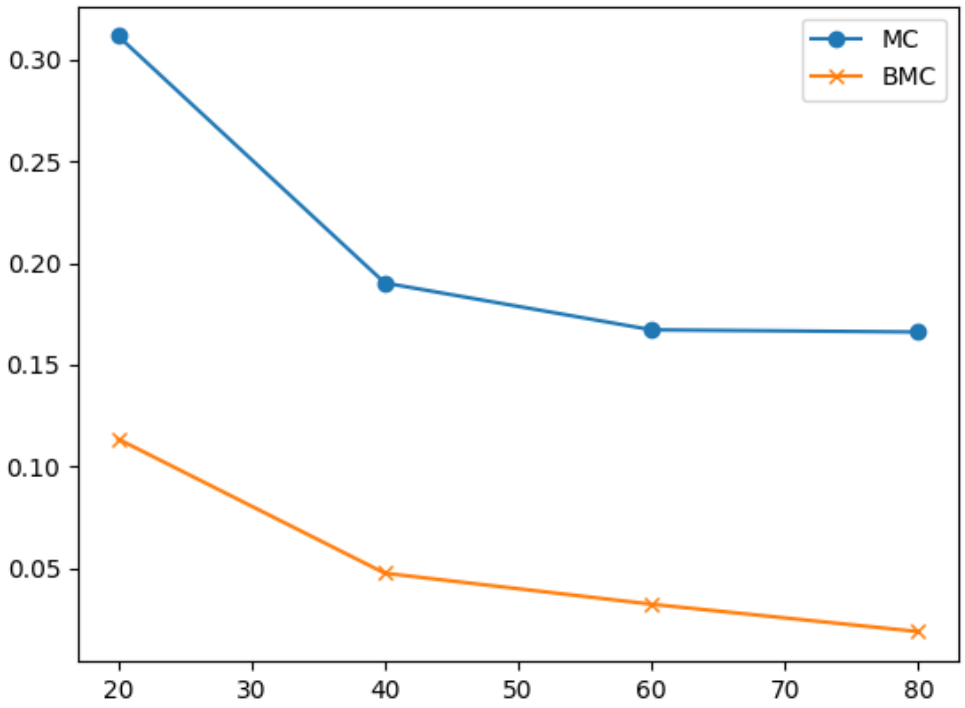

Figure 1. Average error as a function of the number of samples for classic Monte Carlo (MC) and Bayesian Monte Carlo (BMC). For MC, an average over the error of 100 estimates is shown. For the case of BMC, the average error was computed using only 10 estimates. Note the drastic reduction of the average error when using the BMC estimator, compared to classic MC.

Part II – Monte Carlo for rendering (appRenderer)

Assignment 3.3: Estimate the value of the illumination integral using BMC

Assignment 3.3.1: A direct application

In this part, you should apply the BMC estimator to compute a photo-realistic image by completing the class BayesianMonteCarloIntegrator. You will notice that a direct application of the method, where a new GP is created for each integral estimate is infeasible in practice due to time constraints. Why is that the case? What is the costly operation which is slowing down the computation time?

The answer to the above questions is “due to the computation of the sample weights” which, if you recall, is given by the product of two terms (see section Introduction above):

1. The inverse of the covariance matrix Q
2. The z vector

In the following, you will implement different strategies to overcome this problem.

Assignment 3.3.1 Using a single GP for the whole image

To solve the problem of the excessive computation time, one must reduce the number of times the sample weights are computed by transferring this operation to a pre-computation step. To this end, it is important to note that both these terms (Q and z) only depend on the sample positions. More precisely, these terms depend on the relative position between the samples. This means that Q and z are invariant to rotations of the whole sample set, since the rotation of the full sample set does not affect the distance between the samples.

