MC Integration: Rationale

Suppose that we wish to approximate some integral I using simulations. I has the form

$$I = \int g(x)f(x)dx$$

where $g(x)$ is some function of a continuous r.v. X and $f(x)$ is its density. We can draw a sample of size m from the density $f(x)$ and then use the empirical average to get a Monte Carlo estimate of I:

$$\hat{I} = \frac{1}{m}\sum_{i=1}^{m}g(x_i)$$

The standard error $\hat{se}(\hat{I})$ is given by

$$\hat{se}(\hat{I}) = \sqrt{\frac{1}{m^2}\sum_{i=1}^{m}\left(g(x_i) - \hat{I}\right)^2}$$

MC Confidence Intervals

Clearly, we see that the larger the number of simulations, the lower the standard error. Indeed, by the Law of Large Numbers (LLN), we have that

$$\lim_{m \to \infty} \frac{1}{m} \sum_{i=1}^{m} g(x_i) = E_f[g(x)]$$

or equivalently $\lim_{m \to \infty} \hat{I} = I$. Then, by the Central Limit Theorem (CLT), we have that $\hat{I} \to N(I, se(I))$ and the quantity $\frac{\hat{I} - I}{\hat{se}(\hat{I})} \sim N(0, 1)$.

We can then construct a $(1 - \alpha)$ Confidence Interval for I, given by

$$\left[\hat{I} - z_{1-\alpha/2} \hat{se}(\hat{I}), \hat{I} + z_{1-\alpha/2} \hat{se}(\hat{I}) \right]$$

where $z_{1-\alpha/2}$ is the $1 - \alpha/2$ quantile of the Normal distribution.

Working example (1/2)

Suppose that we wish to approximate the following integral

$$I = \int_0^\pi g(x)dx \qquad \text{with } g(x) = sin(x) + cos(x)$$

We can compute it analytically and we will find that $I = 2$. Using Monte Carlo integration, we will generate $m = 10,000$ realizations of $X \sim U[0, \pi]$ and then use the fact that the PDF of X is

$$f(x) = \begin{cases} 1/(\pi - 0) & \text{if } x \in [0, \pi], \\ 0 & \text{otherwise} \end{cases}$$

In our case, we will have

$$I = \int_0^\pi g(x)dx = \pi \int_0^\pi g(x)\frac{1}{\pi}dx = \pi E_f[g(x)]$$

Working example (2/2)

We will implement a MC integration procedure in R and Python to approximate the integral of interest and its standard error:

$$\hat{I} = \frac{\pi}{10,000} \sum_{i=1}^{10,000} g(x_i)$$

$$\hat{se}(\hat{I}) = \frac{\pi}{10,000} \sqrt{\sum_{i=1}^{10,000} \left(g(x_i) - \hat{I} \right)^2}$$

95%-Confidence Intervals spoiler alert: The results are obviously equivalent. These are the obtained output in R and Python respectively, rounded to 5 decimal places and using the same seed.

$$I \in [1.949382.145387]$$

$$I \in [1.91469, 2.10769]$$

MC integration with R

```r
# 1. Define the function g() that we wish to integrate
g = function(x) {sin(x) + cos(x)}

# 2. Perform the MC approximation using m = 10,000 simulations
set.seed(2023)

m = 10000
fx = runif(m, 0, pi)

# MC estimate of the integral
I_hat = ((pi - 0) / m) * sum(g(fx))
# [1] 2.047382

# approximation of the standard error of the MC estimate
se_I_hat = (pi / m) * sqrt(sum((g(fx) - I_hat)^2))
# [1] 0.05000365

# 95% Confidence Interval

c(I_hat - qnorm(1 - 0.05/2) * se_I_hat, I_hat + qnorm(1 - 0.05/2) * se_I_hat)
# [1] 1.949377 2.145387
```

