MONTE CARLO

Monte Carlo methods are named after the famous casino in the ward Monte Carlo, Monaco, where you can gamble.

REUTERS/Eric Gaillard

Monte Carlo methods generally involve (pseudo)random numbers, similar to dice rolls, roulette or the random order due to shuffled cards in a deck. There are a lot of Monte Carlo methods available, with many books written on the subject. The only common characteristic is that they involve such numbers.

HOW TO GENERATE RANDOM VARIABLES?

RANDOM INTEGERS IN PYTHON

In python it is easy to generate **random integers**

```
import numpy as np
np.random.randint(6) # A random number between 0 and 5.
np.random.randint(6) # Another dice roll
np.random.randint(6) # Yet another
```

This will give three (pseudo)random numbers, for example 2, 5 and 0.

PSEUDO-RANDOM

The numbers are not entirely random, but pass (almost all) statistical tests.

It is therefore very similar to throwing a coin (which is actually also not really a true random event).

REPRODUCIBILITY

If you run the program again, you will get different (pseudo)random numbers.

Sometimes you want to generate *the same sequence of (pseudo)random numbers* (for example, for testing purposes). This can be achieved by a *seed*.

SEED

The random number generator can be given a **seed**. In python this is an integer between 0 and 4294967295. A particular seed will generate a deterministic sequence of (pseudo)random numbers. The exact sequence is uniquely determined by the value of the seed.

In python the seed can be set as follows

```
import numpy as np
np.random.seed(0)     # Here the seed is set to 0
np.random.randint(6) # A random number between 0 and 5.
np.random.randint(6) # Another dice roll
np.random.randint(6) # Yet another
```

This will always give 4, 5, and then 0 in Python.

A seed value of 1000 instead of 0 would have given 3 instead of 4 as the first dice roll.

Instead of a fixed seed, we can take the current time (in seconds). Then, if you run the program again you'll get a different sequence random numbers (drawback: does not work when using multiple processes at the same time).

UNIFORM DISTRIBUTION

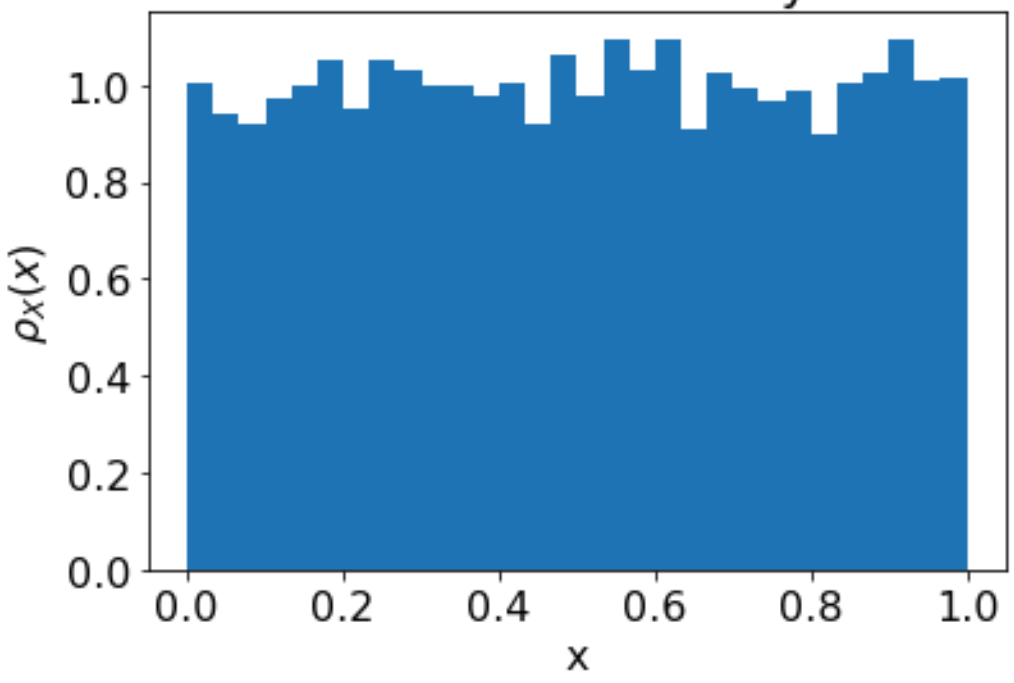
For Python a random number X from a *uniform* distribution in the range $[a, b]$ of reals can be obtained by using `numpy.random.uniform(a, b, 1)`.