Profiting from the invariance of the sample weights to the rotation operation, we can simply pre-compute the sample weights for a single sample set and then, for each integral estimate, rotate the sample set so that its up axis is aligned with the normal at the surface of the point where we wish to evaluate the illumination integral. This way, we compute the sample weights only once when generating the full image. Such an optimization can be implemented as follows:

1. Create a GP process before creating the *BayesianMonteCarloIntegrator* object
2. Provide the GP with a set of N samples positions (which causes the sample weights to be computed)
3. Pass the resulting GP to the constructor (initializer) of the *BayesianMonteCarloIntegrator*, which should store it as an attribute
4. During rendering, in the *compute_color()* method, fetch the sample positions of the GP and rotate it according to the normal at the current intersection point
5. Sample the integrand using the rotated sample set
6. Add the resulting sample values to the GP (note that these values will be different for each integral estimate)
7. Perform a BMC integral estimate

With the above approach, the sample weights are computed only once in a pre-computation step, which will drastically reduce the computation time of BMC. The GP is the locally adapted at by passing it the vector of sample values collected at each intersection point. However, due to the use of the same sample pattern for all pixels, some artefacts (i.e., undesired unnatural patterns) can appear in the final image. In the following, we will see how these can be fixed.

Assignment 3.3.2 Decorrelating the error (noise)

The first measure you can take to decorrelate the estimation error is to apply a random rotation around the *y-axis* to the original sample set, before centering the sample set around the normal at the current intersection point. This will make the sampled directions in neighboring pixels become more distinct among themselves, hence contributing to decorrelate the error of the estimates. The random rotation can be achieved by using the *rotate_around_y()* method which, given a rotation angle and a sample position, rotates the sample around the up axis of the sample set.

Finally, once you implement the above strategy, you might notice that some sampling artifacts might still persist in the final image. To avoid this situation, explore the possibility of precomputing a small set of GPs (with different samples positions) instead of using a single one. Then, during rendering, randomly select one of the GPs available to perform the BMC estimate. The final result should be similar (even if not exactly equal) to the one shown in Fig. 2 below.

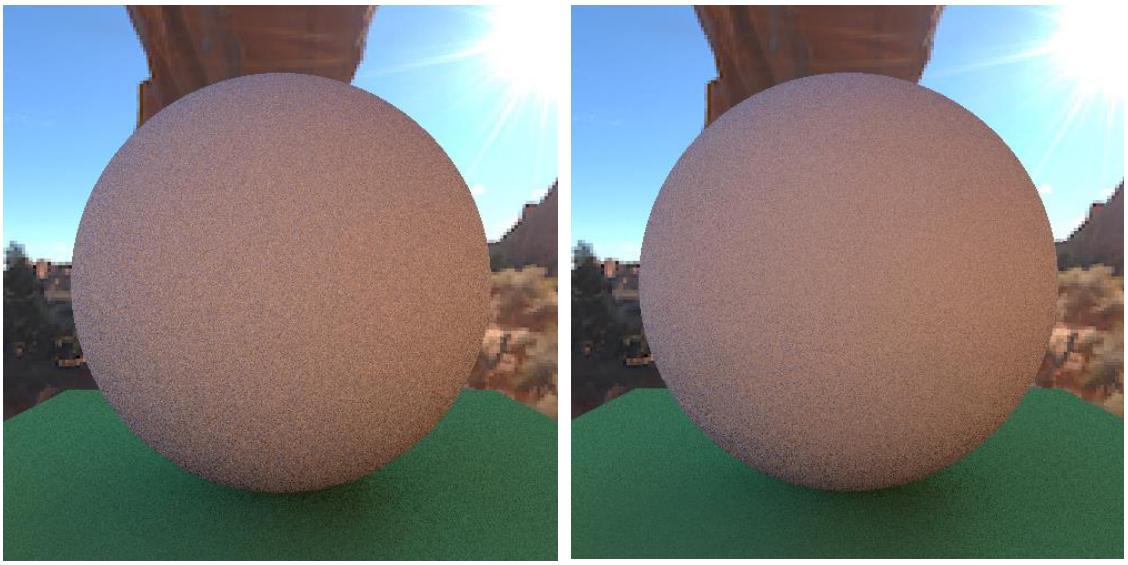

Figure 2. Sphere scene rendered with classic Monte Carlo (left) and Bayesian Monte Carlo (right). In both cases, 40 samples were used to estimate the illumination integral at each primary ray intersection point. Note that the BMC estimator yields a much smaller noise (i.e., error) compared to classic MC. This is in-line with the results shown in Figure 1.