Plots in R

Function to integrate and convergence of MC estimate
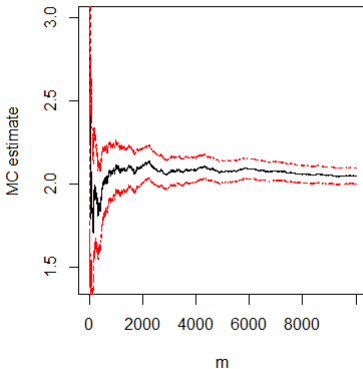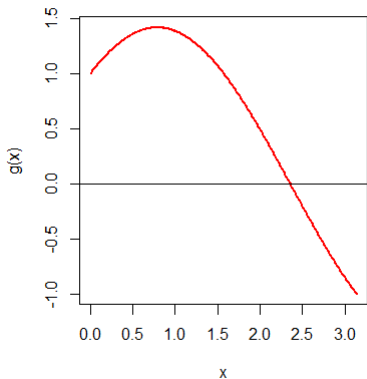
MC integration with Python

```
1  # 1. Define the function g() that we wish to integrate
2  import math
3  import numpy as np
4
5  def g(m):
6      x = np.array(np.random.uniform(low = 0, high = math.pi, size = m), dtype =
            float)
7      g = np.sin(x) + np.cos(x)
8      return g
9
10
11 # 2. Perform the MC approximation using m = 10,000 simulations
12 np.random.seed(2023)
13 k = 10000
14
15 # MC estimate of the integral
16 I_hat = ((math.pi - 0) / k) * np.sum(g(m = k))
17 # 2.0111916600488873
18
19 # approximation of the standard error of the MC estimate
20 se_I_hat = (math.pi / k) * np.sqrt(np.sum((g(m = k) - I_hat)**2))
21 # 0.04923566222812938
22
23 # 95% Confidence Interval
24 import scipy.stats
25
26 [I_hat - scipy.stats.norm.ppf(1 - 0.05/2) * se_I_hat, I_hat + scipy.stats.norm.
        ppf(1 - 0.05/2) * se_I_hat]
27 # [1.9146915353267746, 2.107691784771]
```

Plots in Python

Function to integrate and convergence of MC estimate
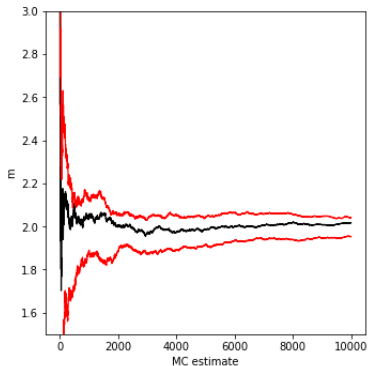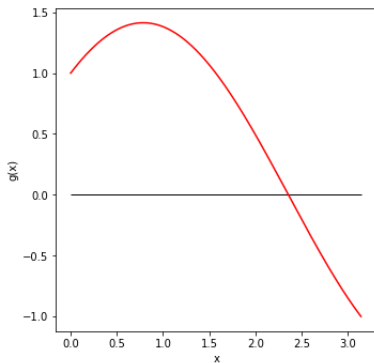
MC simulations with R

To tackle the same problem using what we call MC simulations, in our example that we could call 'MC simulation for MC integration', the idea is to introduce n, the number of simulations in addition to m, the number of replications or realizations. Obviously the process is slower but we do it to illustrate what could be an example of MC simulations.

```
1  # 1. MC simulations (using a for loop)
2  # Using MC simulation and MC integration
3
4  set.seed(12023)
5  n = 10000
6  m = 5000
7  est = numeric(n)
8
9  for(i in 1:n) {
10    x = runif(m, 0, pi)
11    g = sin(x) + cos(x)
12    est[i] = (((pi - 0) / m) * sum(g))
13 }
14
15 I_hat = mean(est)
16 I_hat
17 # [1] 1.99988
```

Histogram and Density in R

Histogram and kernel density estimator of the MC estimates
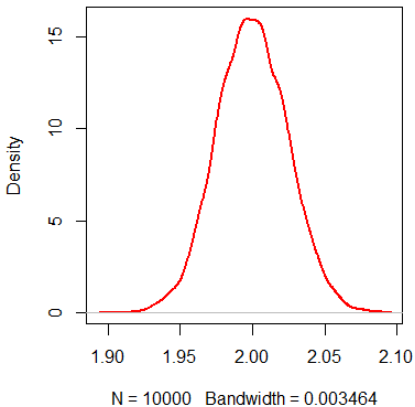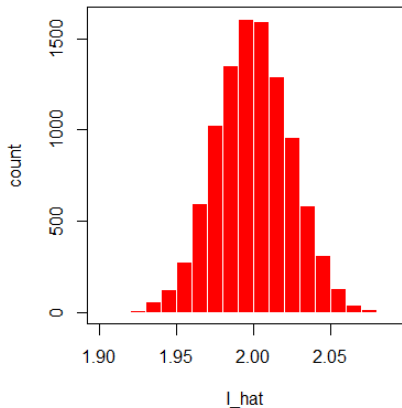
MC simulations with Python

We repeat the exact same procedure, this time in Python. Obviously
the code can be different and optimized, the idea is to use a for loop.

```python
1  # 1. MC simulations (simple for loop)
2  # Using MC simulation and MC integration
3
4  import statistics
5  np.random.seed(2023)
6
7  n = 10000
8  m = 5000
9  est = []
10 I_hat = []
11
12 for i in range(n):
13     g = []
14     x = np.random.uniform(low = 0, high = math.pi, size = m)
15     g = (np.sin(x) + np.cos(x))
16     # MC estimate of the integral
17     est = (((math.pi - 0) / m) * np.sum(g))
18     I_hat.append(est)
19
20 statistics.mean(I_hat)
21 # 2.0002932056229525
```