EXAMPLE

To generate n uniformly distributed random variables X_i in the range $[x_{\min}, x_{\max}]$ in Python and plot a histogram of the results use:

```
import numpy as np
import matplotlib.pyplot as plt
x_min, x_max = 0, 1                    # Ends of the interval
n = 10000                              # Number of random numbers
X_all=np.random.uniform(x_min, x_max, n) # Generate the random numbers
X_i=X_all[0]
print(X_i)                             # Print the first random number
plt.hist(X_all, 30, density=True)      # Show a histogram of all the random
numbers
```

Histogram of 10000 random numbers uniformly distributed between 0 and 1



REGULAR INTEGRATION

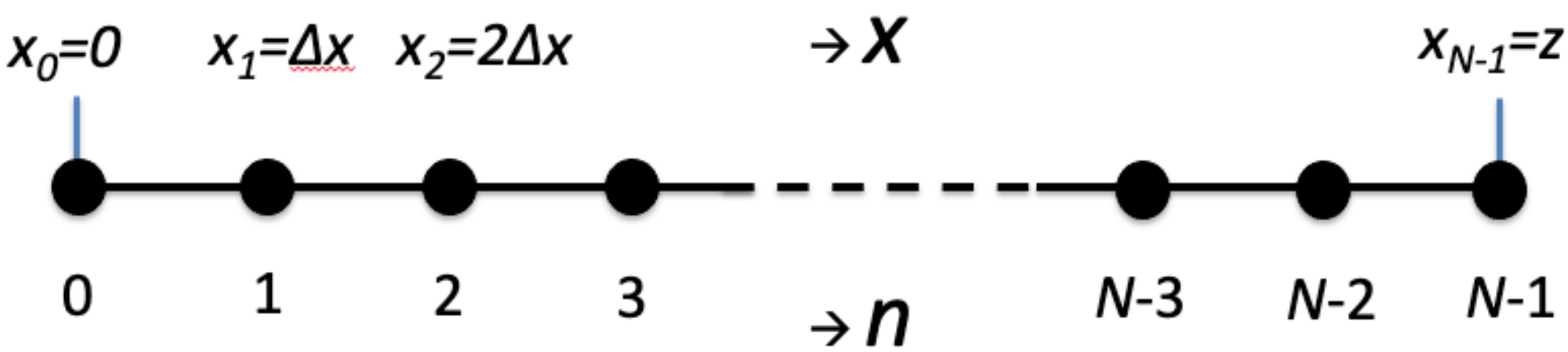
How to determine the integral of a function $f(x)$, such as in:

$$F(z) = \int_0^z f(x)dx$$

We can find the integral using forward/backward Euler (first order), Trapezoid, midpoint or Ralston's method (second order), or a higher order Runge-Kutta method such as RK4 (fourth order), by discretizing the interval for determining the solution to

$$\frac{dF(z)}{dz} = f(z)$$

with initial condition $F(0) = 0$.



MONTE CARLO INTEGRATION IN 1D

Alternative way is to use **Monte Carlo integration** to determine the integral.

