ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

# Monte Carlo Integrator

Group2: Ehsan Abedi, Arash Gol-Mohammadi

In this file we have provided a brief instruction for the program, its compilation, features, limitations and the probable to-dos to eliminate them or extend the program.

# Introduction to Monte Carlo algorithm

Monte Carlo algorithm for integration is simply sampling over the domain of integration to estimate the mean value of the function. There are two main procedures to this problem: one is to uniformly choose samples in domain (uniform sampling) and the other is to choose the samples with respect to a weight function that have similar distribution to original function, so at the end, those regions of domain that have more contribution in the integral are considered more often (importance sampling).

# Compiling test suite

Inside *PCSC2017_Group2/tests/* there are four CPP files, each evaluating one of the features of Monte Carlo integration.

- *Test*: Evaluates three integral numerically with both methods (look at list of tests).
- *StatisticalMomenst:* Evaluates statistical moments of Gaussian distribution up to power 8 using uniform sampling. The answers will be written in a file. *Figure 1* shows the comparison between true and computed values. Note that although the weight is considered constant for different moments-which is not a good assumption- there is no significant difference between true values and our program estimations.
- *CentralLimitTheory*: Integrates the same function (for calculating $\pi$, using uniform sampling) many time to find the distribution of answers. *Figure 2* Shows the distribution of $N=10^5$ runs and the fitted normal distribution.
- *Errors*: Evaluates the true and estimated error for a specific function[1]. *Figure 3* shows a comparison between Uniform Sampling and Metropolis methods and their true error.

[1] This function is taken to be the function of test number 1 (which computes $\pi$)

# Program flow

For running the program, one should first specify the followings outside of the main block:

1. The function that should be integrated. It must get a double and returns a double
2. The strictly positive weight function, if Metropolis method is chosen, otherwise not needed.

Next step is to specify the method of integration by creating objects of the following classes inside the main block:

1. `MonteCarlo_UniformSampling`
2. `MonteCarlo_MetropolisAlgorithm`

After that, the user has to link the function and its weight to those defined outside the main by using the following methods:

1. `SetFunction(double (*f)(double x))`: The argument is the function defined outside of main
2. `SetWeight(double (*w)(double x), bool flag )` : The first argument is the function defined outside of main. The second argument is a flag indicating that if weight function is normalized (true) or not (false). If user is not sure about normalization, he/she can omit the second argument. By default, it is set to false to normalize the weight. Yet it is better to feed this method with normalized weight to avoid normalization error.

Then the user has to specify the upper and lower limit of integration, the moment (if it is desired), and the number of sampling by using the following methods:

1. `SetLowerLimit(const double a):` Specifies lower limit.
2. `SetUpperLimit(const double b)`: Specifies upper limit.
3. `SetMoment(const int m):` Specifies the moment number.
4. `SetSamplingNumber(const int N):` Specifies number of samples.

Finally, the method below, will return an array which contains the value of the integral in the first element and the evaluated error in the second one:

1. `Integrator():` computes integral with respective method

# List of features

1. Integrating $\mathbb{R} \to \mathbb{R}$ functions inside a bounded interval using Uniform Sampling and Importance Sampling (using Metropolis procedure)
2. Computing different statistical moments of a function (integer moments)
3. Normalizing weight function if needed. It will be done by default
4. Estimating the error of integration
5. Being able to increase the accuracy by taking more samples as user's wish

# List of tests

We have tested program under the following benchmarks[2].

1. Computing $\pi$:

$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$

2. Computing normal distribution integral:

$$\int_0^7 \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = 0.5$$

3. Computing gamma-function[3] for z=4 :

$$\Gamma(z) = \int_0^{20} x^{z-1} e^{-x} dx = (z-1)!$$

4. Evaluating error and comparing them with true error for all integrals above
5. Computing even-moments of normal distribution up to 10:

$$\int_0^{20} \frac{1}{\sqrt{2\pi}} x^m e^{-\frac{x^2}{2}} dx$$

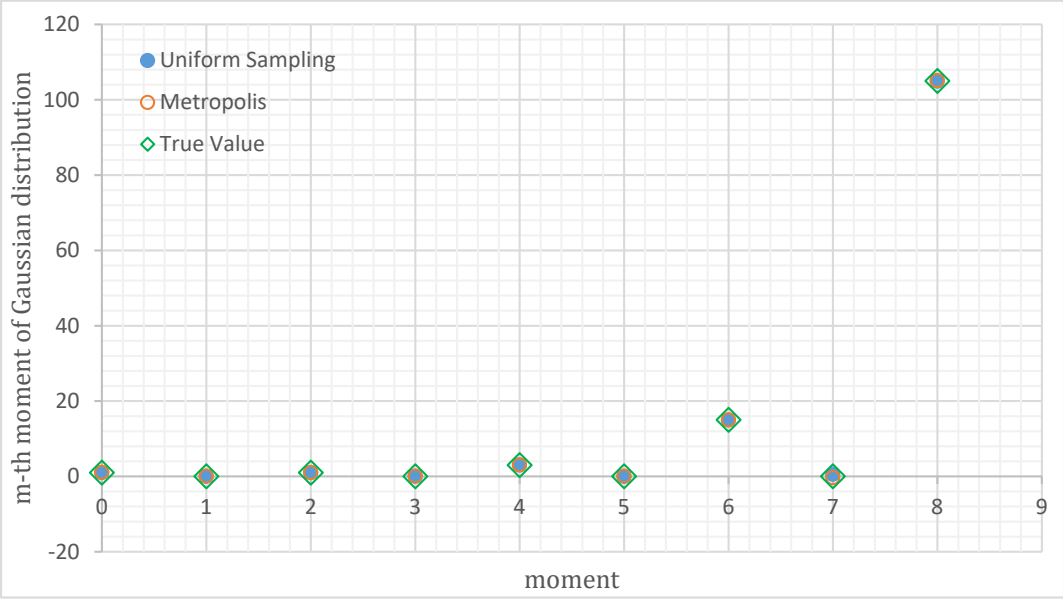
[2] For improper integrals, we have put our interval sufficiently large to tend to real answer.
[3] Which is equivalent to 4th moment of function $x^{-1}e^{-x}$

# Limitations, To-do and prospective

In this part, a number of program limitations have been mentioned, in each case we have suggested a way to improve the program:

1. The program can only integrate one variable functions. Using vectors, one may extend it for integrating functions with $\mathbb{R}^n \to \mathbb{R}^m$.
2. The program doesn't address the problem of improper functions, that is, the functions which at some points go to infinity, such as $f(x) = \frac{1}{x}$ at zero. One may add a warning method in such cases.
3. In Metropolis Algorithm, selecting a good step-size for moving from one point to another is very important for the accuracy of the result. We calculate the set size by multiplying a random number to a constant value. One can write self-corrector method and vary this value as the algorithm runs.
4. The integrals cannot be calculated over unbounded domain[4].

# Figures




[4] We've already provided such feature. Yet, we decided to comment it for the following reason. Using Monte Carlo algorithm on unbounded domain, is only possible with importance sampling (Metropolis) method. For using Metropolis, one should provide a suitable weight function considering the mathematical behavior of the original function. Also it is crucial to choose a good starting point to be able to span most of the subsets that contribute the most in the value of integral. These considerations cannot be implemented without pre-knowledge about shape of the function. In most cases, though, considering sufficient large upper and lower limit will give a reasonable answer.