Histogram and Density in Python

Histogram and kernel density estimator of the MC estimates
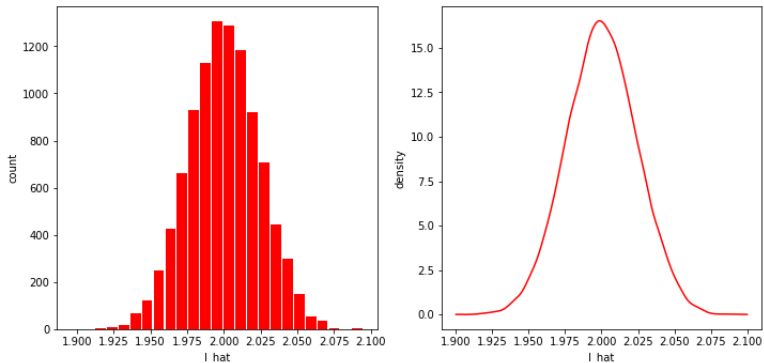
MC simulations with R: advanced example for the same result

```
1  # 3. Advanced example of MC simulations: nested for loop
2  # MC simulations (example taking about 3 min to run)
3  # for MC integration
4
5  # Define the function g() that we wish to integrate
6  g = function(x) {sin(x) + cos(x)}
7
8  # 4. Define storage objects and erform the MC approximation
9  # using m = 500 replications and n = 10,000 simulations.
10 set.seed(2023)
11
12 n = 10000
13 I_hat = numeric(n)
14 m = 500
15 est = matrix(m*n, nrow = n, ncol = m)
16
17 for(i in 1:n) {
18   for(j in 1:m) {
19
20     # Sum on all rows then we take the mean to get our final estimate
21     est[i, j] = ((pi - 0) / m) * sum(g(runif(j, 0, pi))[j])
22   }
23   # MC estimate of the integral
24   I_hat[i] = sum(est[i,])
25 }
26
27 mean(I_hat)
28 # [1] 2.000983
```

Behavior of the MC estimates in R

Behavior of the MC estimates of I
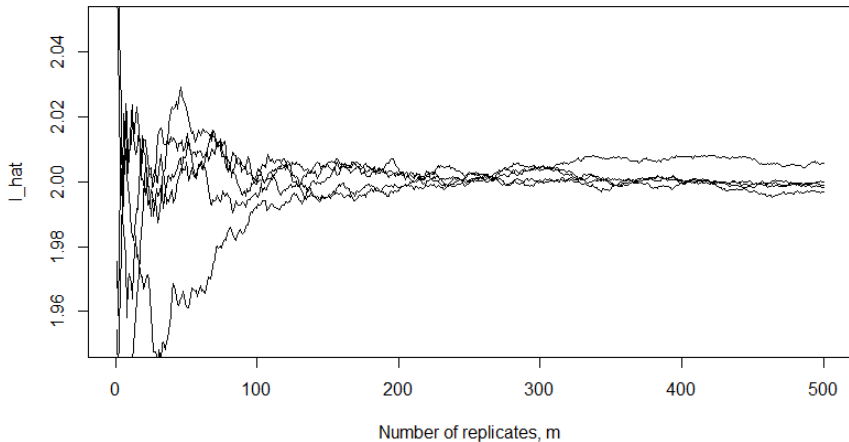
Conclusion from R and Python outputs

Setting: The integral to approximate is $I = \int_0^\pi g(x)dx$, with $g(x) = sin(x) + cos(x)$. Using $m = 10,000$ realizations and with $f(x) \sim U[0, \pi]$, the estimate $\hat{I}$ of I is about 2, which is the analytical answer. When computing 95%-Confidence Intervals, we get:

In R:

```r
# 95% Confidence Interval
c(I_hat - qnorm(1 - 0.05/2) * se_I_hat, I_hat + qnorm(1 - 0.05/2) * se_I_hat)
# [1] 1.949377 2.145387
```

In Python:

```python
# 95% Confidence Interval
[I_hat - scipy.stats.norm.ppf(1 - 0.05/2) * se_I_hat, I_hat + scipy.stats.norm.ppf(1 - 0.05/2) * se_I_hat]
# [1.9146915353267746, 2.107691784771]
```

For the same number of replications m and provided that the MC estimates $\hat{I}$ and $\hat{se}(\hat{I})$ are correctly computed, the results are obviously equivalent using both programming languages.

Further reading and code

Rizzo, M.L. (2019). Statistical Computing with R, Second Edition (2nd ed.). Chapman and Hall/CRC.
https://doi.org/10.1201/9780429192760

The R Project for Statistical Computing:
https://www.r-project.org/

Python:
https://www.python.org/

Accessing R and Python code:
https://github.com/JRigh/MC-Integration-and-simulations-in-R-and-Python