$$F = \int_a^b f(x)dx$$

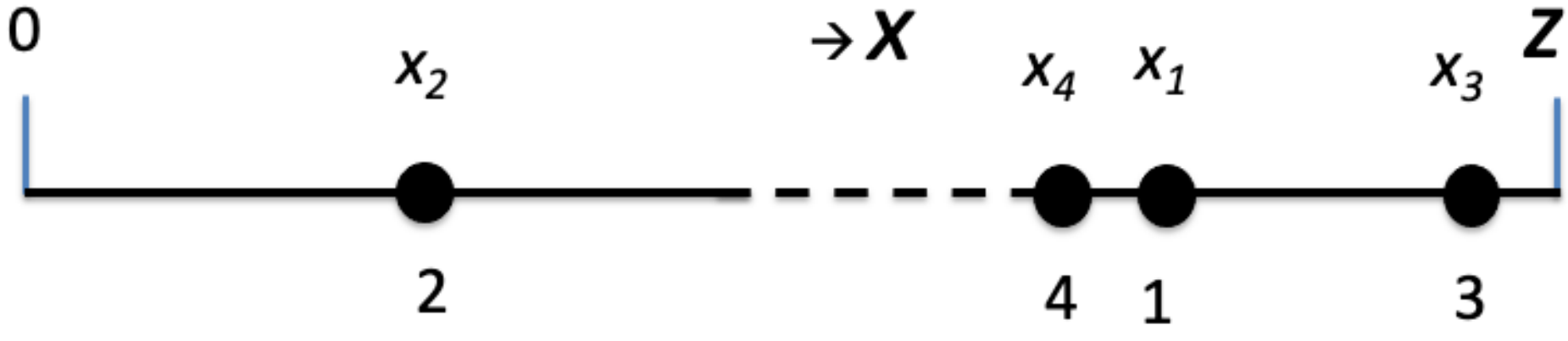
The idea is to calculate the average of the integrant first:

$$\langle f \rangle = \frac{\int_a^b f(x)dx}{b-a}$$

Then the integral is just the average of $\langle f \rangle$ multiplied by the interval $b - a$

$$F = (b-a)\langle f \rangle$$

ALGORITHM



Choose N random points X_n uniformly distibuted in the interval $[a, b]$, so $X_n \in [a, b]$, and $n = 1, 2, 3, \ldots, N$. The N-point *average value* of the function is

$$\langle f \rangle_N \equiv \frac{1}{N} \sum_{n=1}^{N} f(X_n)$$

The integral is the *average value* of f, $\langle f \rangle$, times the *length* of the interval,

$$L = b - a$$

Hence the estimate for the integral based on N points is

$$\boxed{F_N = L \langle f \rangle_N}$$

The estimate will approach the exact solution when the number of points goes to infinity:

$$F = \lim_{N \to \infty} F_N$$

since then the real average $\langle f \rangle$ will be obtained.

ERROR ESTIMATE MC INTEGRATION

A one-standard-deviation error estimate for the integral is related to the square root of the variance times the length of the interval,

$$\sigma_F = L\sqrt{\frac{1}{N}\left(\langle f^2\rangle_N - \langle f\rangle_N^2\right)}$$

Here f^2 is defined as squaring the function value *before* taking the average $\langle\cdots\rangle$.

This results in the following estimate of the value of the integral with 1 standard deviation error bounds

$$F \approx L\left(\langle f\rangle_N \pm \sqrt{\frac{1}{N}\left(\langle f^2\rangle_N - \langle f\rangle_N^2\right)}\right)$$

GENERALIZATION TO HIGHER DIMENSIONS

The method can easily be generalized to higher dimensions, say D dimensional

$$F = \int f d^D x$$

Let us denote the (hyper)volume of the integration region Ω by $V = \int_\Omega d^D x$. The integral can be determined by replacing the aforementioned L by V, as long as all the points X_n are uniformly distributed inside this region Ω.

For a 2-dimensional integral this would become

$$F = \int_{y_{\min}}^{y_{\max}} \int_{x_{\min}}^{x_{\max}} f(x,y) dx dy$$

with the 2D volume given by

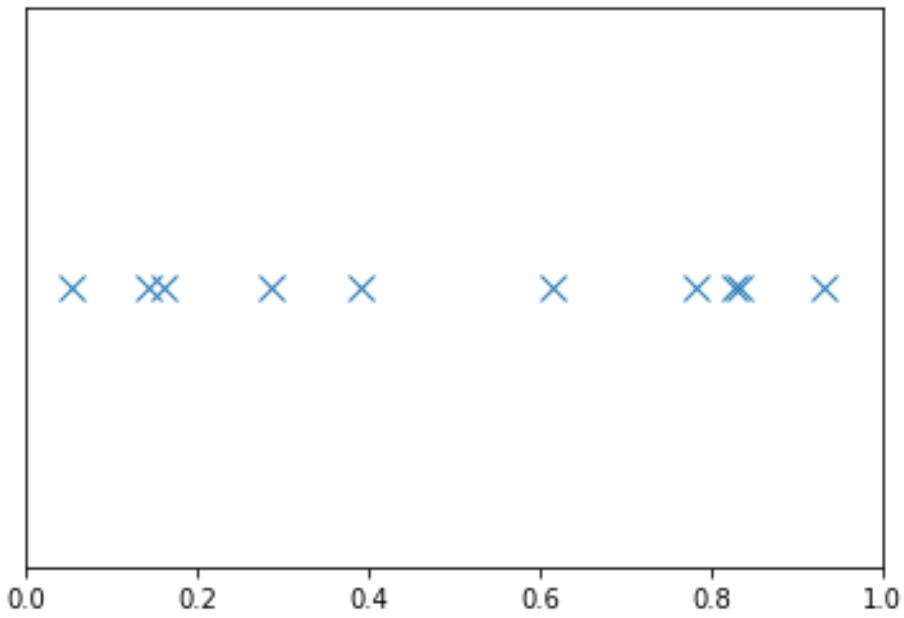
$$V = \int_{y_{\min}}^{y_{\max}} \int_{x_{\min}}^{x_{\max}} dx dy = (x_{\max} - x_{\min})(y_{\max} - y_{\min})$$