*Figure 1*: Estimated m[th] moment of normal distribution

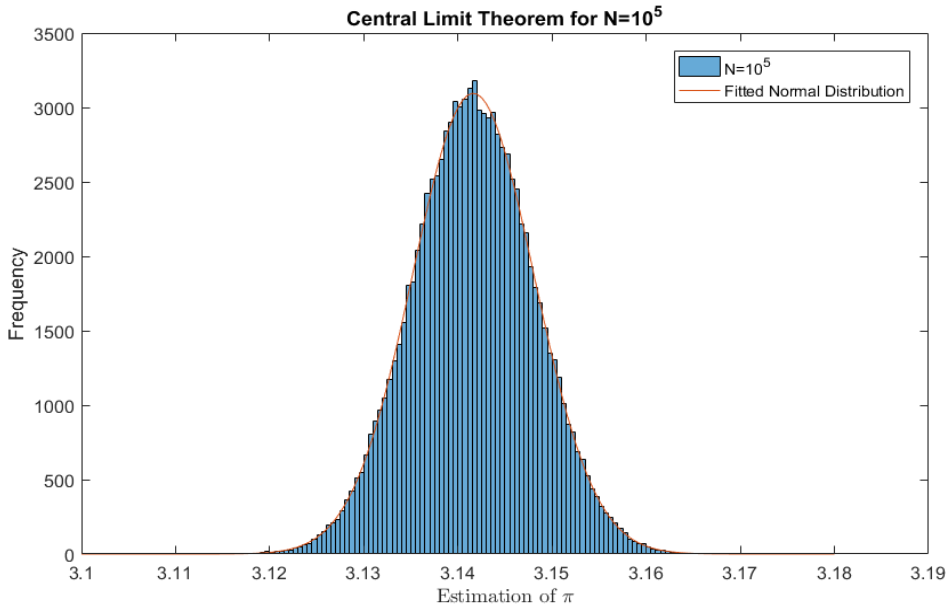



*Figure 2*: Distribution of estimated value of $\pi$ using uniform sampling

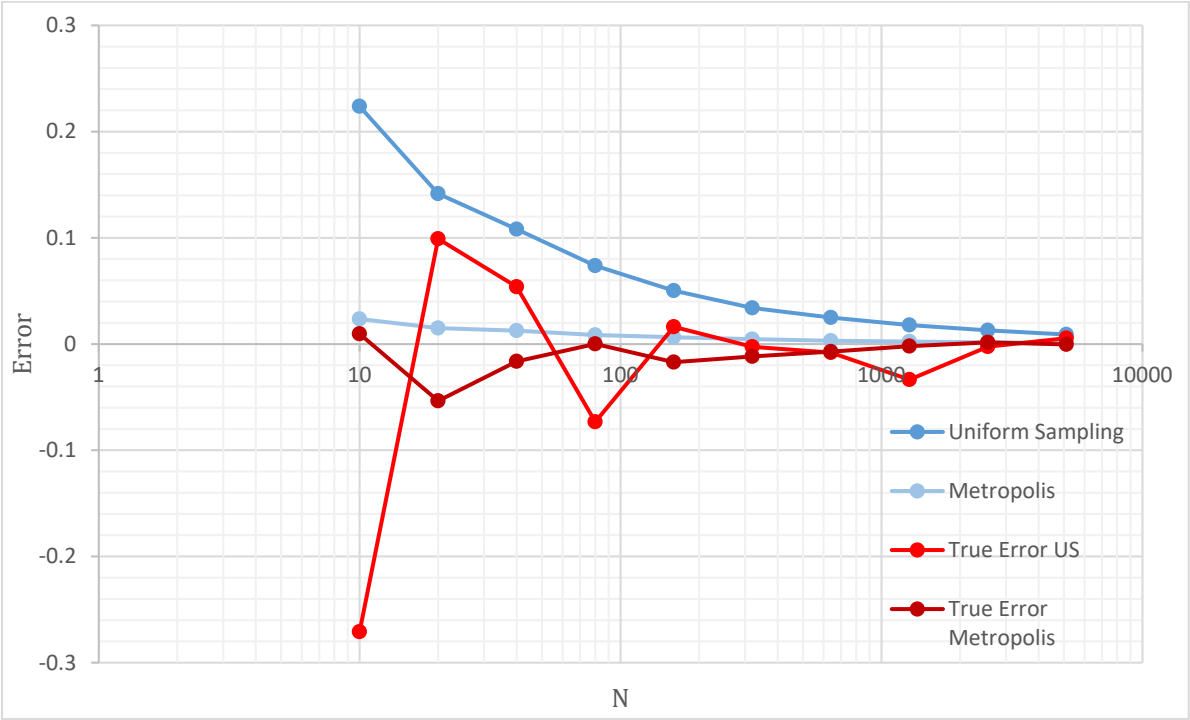



*Figure 3*: Estimated and true error of each method