EXAMPLE

10 random points, uniformly distributed between 0 and 1.

EXAMPLE

10 random points, uniformly distributed between 0 and 1.



INTEGRATING $f(x) = x^2$ FROM 0 TO 1

Python code:

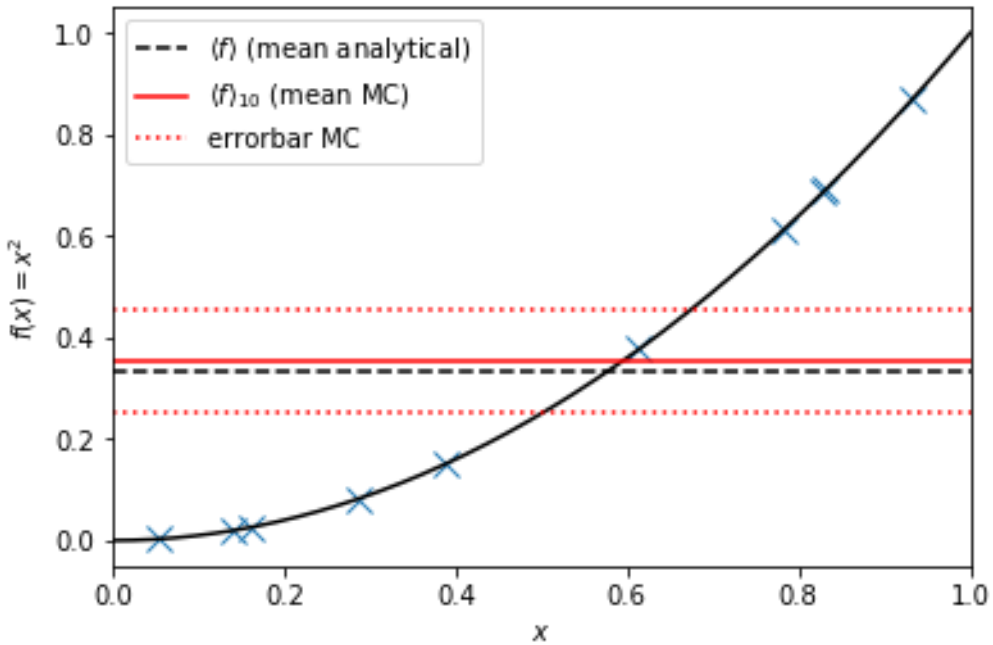
```
import numpy as np
# Function to integrate
def f(x):
    return x**2

# Generate a vector containing the random numbers from a uniform distribution
a, b = 0, 1
nsamples = 10
X=np.random.uniform(a, b, nsamples)

# f(X): value of integral for each vector element
# The mean value is then
mean_f=np.mean(f(X))
# Value of the integral
F=(b-a)*mean_f
```

Alternatively, you could have a 'for' loop to go over every element.

Analytical result: $F = \frac{1}{3}x^3\Big|_0^1 = 1/3.$


⟨f⟩ (mean analytical)
⟨f⟩10 (mean MC)
errorbar MC
f(x) = x²
x
0.0
0.2
0.4
0.6
0.8
1.0

EFFICIENCY AND ORDER OF THE METHOD

What is the order of the MC method?

- Euler: first order, $O(\Delta x)$ (global) error, or $O(1/N)$.
- RK4: fourth order, $O(\Delta x^4)$ (global) error, or $O(1/N^4)$.

It turns out that the MC method is $O(1/N^{1/2})$, or 'half' order in the average spacing $\langle \Delta x \rangle = (a-b)/N$, $O(\langle \Delta x \rangle^{1/2})$.

Much worse than other methods?!

EXAMPLE

Consider the function

$$f(\vec{x}) = 1 - \vec{x} \cdot \vec{x}$$

Let us integrate this from -4 to 4 in every dimension in a D dimensional space. Hence $V = 8^D$. The integral to be determined is therefore

$$\left(\int_{-4}^{4} \right)^D (1 - \vec{x} \cdot \vec{x})\, d^D x$$

As an example, for 2 dimensions the integral reduces to

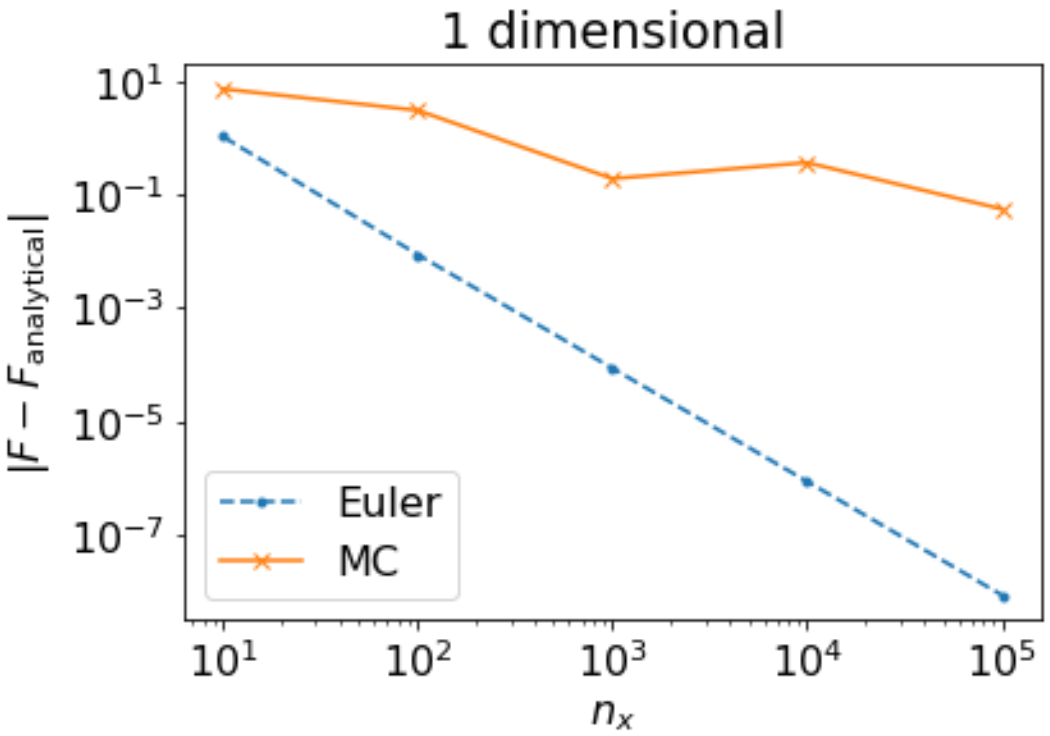
$$\int_{-4}^{4}\int_{-4}^{4}\left(1-(x^2+y^2)\right)dxdy$$

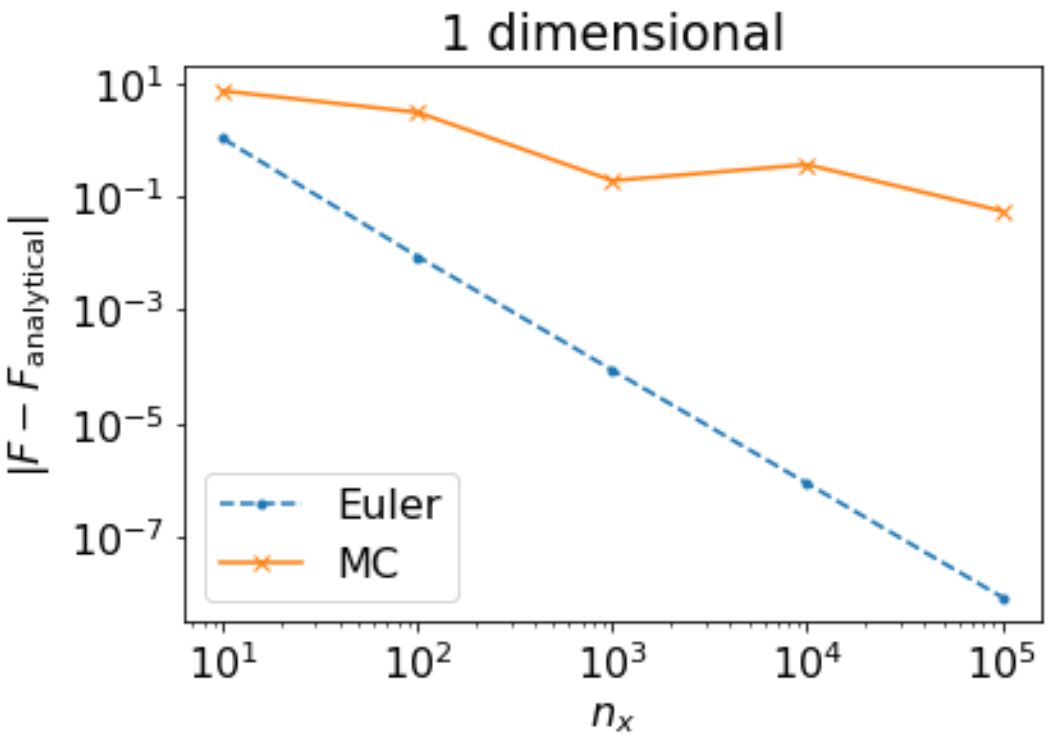
and $V = 8^2 = 64$.

For the Euler integration method we take n_x points per dimension, so total number of points is $N = n_x^D$, where D is the dimension.

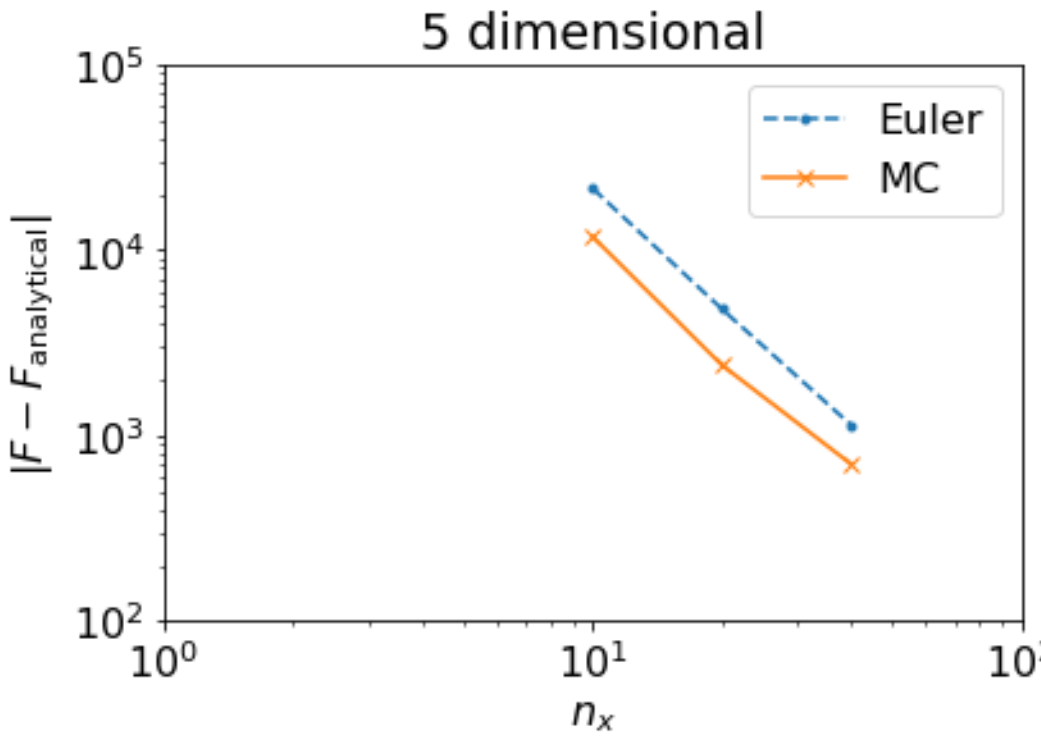
To allow for fair comparision: same amount of points for the MC integration. Hence we will take for MC: $N = n_x^D$ samples.

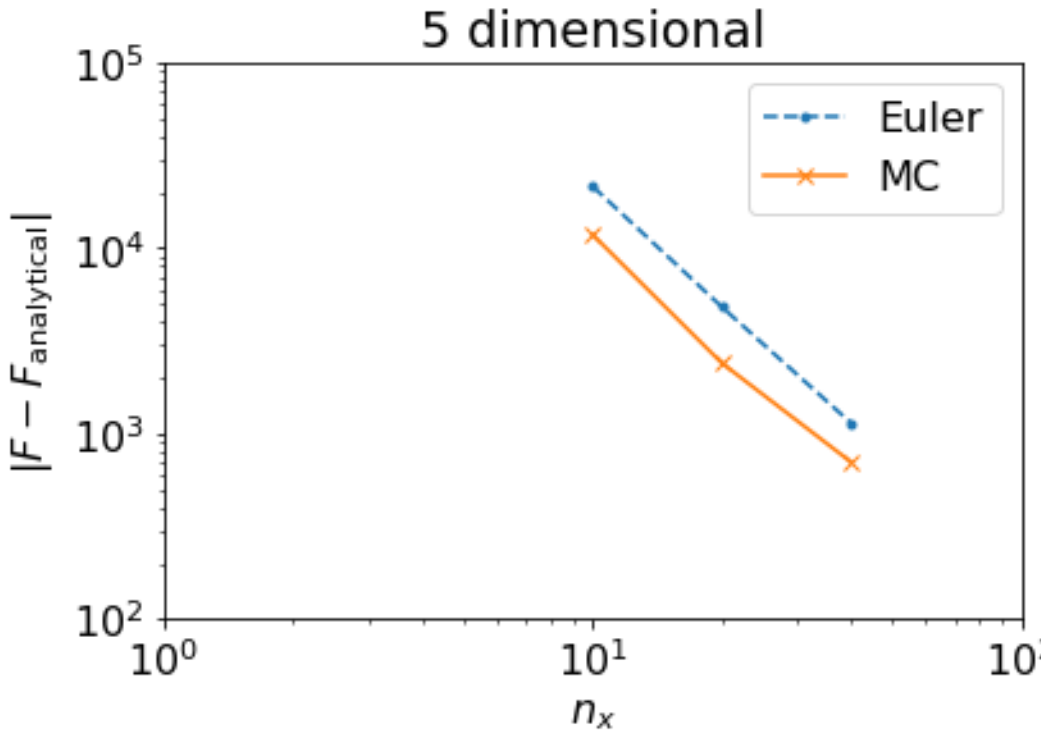
For the MC integration we need to generate N random D-dimensional vectors $\vec{X}_n$, which can be established by generating D random numbers per vector, each uniformly distributed between -4 and 4. In total we need $D \cdot N$ random numbers.


1 dimensional
10^1
10^{-1}
10^{-3}
10^{-5}
10^{-7}
$|F - F_{\text{analytical}}|$
Euler
MC
10^1
10^2
10^3
10^4
10^5
n_x



For the one-dimenional integral the Euler algorithm is much more efficient than MC; after 100000 points the integral is accurate up to 10^{-8}. The MC algorithm is only accurate up to 10^{-1}.


5 dimensional
Euler
MC
$|F - F_{\text{analytical}}|$
10^5
10^4
10^3
10^2
10^0
10^1
10^2
n_x



For 5 dimensions it looks very different. The error for the MC integration is now less than for the Euler integration.

MC integration becomes competitive for higher dimensions!

EXERCISES SESSION 7

1. What is the Monte Carlo method, and what is the order of the Monte Carlo method?

2. See Assessment section on Blackboard.